Shuffled Frog Leaping Algorithm for Hardware/Software Partitioning

Jiayi Du[a], Xiangsheng Kong[b], Xin Zuo**[c], Lingyan Zhang[d], Aijia Ouyang[e,f],
[a] School of Information Science and Engineering, Hunan Provincial Key Laboratory of Embedded and Network Computing, Hunan University, Changsha 410082, China
Email: maxdujiayi@hnu.edu.cn
[b] Department of Computer Engineering, Xinxiang University, Xinxiang, Henan, 453003, China
Email:fallsoft@163.com
[c] Office of Information Department, Hunan University, Changsha 410082, China
Email: zuoxin@hnu.edu.cn
[d] College of Information Science and Technology, Jinan University, Guangzhou 510632, China
Email: jnuzly@163.com@163.com
[e] College of Computer, Hunan Institute of Traffic Engineering, Hengyang 421001, Hunan, China
[f] School of Information Science and Engineering, Hunan City University, Yiyang, Hunan 413000, China
Email: ouyangaijia@163.com

Abstract— **Reconfigurable system on chip is well known for its flexibility for high performance embedded systems. The hardware/software (HW/SW) partitioning is the most important phase during the design of reconfigurable system on chip. A great many different algorithms have been adopted for solving the hardware/software partitioning problem. Shuffled Frog Leaping Algorithm (SFLA) is popular for its simple concepts, little parameter adjustment, high calculation speed, strong global search optimization capability and easy execution. In this paper, we apply the SFLA algorithm to solving hardware/software partitioning problem on reconfigurable system on chip with coarse-grained.**

The experimental results show that the SFLA algorithm can reduce the time cost by 45.54% on average with three different area constrain, compared with greedy algorithm. The time cost of SFLA algorithm are also reduced by 23.57% and 9.99% on average with simulated annealing algorithm (SA) and combined algorithm with greedy and simulated annealing algorithm (GSA). When area constrain is a half of area cost which all tasks are implemented by hardware will be taken, SFLA algorithm can reduce the time cost by 51.30%, 21.04% and 11.61% on average, compared with that of Greedy algorithm, SA, and GSA, respectively.

Index Terms— **Reconfigurable System on Chip, Hardware/Software Partitioning, Shuffled Frog Leaping Algorithm, Coarse-grained, Greedy Algorithm, Simulated Annealing Algorithm, Combined Algorithm**

I. INTRODUCTION

With the development of microelectronics and computer technology, especially large-scale emergence of FPGA programmable devices, real-time circuit remodeling ideas have attracted increasing attention of researchers. In reconfigurable systems, hardware information (configuration information of programmable devices) can be the same as the software program. This will not only keep the performance of computing hardware, but also the flexibility of the software.

HW/SW co-design is popular in designing the embedded system. HW/SW partitioning is the most important phase during the HW/SW co-design. As far as we know, HW/SW partitioning problem is a typical optimization problem. If we only take into consideration one cost of the embedded system, the HW/SW is simply a single optimization. Unfortunately, in the real life, the embedded system puts strict requirements on size, power consumption, time consumption and reliability. Hence, it becomes a multiobjective optimization problem. Furthermore the multiobjective optimization problem is a NP-complete problem, which we can not find a polynomial time algorithm to solve it. For this reason, HW/SW partitioning problem is a NP-complete problem. A lot of researchers have focused on this problem.

Generally speaking, HW/SW partitioning problem involves two scenarios: namely static HW/SW partitioning and dynamic HW/SW partitioning. Static HW/SW partitioning, refers to that the functional unit is mapped to the hardware or software is decided during the design of embedded system. Once the design of embedded system is completed, the implementation of the functional unit can not be changed. The emergence of reconfigurable technology allows us to use the dynamic HW/SW partitioning algorithm for dealing with the HW/SW partitioning problem. We can change the implementation of each function unit in runtime with the dynamic partitioning technology. Dynamic HW/SW partitioning is flexible for the embedded system. However, one of its main drawback is that, if the partitioning algorithm performance happens to be less than satisfactory, the performance of the whole embedded system will be poor. Against this backdrop, we usually use the Static HW/SW Partitioning for large-scale partitioning problems.

**Corresponding Author: Xin Zuo Email: zuoxin@hnu.edu.cn

doi:10.4304/jcp.9.11.2752-2760

In this paper, our problem for large-scale task embedded system, so the static partitioning is adopted. In particular, we use the SFLA algorithm for HW/SW partitioning with the area constrain. SFLA algorithm is popular for its simple concepts, little parameter adjustment, fast calculation speed, strong global search optimization capability and easy execution. The main contributions of our paper are as follows:

- We have been the first to apply SFLA algorithm to solving HW/SW partitioning problem.
- The value of our object function is the completion time of critical path in the DAG. It is more reasonable in the real reconfigurable system on chip.
- We compare three algorithms with SFLA algorithm, which include greedy algorithm, simulated annealing algorithm and the algorithm combining greedy and simulated annealing algorithm.
- Our algorithm is compared with other algorithms in three different area constrains.

The rest of this paper is organized as follows. In the section II, the research background and previous related work are introduced. We outline the hardware model, compute model and problem definition in Section III. It then moves to detail how the shuffled frog leaping algorithm is applied to solving HW/SW partitioning problem, as seen, in the Section IV. Experimental are presented in the section V. The conclusions of our paper are presented in Section VI.

II. Related Works

A number of researchers have carried research on models of HW/SW partitioning problem. For automate model the partitioning, Adhipathi proposes a novel strategy, Process Model Graphs. He simulates and verifies this strategy in his paper [1]. Sapienza et al. propose the meta model for the HW/SW partitioning problem. The meta model enables use for the partitioning and reuse [2]. However, they do not apply algorithm to solving the HW/SW partitioning problem.

Some researchers focus on the partitioning algorithm for the HW/SW partitioning problem. Niemann et al. were the first to propose using integer programming (IP) to solve HW/SW partitioning problem. The merger programming can obtain the optimal solution, but it yields poor performance when solving the large scale problem [3]. Resano et al. consider the system performance constrain. They propose a strategy for HW/SW partitioning and task scheduling to reduce the energy consumption [4]. In [5]. Abdelhalim et al. combine several swarm intelligence algorithm for the unconstrained design problem in embedded system. According to them, the particle swarm optimization algorithm followed by genetic algorithm, rather than other sequences, can obtain the best results than other sequence [6]. Arato et al. propose a heuristic algorithm which is a polynomial-time algorithm for the HW/SW partitioning problem. Guo et al. focus on the automated HW/SW partitioning problem. Their target is real-time operating system in the SoC. They propose a discrete Hopfield neural network algorithm to solve this problem [7].

In 2006, Kaizhong et al. model HW/SW partitioning to a 0-1 model over IP cores. This algorithm is called 0-1 algorithm, which makes fully use of the IP core and yields efficient partitioning [8].

In 2007, Mann et al. and Farmahini et al. respectively propose a new algorithm [9], based on branch and bound algorithm, and particle swarm optimization algorithm [10].n order to improve the reuse functionalities of system, Arunachalam et al. propose the genetic algorithm. In this case the system has requirement in functionalities' concurrency and cost constraints [11]. Liu et al. use improved directed acyclic graph to model the HW/SW partitioning problem, and subsequently make use of the immune algorithm to solve it [12]. Further-more, concerning the multi-objective optimization on HW/SW partitioning problem, they propose an immune algorithm to obtain the parieto optimal solution [13].

In 2013, Pando et al. take a fuzzy approach to solve the HW/SW partitioning problem. This approach is attractive for its flexibility [14]. For special application, sensorless current controller, Bahri et al. put forward a non-dominated sorting genetic algorithm with regards to the HW/SW partitioning. This algorithm can obtain the optimal solution [15]. In [16], a reliable delay estimation approach is proposed by Hansan et al. In [17], Han et al. propose a heuristic solution for scheduling and partitioning on multi-processor system on chips (MPSOC). Wu et al. come up with two methods to solve the HW/SW partitioning problem. This is firstly done by transforming the partitioning problem to an extended 0-1 knapsack problem, which will then be solve it by the heuristic one generated by the first method [18]. Sha, et al focus on the HW/SW partitioning problem on MPSoC. They use a dynamic programming method to minimize the system's power consumption under time and area constraints, and propose an optimal algorithm and a heuristic one for tree-structured inputs and DAG, respectively [19]. As seen from the literature, the researchers propose their own algorithms for different target systems with different grain. However, they do not use the SFLA algorithm to solve the HW/SW partitioning problem.

A number of researchers focus on using the SFLA algorithm to solve the optimization algorithm. SFLA algorithm is popular for its simple concepts, little parameter adjustment, high calculation speed, strong global search optimization capability and easy execution. In [20], Elbeltagi et al. compare memetic algorithm (MA), ant colony algorithm (ACO), genetic algorithms (GA), and SFLA algorithm. In [21], Horng et al. propose the maximum entropy based on SFLA algorithm thresholds (MESFLOT) method. They use the MESFLOT algorithm to multilevel image threshold selection. For the economic load dispatch problem, Roy et al. combine the genetic algorithm with improved shuffled frog leaping algorithm (MSFLA). The MSFLA algorithm can obtain a better solution in this instance. [22]. Xiao et al. apply an

improved shuffled frog leaping algorithm (MSFL) for simulation capability scheduling problem in cloud simulation platform. The simulation capability scheduling problem has a number of multimode constraints [23]. Xu et al. propose an improved SFLA algorithm to solve the hybrid flowshop problem with multiprocessor tasks [24]. In the multidepots vehicle routing problems, Luo et al. bing up an improved SFLA. [25]. However, their technique does not consider using SFLA algorithm for HW/SW partitioning problem, either. In this paper, we apply the SFLA algorithm to solving the HW/SW partitioning problem on reconfigurable system on chip with coarse-grained.

III. MODELS

In this section, we will introduce the hardware architecture model and computing model. The architecture of reconfigurable system on chip will be touched upon, which will be followed by an introduction of the Task Data Flow Graph (TDFG) model for HW/SW partitioning problem.

A. Hardware model

Our target architecture of hardware architecture is shown in Figure 1. The embedded system consists of CPU and reconfigurable unit. The reconfigurable unit employs Field Programmable Gates Arrays (FPGA). While the main memory uses the non-volatile memory. The Reconfigurable unit and CPU access each other via the share bus and the data is loaded to main memory by bus and will be stored there since then. Non-volatile memories (NVMs) are characterized by low energy consumption, low cost, and high density. Therefore, NVMs likely to be used as the main memory instead of DRAM. For our future work in task schedule on NVMs, we use the NVMs as main memory.

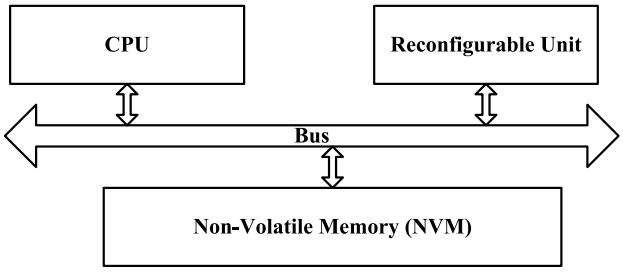

Figure 1. Hardware architecure

B. Computing model

We take the Task Data Flow Graph (TDFG) to model the HW/SW partitioning problem. Hence, the grain of partitioning is on task level. The TDFG graph $G =< V, E, TCost, ACost, X >$ is a node-weighted directed acyclic graph (DAG). A node set $V = \{v_1, v_2, v_3, ..., v_n\}$ represents a set of tasks. A set $E \subset V \times V$ represents dependency relations among tasks. $TCosth_i$ represents the execution time of task $v_i \in V$ on the hardware, $TCosts_i$ represents the execution time of task $v_i \in V$ on the software. $ACost$ represents the area cost of task. When the task is implemented by software, the value of $ACost$ is 0. X is a binary set, which contains two elements, 0 and 1. When the task is implemented by software, X is 0, while if the task is implemented by hardware, X is 1. Our model of HW/SW partitioning problem is based on the task level. Because we take the Coarse-grained to HW/SW partitioning problem, the communication time between CPU and Reconfigurable Unit is taken into consideration.

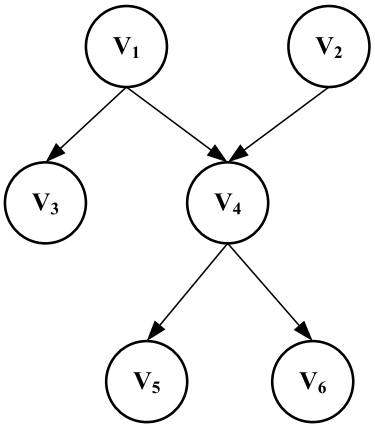

Figure 2. Compute model

IV. HARDWARE/SOFTWARE PARTITIONING PROBLEM

In this section, we describe the SFLA algorithm in details in the beginning, such as Algorithm parameters, Updating strategy, and so on. Then we introduce in detail the SFLA algorithm which is adopted to solve the HW/SW partitioning problem.

A. SFLA description

SFLA is a new evolutionary heuristic algorithm with high-performance computing and excellent global search capability. After outlining the basic principles of mixed SFLA, we have come up with an updated SFLA based on threshold selection strategy. This can solve the problem of significant changes in individual spatial location and reduce the speed of convergence caused by local updating operation. The strategy does not update those individuals which do not meet the threshold conditions, thereby reducing the individual spatial differences and improving the performance of the algorithm. Numerical experiments have demonstrated the effectiveness of the improved algorithm and determined the threshold parameters of the algorithm. SFLA was proposed by Eusuff and Lansey in 2003 in order to solve combinatorial optimization problems [26]. As a new type of artificial intelligence optimization algorithm featuring biological evolution process imitation, SFLA integrates the advantages of two groups of artificial intelligence optimization algorithms, namely, the memetic algorithm (MA) based on memetic evolution and the particle swarm optimization algorithm

(PSO) based on group behaviors. SFLA is characterized by simple concepts, little parameter adjustment, high calculation speed, strong global search optimization capability and easy execution.

Shuffled Flog Leaping Algorithm (SFLA) is developed on the idea that a group of frogs live in a wetland, where many stones are scattered around. The frogs search for stones and leap onto places where there exist more food and each frog exchanges information through mutual cultural communication. Each flog has its own culture, which is defined as one solution to the problem. The whole population of frogs in the wetland is divided into different sub-groups, which has its own culture and executes local search strategies. Accordingly, each frog in the sub-group has its own culture, which influences other frogs. Meanwhile, each frog is subject to the impacts of others and evolves together with the sub-group. Once the sub-group evolves at a certain point, idea exchanges will take place across different sub-groups, namely global information exchange, and further realize the mixed algorithm among different sub-groups until the set conditions are met.

Algorithm parameters

Similar to other optimized algorithms, SFLA needs some necessary algorithm parameters, including frogs number F, number of populations m, number of frogs in the populations n, maximum allowed step S_{max}, global best solution P_b, local best solution P_b, local worst solution P_w, the number of frogs in sub-groups of q, times of local mimetic evolution LS.

Updating strategy

For frog populations, the solution with global best fitness is expressed as P_x, while the solution with the best fitness and the one with the worst fitness for each sub-group are expressed as P_b and P_w respectively. First, a local search for each sub-group will be conducted, to be more exact, a set of updated operation on individual frogs with the worse fineness in the sub-group.

The updating strategy in distance of frogs is as equation 1:

$$D_s = rand() * (P_b - P_w) \tag{1}$$

The value of each frog after updates is calculated by equation 2:

$$newD_w = oldP_w + D_s(-D_{max} \leq D_s \leq D_{max}) \tag{2}$$

Where, D_s represents the adjustment vector of individual frog, D_{max} represents the maximum allowed step in individual frog [27].

For example, Set P_w is [3 5 4 2 1], P_b is [1 5 3 4 2], the maximum allowed step D_{max} is 2, and if the rand is 0.5, $newD_w$ (1) will be $3+\max\{int[0.5\times(1-3)], -2\} = 2$; $newD_w(2) = 5 + \min\{int[0.5 \times (5-5)], 2\} = 5$. A new solution, $newD_w$= [2 5 4 3 1], can be obtained once the same operation is completed following the updated strategy.

B. SFLA for hardware/software partitioning

In this section, we present the SFLA algorithm in detail. We use the SFLA algorithm to solve the HW/SW partition problem in reconfigurable system on chip. The SFLA algorithm can get an approximate optimal result when satisfying the area constraint. The SFLA algorithm for HW/SW partitioning is presented in Algorithm 1.

Before presenting the details of SFLA algorithm, we define the fitness function denoted by $fitness$ in Equation 3.

$$fitness = \sum_{i=0}^{n} \begin{cases} TCosth_i, \text{if task on hardware} \\ TCosts_i, \text{if task on software} \end{cases} \tag{3}$$

where $TCosth_i$ represents the time cost, when task V_i is executed on hardware. $TCosts_i$ represents the time cost, when task V_i is executed on software. The fitness function $fitness$ represents the total completing time cost of all tasks in the critical path. We schedule the tasks with list scheduling algorithm, which is not our key focus in this paper. More details about list scheduling can be found, in [28].

The area cost constrain denoted by C_a in Equation 4.

$$\sum_{i}^{n} ACost_i \leq C_a \tag{4}$$

where $ACost_i$ represents the area cost of task V_i. The Equation 4 is mean that the total area cost of all tasks is not more than the value of C_a.

algorithm 1 Shuffled Frogs Leaping Algorithm for hardware/software partitioning (SFLA)

Require: A TDFG graph G.
The area constraint C_a
The max iteration S_{max}.
The number of frog populations F.
The number of populations m.
The number of frogs in the populations n.
Ensure: The optimal or approximate optimal partitioning.
1: Randomly initialize each node of graph G for each frog.
2: **for** $i \leftarrow 1$ to S_{max} **do**
3: Compute the fitness $fitness$ of each frog in the swarm by the fitness function in equation 3.
4: Sort all the frogs by descending order of $fitness$.
5: Find the global optimal P_x.
6: Divide the frogs into m groups, and each group contains n frogs.
7: Find the local optimal P_b and the local worst P_w in each groups.
8: Update the each node in each frog by equation 1 and equation 2.
9: **end for**
10: **return** P_b.

The following, sheds more light on the details of the SFLA algorithm for HW/SW partitioning problem. In

steps 1, we randomly initialize each node of graph G for each frog, X is randomly set as 0 or 1. From steps 2 to 8, are the procedures of optimization using SFLA algorithm for HW/SW partitioning problem. In steps 3, we compute the fitness function $fitness$ of each frog in the swarm by the fitness function in equation 3. Then we sort all the frogs by descending order of $fitness$ and find the global optimal P_x in step 4 and 5. In step 6, we divide the frogs into m groups, and each group contains n frogs. In step 7, we find the local optimal fitness P_b and the local worst fitness P_w. Step 8 is the most important phase of the SFLA algorithm. We will update the each node in the frog with the worst fitness by equation 1 and equation 2. When times of iteration is equal to S_{max}, the SFLA algorithm is terminated.

In this paper, the HW/SW partitioning problem has the area cost constrain. When we randomly initialize and update the node of each frog, we may get the unreasonable solution, which means that the total area cost of all frogs exceed the limit. During the initialization, we sum the each area cost $ACost_i$ from 1 to n when X is 1. If the value of total area cost is more than C_a, the procedure of initialization is stopped. We can do the same during updating process.

We will give more details about step 8 as in procedure 1.

procedure 1 Local Search about Step 8

1: **for** $i \leftarrow 1$ to m **do**
2: **for** $j \leftarrow 1$ to LS **do**
3: Update the frog with the worst fitness P_w by equation 1 and equation 2.
4: **if** The fitness of $newD_w$ is better than P_w. **then**
5: Use the new solution $newD_w$ instead of the old solution P_w.
6: **else**
7: Use the P_x instead of P_b in equation 1
8: Update as step 3.
9: **if** The fitness of $newD_w$ is worse than P_w. **then**
10: Randomly initialize the frog with the worst P_w.
11: **end if**
12: **end if**
13: Sort all the frogs in the group by descending order of $fitness$.
14: Find the local optimal P_b and the local worst P_w in the group.
15: **end for**
16: **end for**

According to equation 1, the value of D_s could be between 0 and 1. When we update each frog with equation 2, the values of $newD_w$ can take any value between -1 to 1. However, the HW/SW partitioning problem is a binary problem in this paper, so the nodes of each frog value must be chosen from either 1 or 0. Therefore we take the following method to design the values of node.

- If the value of $newD_w$ is more than 0.5, the value of node is 1, which means that the task will be implemented by hardware.
- While the value of $newD_w$ is between -0.5 to 0.5, the value of node is as it was.
- The value of node is 0, which means that the task will be implemented by software if the value of $newD_w$ is lower than -0.5.

V. Experimental

In this section, we will present the experimental results of evaluating the SFLA algorithm when applied to solving the HW/SW partitioning problem. The benchmarks are DFGs randomly generated by Task Graphs For Free (TGFF) [29], which is widely used in embedded system and HW/SW co-design for its flexibility and standardized performance. TGFF can generate the pseudo random task graphs for scheduling and allocation research. In this paper, all experiments are executed on a simulator, which is built with C++ program language, using an Intel(R) Core(TM) Duo Processor i3-3327U 1.90G processor.

We set the experiment parameters as follows:

- A is the area cost when all tasks are implemented by hardware.
- The number of frogs F is 100.
- The number of groups m is 10.
- The number of frogs in the populations n is 10.
- The times of local mimetic evolution LS is 10.
- The max iteration S_{max} is 10000.
- For our experiments, we choose three different area constrains, which is respectively $C_a = A/3$, $C_a = A/2$, $C_a = 3A/4$.

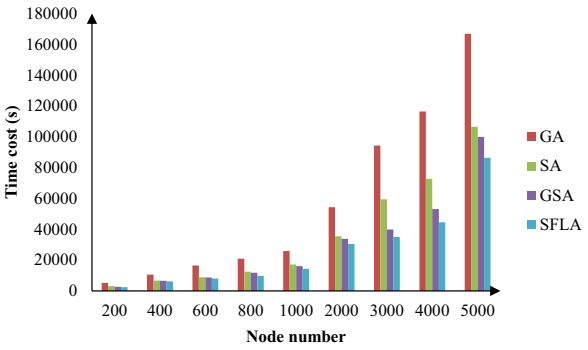

Figure 3. The task completing time results in the experiment, $C_a = A/2$

We compare the completing time of tasks in the critical path comparison among the Greedy Algorithm derived from Grode's Algorithm [30], Simulated Annealing algorithm (SA) derived from Eles's Algorithm [31], combined algorithm with Greedy and Simulated Annealing algorithm (GSA) derived from Jing's Algorithm [32], and SFLA algorithm. The value of time cost are shown in Figure 3, Figure 4, and Figure 5, respectively.

TABLE I.
COMPARE WITH THREE ALGORITHMS, $C_a = A/3$

Bench mark	Node number	Greedy	SA	GSA	SFLA	SFLA vs Greedy	SFLA vs SA	SFLA vs GSA
Random1	200	5790	3297	2663	2530	56.30%	23.26%	4.99%
Random2	400	11397	9436	8756	7325	35.73%	22.37%	16.34%
Random3	600	17420	15654	12034	9976	42.73%	36.27%	17.10%
Random4	800	21267	18935	16599	14885	30.01%	21.39%	10.33%
Random5	1000	27889	21226	19177	17341	37.82%	18.30%	9.57%
Random6	2000	56740	33018	32022	31067	45.25%	5.91%	2.98%
Random7	3000	97622	59887	39821	36015	63.11%	39.86%	9.56%
Random8	4000	120071	67508	53755	51215	57.35%	24.13%	4.73%
Random9	5000	171547	103490	91579	88123	48.63%	14.85%	3.77%
Average impov.						46.33%	22.93%	8.82%

TABLE II.
COMPARE WITH THREE ALGORITHMS, $C_a = A/2$

Bench mark	Node number	Greedy	SA	GSA	SFLA	SFLA vs Greedy	SFLA vs SA	SFLA vs GSA
Random1	200	5254	3083	2727	2430	53.75%	21.18%	10.89%
Random2	400	10596	6736	6586	6134	42.11%	8.94%	6.86%
Random3	600	16573	8902	8706	8054	51.40%	9.53%	7.49%
Random4	800	20991	12486	11837	9873	52.97%	20.93%	16.59%
Random5	1000	25987	17164	16136	14352	44.77%	16.38%	11.06%
Random6	2000	54406	35401	33831	30543	43.86%	13.72%	9.72%
Random7	3000	94340	59426	39880	35015	62.88%	41.08%	12.20%
Random8	4000	116444	72778	53151	44564	61.73%	38.77%	16.16%
Random9	5000	166993	106547	100000	86432	48.24%	18.88%	13.57%
Average impov.						51.30%	21.04%	11.61%

TABLE III.
COMPARE WITH THREE ALGORITHMS.$C_a = 3A/4$

Bench mark	Node number	Greedy	SA	GSA	SFLA	SFLA vs Greedy	SFLA vs SA	SFLA vs GSA
Random1	200	3740	3353	2087	1934	48.29%	42.32%	7.33%
Random2	400	7879	4822	4021	3605	54.25%	25.24%	10.35%
Random3	600	10460	8409	7837	6012	42.52%	28.51%	23.29%
Random4	800	12446	11014	10375	9432	24.22%	14.36%	9.09%
Random5	1000	14137	13310	11937	10842	23.31%	18.54%	9.17%
Random6	2000	34176	30566	26595	23517	31.19%	23.06%	11.57%
Random7	3000	63858	47231	39685	34015	46.73%	27.98%	14.29%
Random8	4000	80085	70878	50350	49215	38.55%	30.56%	2.25%
Random9	5000	113399	97340	66589	60123	46.98%	38.23%	9.71%
Average impov.						39.56%	27.65%	10.78%

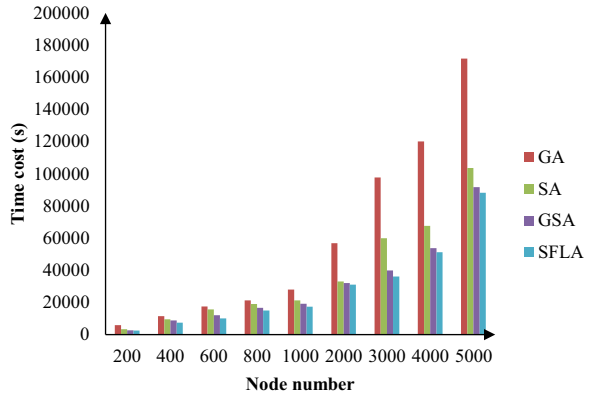

Figure 4. The task completing time results in the experiment, $C_a = A/3$

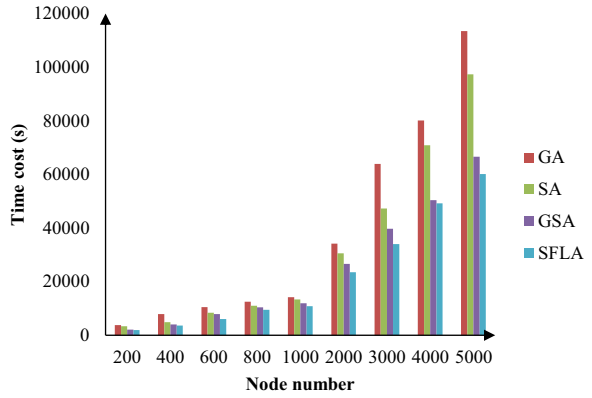

Figure 5. The task completing time results in the experiment, $C_a = 3A/4$

As shown in Table V, compared with SA algorithm, with the best in three different area constrain, our algorithm can reduce time cost about 41.09%%, as verse it also can reduce about 9.74%.

As shown in Table VI, compared with GSA algorithm, with the best in three different area constrain, our algorithm can reduce time cost by 18.99%, as verse it also can reduce about 3.37%.

Based on the three experimental results, while the vulue of area constrain C_a is equal to $A/2$, the performance of SFLA algorithm is the best among three area constrains. The cost of completing time can be reduced by 51.30%, 21.04% and 11.61% on average.

TABLE IV.
COMPARED WITH GREEDY ALGORITHM

Area constrain	Average impov.	Max impov.	Min impov.
C_a=A/3	46.33%	63.11%	30.01%
C_a=A/2	50.73%	62.88%	42.11%
C_a=A3/4	39.56%	54.25%	23.31%
Average	45.54%	60.08%	31.81%

TABLE V.
COMPARED WITH SA

Area constrain	Average impov.	Max impov.	Min impov.
C_a=A/3	20.13%	41.08%	8.94%
C_a=A/2	27.65%	42.32%	14.36%
C_a=A3/4	22.93%	39.86%	5.91%
Average	23.57%	41.09%	9.74%

TABLE VI.
COMPARED WITH GSA

Area constrain	Average impov.	Max impov.	Min impov.
C_a=A/3	8.82%	17.10%	2.98%
C_a=A/2	10.36%	16.59%	4.87%
C_a=A3/4	10.78%	23.29%	2.25%
Average	9.99%	18.99%	3.37%

VI. CONCLUSIONS

In this paper, we firstly apply SFLA algorithm to solving HW/SW partitioning problem on the reconfigurable system on chip. The potential of SFLA algorithm is substantially exploited so that we can get a low completing time cost of the critical path. Compared with greedy algorithm, the experimental results show that the SFLA algorithm can reduce the time cost by 45.54% on average with three different area constrain. The time cost of SFLA algorithm are also reduced by 23.57% and 9.99% on average with simulated annealing algorithm and combined algorithm with greedy and simulated annealing algorithm. And when area constrain C_a is $A/2$, SFLA algorithm can reduce the time cost by 51.30%, 21.04% and 11.61% on average, compared with that of greedy algorithm, SA, and GSA.

In the future, we will do more research on hardware/software partitioning algorithm. To be more specific, we will engage ourselves in how to improve the SFLA algorithm, and compare it with more algorithm, such as PSO, GA, Chemical Reaction Optimization (CRO) and so on.

ACKNOWLEDGMENT

This work is partially supported By National Natural Science Foundation Of China (NSFC) No. 61173014, National Natural Science Foundation Of China No. 61173036, Hunan Provincial Education Department No.13C333, the Research Foundation of Education Bureau of Hunan Province, China No.11C0573 , and the Science and Technology Research Foundation of Hunan Province No.2014GK3043.

REFERENCES

[1] P. Adhipathi, "Model based approach to hardware/software partitioning of soc designs," Ph.D. dissertation, Citeseer, 2004.

[2] G. Sapienza, T. Secelanu, and I. Crnkovic, "Modelling for hardware and software partitioning based on multiple properties," in *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*. IEEE, 2013, pp. 189–194.

[3] R. Niemann and P. Marwedel, "Hardware/software partitioning using integer programming," in *Proceedings of the 1996 European conference on Design and Test*. IEEE Computer Society, 1996, p. 473.

[4] J. Resano, D. Mozos, E. Pérez, H. Mecha, and J. Septién, "A hardware/software partitioning and scheduling approach for embedded systems with low-power and high performance requirements," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*. Springer, 2003, pp. 580–589.

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

[6] M. Abdelhalim, A. Salama, and S.-D. Habib, "Hardware software partitioning using particle swarm optimization technique," in *System-on-Chip for Real-Time Applications, The 6th International Workshop on*. IEEE, 2006, pp. 189–194.

[7] B. Guo, D. Wang, Y. Shen, and Z. Liu, "Hardware–software partitioning of real-time operating systems using hopfield neural networks," *Neurocomputing*, vol. 69, no. 16, pp. 2379–2384, 2006.

[8] J. Kaizhong, L. Zhao, and G. Junzhong, "A 0-1 hardware/software partitioning algorithm over ip cores," in *Autonomic and Autonomous Systems, 2006. ICAS'06. 2006 International Conference on*. IEEE, 2006, pp. 25–25.

[9] Z. Á. Mann, A. Orbán, and P. Arató, "Finding optimal hardware/software partitions," *Formal Methods in System Design*, vol. 31, no. 3, pp. 241–263, 2007.

[10] A. Farmahini-Farahani, M. Kamal, S. M. Fakhraie, and S. Safari, "Hw/sw partitioning using discrete particle swarm," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*. ACM, 2007, pp. 359–364.

[11] V. Arunachalam, S. Sapra, N. Chaitanya, and J. Prakash Rainac, "Hardware/software partitioning algorithm for embedded systems with repeated functionalities," in *TENCON 2008-2008 IEEE Region 10 Conference*. IEEE, 2008, pp. 1–6.

[12] Y. Liu, Q. C. Li, J. X. Liu, and J. Ma, "Study on hardware software partitioning using immune algorithm and its convergence property," in *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, vol. 3. IEEE, 2009, pp. 4–8.

[13] Y. Liu and Q. C. Li, "Hardware software partitioning using immune algorithm based on pareto," in *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on*, vol. 2. IEEE, 2009, pp. 176–180.

[14] H. D. Pando, S. C. Asensi, R. S. Lima, J. F. Calderín, and A. R. Suárez, "An application of fuzzy logic for hardware/software partitioning in embedded systems," *Computación y Sistemas*, vol. 17, no. 1, pp. 25–39, 2013.

[15] I. Bahri, L. Idkhajine, E. Monmasson, and M. E. A. Benkhelifa, "Optimal hardware/software partitioning of a system on chip fpga-based sensorless ac drive current controller," *Mathematics and Computers in Simulation*, vol. 90, pp. 145–161, 2013.

[16] R. O. Hassan, M. Abdelhalim, and S.-D. Habib, "Reliable pre-scheduling delay estimation for hardware/software partitioning," in *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on*. IEEE, 2013, pp. 1246–1250.

[17] H. Han, W. Liu, J. Wu, and G. Jiang, "Efficient algorithm for hardware/software partitioning and scheduling on mpsoc." *Journal of Computers*, vol. 8, no. 1, 2013.

[18] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 118–134, 2013.

[19] E. Sha, L. Wang, Q. Zhuge, J. Zhang, and J. Liu, "Power efficiency for hardware/software partitioning with time and area constraints on mpsoc," *International Journal of Parallel Programming*, pp. 1–22, 2013.

[20] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced engineering informatics*, vol. 19, no. 1, pp. 43–53, 2005.

[21] M. Horng, "Multilevel image threshold selection based on the shuffled frog-leaping algorithm." *Journal of Chemical & Pharmaceutical Research*, vol. 5, no. 9, 2013.

[22] P. Roy, P. Roy, and A. Chakrabarti, "Modified shuffled frog leaping algorithm with genetic algorithm crossover for solving economic load dispatch problem with valve-point effect," *Applied Soft Computing*, vol. 13, no. 11, pp. 4244–4252, 2013.

[23] Y. Xiao, X. Chai, L. B. Hu, C. Yang, and T. Lin, "Modified shuffled frog leaping algorithm for simulation capability scheduling problem," in *AsiaSim 2013*. Springer, 2013, pp. 71–81.

[24] Y. Xu, L. Wang, M. Liu, and S.-y. Wang, "An effective shuffled frog-leaping algorithm for hybrid flow-shop scheduling with multiprocessor tasks," *The International Journal of Advanced Manufacturing Technology*, vol. 68, no. 5-8, pp. 1529–1537, 2013.

[25] J. Luo and M.-R. Chen, "Improved shuffled frog leaping algorithm and its multi-phase model for multi-depot vehicle routing problem," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2535–2545, 2014.

[26] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.

[27] E. Afzalan, M. Taghikhani, and M. Sedighizadeh, "Optimal placement and sizing of dg in radial distribution networks using sfla," *International Journal of Energy Engineering*, vol. 2, no. 3, pp. 73–77, 2012.

[28] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Computing*, vol. 19, no. 12, pp. 1321–1344, 1993.

[29] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, 1998, pp. 97–101.

[30] J. Grode, P. V. Knudsen, and J. Madsen, "Hardware resource allocation for hardware/software partitioning in the lycos system," in *Proceedings of the conference on Design, automation and test in Europe*. IEEE Computer Society, 1998, pp. 22–27.

[31] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design automation for embedded systems*, vol. 2, no. 1, pp. 5–32, 1997.

[32] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," in *Parallel Computational Fluid Dynamics*. Springer, 2014, pp. 532–540.

Jiayi Du received his B.S. and M. S. degrees in computer science and technology from Hunan University, China in June 2004 and 2010. He is currently working towards his Ph.D. in computer science and technology at Hunan University, China. His current research interest includes embedded computing and parallel computing. He has published research articles in international conferences and journals of parallel computing and swarm intelligence algorithms. He is also a student member of IEEE Computer Society.

Kong Xiangsheng is a Lecturer in the Department of Computer Engineering, Xinxiang University. He received B.E. Degree in 2003 from PLA Information Engineering University, China. He has teaching experience of 11 years. He has published one PHP book, one JAVA book, two MySQL books and seven international papers. His interests are in system analysis & design and software testing.

Xin Zuo received his B.S. and M. S. degrees in computer science and technology, and management from Hunan University, China in 2009 and 2013. He is an officer working Information Department in Hunan University. His current research interest includes information system and corporate governance.

Lingyan Zhang received his B.S. and M. S. degrees in computer science and technology of Qingdao Technological University, Qingdao, China in 2003 and 2007. She is an officer in College of Computer Science at Jinan University. Her research interests include computer architecture and operating system.

Aijia Ouyang is a Lecturer in the College of Computer at Hunan City University, China. He is currently a Ph.D. candidate in the College of Information Science and Engineering at Hunan University, Changsha, China. His research interests lie in parallel computing, cloud computing and artificial intelligence. He has published many research articles on international conferences and journals of parallel computing and swarm intelligence algorithms.