IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 30, 2019, accepted October 28, 2019, date of publication November 8, 2019,
date of current version November 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2952434

Resource-Aware Task Scheduling and Placement in Multi-FPGA System

ZICHANG SUN, HAITAO ZHANG, AND ZEHAN ZHANG
Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, China
Corresponding author: Haitao Zhang (zht@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61532012, in part by the NSFC-Guangdong Joint Fund under Grant U1501254, and in part by the 111 Project under Grant B18008.

ABSTRACT With the development of high performance computing, Field Programmable Gate Arrays (FPGAs) are widely used for task acceleration. Partial reconfigurable technology provides the possibility of multi-task concurrent computation on FPGAs. However, it is still difficult to achieve high performance multi-task acceleration in multi-FPGA systems due to the resource contention in limited hardware resources and reconfiguration overhead. In this paper, we propose a two-stage task scheduling approach in multi-FPGA systems to optimize task execution efficiency. At the first scheduling stage, we select an appropriate computing unit for each task considering the subtask similarity and resource requirement similarity to reduce the possibility of reconfiguration and resource contention. At the second scheduling stage, we coordinate the subtask scheduling and placement to make full use of the hardware resources of FPGAs and improve the acceleration performance. Experiments show that our two-stage scheduling approach significantly reduces the tasks makespan and improves the utilization of resources compared with other traditional methods.

INDEX TERMS Multi-FPGA, task scheduling, task placement, partial reconfiguration.

I. INTRODUCTION

In recent years, with the increasing demand for computing power in data centers, FPGA, Graphics Processing Units (GPUs) and Application Specific Integrated Circuits (ASICs) have been widely used as accelerating devices to improve the execution efficiency of applications. Among them, FPGA-based accelerators are gradually becoming the core acceleration devices for computation-intensive tasks due to their high performance, low power consumption and reconfigurability [1], [2]. Several cloud service providers, such as Amazon, have deployed FPGA-based accelerators in their commercial environments to support the high performance computing of computation-intensive applications [3]. In the heterogeneous system with multiple FPGA-based acceleration devices, the CPU is mainly used as a controller for task management, and the complex computing tasks can be executed on the FPGAs. However, how to perform the efficient scheduling of computing tasks in such a multi-FPGA system brings new challenges.

The associate editor coordinating the review of this manuscript and approving it for publication was Qiang Ni.

Efficient scheduling of tasks in multi-FPGA system is different from that in other multi-processor or multi-core systems. We need to consider the following important factors. Firstly, the FPGA supports a technique called Partial Dynamic Reconfiguration (PDR) [4], which allows the FPGA to reconfigure some logic blocks at runtime without affecting the rest of the logic blocks [5]. This technology can dynamically replace the internal logic functions of the FPGA, which greatly increases system flexibility and resource utilization. At the same time, however, we need to explicitly consider the reconfiguration overhead caused by PDR because it can easily affect the performance gains from hardware acceleration. Secondly, different tasks require different hardware resources. In multi-FPGA system, in order to improve the resource utilization of the system as much as possible, it is necessary to reasonably allocate the available hardware resources and balance the utilization rate of fine-grained hardware resources in the FPGA. Finally, the tasks assigned to the FPGA must be placed in accordance with specific spatial allocation principles, and different task execution orders produce different size fragments of the reconfiguration region. Effectively reducing resource fragmentation is also

an important factor to be considered when designing task placement. Overall, the efficient online task scheduling and placement approach is needed to provide an appropriate solution for optimizing the task execution efficiency in multi-FPGA systems.

In this paper, we focus on the efficient task scheduling and placement in multi-FPGA system. The main contributions of this paper are as follows. First, we define a multi-task scheduling problem with fine-grained subtask division, scheduling and placement in the multi-FPGA system that supports PDR. Then, we propose a two-stage scheduling approach to minimize the makespan of multiple tasks by reducing reconfiguration overhead, resource contention and resource fragmentation. At the first scheduling stage, in order to reduce reconfiguration overhead and resource contention, we select an appropriate FPGA compute unit for each task based on the combination of the similarity of subtask types and the similarity of subtask multi-resource requirements. At the second scheduling stage, through the heuristic optimization method and effective three-dimensional spatial position planning, we give the reasonable scheduling order and the configuration location for each fine-grained subtask to reduce reconfigurable overhead and resource fragmentation. Finally, we analyze the impacts of two similarity weight factors on the scheduling results through experiments. The extensive experimental results show that our two-stage scheduling approach can achieve the higher hardware resource utilization and reduce the makespan of tasks compared with other popular methods.

The remainder of the paper is organized as follows. Section II describes the background of reconfigurable FPGA and reviews the related work. Section III defines the task scheduling and placement problem to be solved for the multi-FPGA system. Section IV presents the task scheduling and placement approach. Section V gives the experimental analysis. Section VI concludes this paper.

II. BACKGROUND AND RELATED WORK

In this section, we firstly briefly introduce the PDR technology of FPGAs and the challenges in the research process. Then, we intruduce the communication between reconfigurable modules. Finally, we introduce the work related to task scheduling in multi-FPGA system.

A. PARTIAL DYNAMIC RECONFIGURATION TECHNIQUE IN FPGA

The FPGA can be viewed as a resource pool with programmable functions. CLB (Configurable Logic Block) is the logical operation unit, BRAM (Block RAM) is the storage unit, and DSP (Digital Signal Processing) is the calculation unit. These units can be organized in different ways to achieve deep customization. PDR-supported FPGAs allow for dynamic reconfiguration of a portion of a device, while other components that are not affected by the reconfiguration process are still operational [6]. PDR can dynamically switch logic functions by changing part of the configured hardware module at runtime, which greatly improves the flexibility and resource utilization of the hardware system. The reconfiguration process is performed by means of a dedicated component, such as the Internal Configuration Access Port (ICAP) available on Xilinx devices, that is exploited to load the partial bitstream for the reconfigurable region to the FPGA configuration memory [7].

In an FPGA-based reconfigurable system, the tasks can be synthesized and divided into sets of subtasks. Different tasks containing multiple independent sets of subtasks can execute concurrently on reconfigurable resources. Before these tasks are executed, they need to be placed on the idle resources of the FPGA. After these tasks are executed, the reconfigurable resources occupied by these tasks need to be reconstructed, so that the next task set can continue to execute. The divided subtasks are usually abstracted into the small logical functional modules, and the entire reconfigurable resources are abstracted into one large logical functional region. The logical functional modules can be placed in parallel to execute on large logical functional region. Generally, the rectangular region is used to model the reconfigurable hardware resources on the FPGA, so the problem of placing tasks is transformed into the problem of finding an appropriate placement position for each subtask on the given two-dimensional reconfigurable rectangular region. At the same time, we must also consider the extra time overhead incurred during the reconfiguration process. When using FPGA for task acceleration, reconfiguration time is proportional to the size of the reconfigurable region, and the size of the reconfigurable region directly affects the amount of resources in this region. Therefore, effectively allocating idle resources and reducing reconstruction time are the focus in our work.

B. COMMUNICATION BETWEEN RECONFIGURATION MODULES

PDR technology of FPGA is to allow reconfigurable regions to be reconfigured without affecting the work of the rest of the configuration process. In PDR technology, the system is usually divided into fixed module and reconfigurable module. The communication between reconfigurable module and other modules (including reconfigurable module and fixed module, reconfigurable module and reconfigurable module) is realized by bus macro. Each module realizes in its own area. If two modules A and B want to communicate, they need bus macro, which straddles the boundary of the two modules, so that each module can be connected in its own area. So each module can be implemented in the specified area and communicate with the adjacent modules. In different series of FPGA devices, there are different ways to implement bus macro structure, such as Three-State Buffers (TBUF) Based and Slice based. In addition, Xilinx Company provided ready-made bus macro files for the developers of dynamic reconfigurable system and we use it in our system.

C. RELATED WORK

In multi-FPGA systems, task scheduling needs to be carefully designed by considering several complex factors comprehensively. Many previous researches focus on the task planning on a single FPGA. A small amount of other literatures specialize in task scheduling in multi-FPGA systems. In addition, there are also many researchers working on task scheduling in multi-processor or multi-core systems. Next, we review the related work from the above three aspects.

In order to minimize the resource fragmentation in FPGA, the authors in [8] proposed a heuristic task placement method based on 3D-Adjacency and Look-Ahead algorithm, which placed the task next to the boundary of the free area as much as possible. However, it does not take into account the simultaneous arrival of multiple tasks. Roman *et al.* [9] divided the reconfigurable resources into equal-width slots, which reduced the scale of the problem. However, this method ignores the size difference of the tasks, and the resource utilization ratio is quite low. Aiming at reducing chip area fragmentation, two resource allocation algorithms are developed in [10]. But these algorithms do not fully consider the resource type difference and order of the tasks. The approaches presented in [1], [11] focus on optimizing the utilization of available reconfigurable resources for task scheduling issues in parallel reconfigurable architectures. Reference [12] proposed a novel MER-based heuristic method based on 3D-Adjacency heuristic algorithm, which reduced regional fragmentation, improved utilization rate, and solved the problem that more than one tasks may arrive at the same time. These algorithms are basically aimed at task placement or resource allocation issues on a single FPGA device without considering the task scheduling in multi-FPGA system.

In addition to studying the task placement on a single FPGA, some researchers have also studied the task scheduling in multi-FPGA systems. In [13], [14], the authors considered the mapping of tasks on a multi-FPGA system and the scheduling at execution time, which provided the references for our work. In [14], the authors further proposed a Mixed-Integer Linear Programming (MILP)-based algorithm. Even though this approach could potentially be used to search for the optimal solution to the problem, the execution time of the algorithm increases exponentially with the number of tasks. Therefore, it is impractical for large problem instances. In [6], the authors proposed a fast deterministic scheduling heuristic algorithm to reduce the overhead incurred by partial dynamic reconfiguration. However, the author did not fully consider module reuse, so the reconfiguration overhead of tasks that can be executed by the same hardware is still the bottleneck of the algorithm. In [15], the authors proposed an energy-efficient scheduling algorithm based on ant colony optimization to solve the scheduling problem. And based on it, an improved algorithm is proposed to handle tasks with precedence and interdependencies. However, these two algorithms are designed to reduce the overall energy consumption, resulting in lower task parallelism. The authors of [16] proposed a benefit-based scheduling metric to evaluate the task assignment. However, they did not fully consider the resource contention between different tasks.

Task scheduling on multi-processor or multi-core systems has received widespread attention. The authors of [17], [18] tried to minimize job completion time by balancing the task queues of different types of heterogeneous resources. But in their models, each task can be executed only on its matching type of processors. The method proposed in [19] is based on constrained programming to allocate and schedule tasks in heterogeneous multi-core systems. Some previous works proposed heuristic scheduling methods centered on the critical path of the task graph to reduce the total execution time [20]–[22]. They proposed some dynamic scheduling approaches that reduce the total execution time by detecting the longest path or critical path of the dynamic task dependency graph. The authors of [23], [24] proposed the biology-inspired algorithms to solve Steiner tree problem in networks, which also provide an efficient solution for the resource composition optimization in parallel computing systems. Authors of the [25] designed a transformation strategy that reduces the problem of scheduling multi-type resources to single resource-type scheduling, thereby significantly reducing the algorithms running times without compromising the approximation ratios. Since these algorithms mainly consider the characteristics of general heterogeneous computing systems and do not take advantage of the unique advantages of FPGAs, they are not completely applicable to multi-FPGA systems. In addition, some other effective task programming algorithms have been proposed in literatures [26], [27], which stimulate the design principle of our task allocation approach.

In summary, task scheduling in various computing systems has been studied extensively, but there are still many problems needed to be solved for efficient scheduling in reconfigurable multi-FPGA systems. Therefore, this paper tries to deal with above unresolved issues, and the principle is to make full use of the FPGA computing power while reducing the overhead in multi-FPGA system.

III. PROBLEM DESCRIPTION

In this section, we first introduce the model of our multi-FPGA computing system and then give a detailed description of task scheduling and placement problem.

A. SYSTEM MODEL

Our multi-FPGA system contains several FPGA computing units which are responsible for accelerating the processing of computing-intensive tasks, and each of them has a reconfigurable region that contains various hardware resources such as CLBs, BRAMs, and DSPs. In the reconfigurable region of FPGA, resources are arranged in columns, and BRAMs and DSPs usually are less than CLBs as Figure 1.

The reconfigurable region can be partially reconfigurable at runtime to provide parallel acceleration for different tasks. The system accepts tasks submitted by users and then

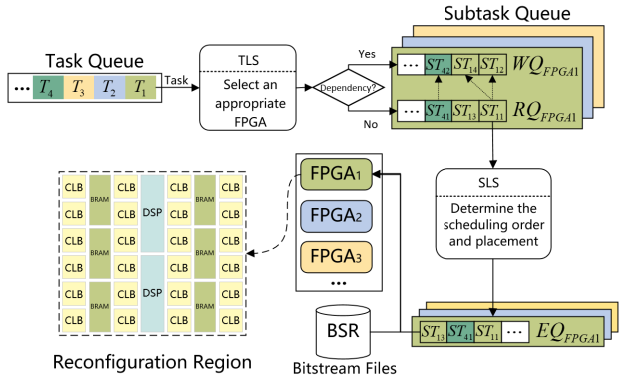

FIGURE 1. Scheduling model in multi-FPGA system.

performs the task scheduling and parallel computing process. Note that the tasks submitted by users can be divided into several subtasks in our system. In addition, our system also periodically collects the configuration state of reconfiguration region and then makes reasonable scheduling decisions based on the obtained information. There are two important schedulers in our system: the Task-Level Scheduler (TLS) and the Subtask-Level Scheduler (SLS). Moreover, the system has a subtask Bitstream Repertory (BSR) which stores the bitstream files for each subtask. And these files have been compiled by HLS Tool in advance [28]–[30]. The scheduling model in our system is shown in Figure 1.

As shown in Figure 1, the tasks submitted by users are stored in the task queue, and TLS analyses the subtasks structure of each task and the information of each FPGA to select an appropriate FPGA for the task. Then the system assigns these subtasks to the corresponding Waiting Queue (WQ_{FPGA}) and Ready Queue (RQ_{FPGA}) according to dependencies among the subtasks which have been predefined for each type of task. Note that we implement these two corresponding queues for each FPGA in our system, which are used to store the dependent subtasks and the independent subtasks, respectively. Then the SLS determines the appropriate scheduling orders and placement locations for those subtasks in each RQ_{FPGA}, and sends these subtasks into the Executing Queue (EQ_{FPGA}) according to the scheduling orders. Once the parent subtask ST_{ij} in the RQ_{FPGA} is completed, the child subtask ST_{ik} in the WQ_{FPGA} can be put into the RQ_{FPGA}. Finally, the system sends the bitstream files loaded from the BSR and the subtasks in each EQ_{FPGA} to the FPGAs. And then the subtasks are configured and executed on the FPGAs.

We assume the system has a set of m FPGAs denoted by $F = \{F_1,\ldots, F_m\}$. Each FPGA $F_i \in F$ has a reconfiguration region that can be regarded as a 2D rectangle with width of F_w and height of F_h. It can be divided into a number of contiguous free logic areas which can be allocated to different tasks and reconfigured for execution of tasks. We define the configuration state of each FPGA at time t as $F_i(t) = \{TC_{i1}(t),\ldots, TC_{i,n_i}(t)\}$, where n_i is the number of subtasks that has been configured on F_i, and $TC_{ij}(t)$, $j \in \{1,\ldots, n_i\}$ represents the *j-th* subtask currently configured to the *i-th* FPGA. In addition, $TC_{ij}(t)$ can be characterized as a vector $TC_{ij}(t) = (TC_{ij}^{id}(t), TC_{ij}^{clb}(t), TC_{ij}^{bram}(t), TC_{ij}^{dsp}(t), TC_{ij}^{x}(t), TC_{ij}^{y}(t), TC_{ij}^{w}(t), TC_{ij}^{h}(t), TC_{ij}^{trem}(t))$. Here $TC_{ij}^{id}(\cdot)$ represents the type of the subtask TC_{ij}, and $TC_{ij}^{clb}(\cdot)$, $TC_{ij}^{bram}(\cdot)$ and $TC_{ij}^{dsp}(\cdot)$ respectively represent the number of CLBs, BRAMs and DSPs needed. $TC_{ij}^{x}(\cdot)$ and $TC_{ij}^{y}(\cdot)$ are the horizontal and vertical coordinates of the top-left corner of the occupied logical area. $TC_{ij}^{w}(\cdot)$ and $TC_{ij}^{h}(\cdot)$ represent the width and height of the occupied logical resource area, respectively. In addition, $TC_{ij}^{trem}(\cdot)$ is the remaining execution time of the subtask $TC_{ij}(\cdot)$. We record the moment at which the subtask starts to be configured and calculate the remaining execution time at the current moment t according to the execution time and configuration time of the task.

B. WORKLOAD MODEL

We define each task as $T_i = (ID_i, G_i)$, where ID_i is the number of task type and G_i represents the workflow graph of the task. In addition, we define $G_i = (V_i, E_i)$, where V_i and E_i respectively represent the vertex set and the edge set of T_i. The vertex set V_i is defined as $V_i = \{ST_{i1},\ldots, ST_{i,m_i}\}$ where m_i is the number of the subtasks in T_i, and each directional edge $E_i^{kj} \in E_i$ represents the dependency from the parent subtask ST_{ij} to the child subtask ST_{ik}. Each subtask $ST_{ij} \in V_i$ can be characterized by $ST_{ij} = (ST_{ij}^{id}, ST_{ij}^{clb}, ST_{ij}^{bram}, ST_{ij}^{dsp}, ST_{ij}^{w}, ST_{ij}^{h}, ST_{ij}^{texe}, ST_{ij}^{bits})$ where ST_{ij}^{id} represents the number of subtask type, and ST_{ij}^{clb}, ST_{ij}^{bram} and ST_{ij}^{dsp} respectively represent the number of CLBs, BRAMs and DSPs needed. ST_{ij}^{w} and ST_{ij}^{h} represent the width and height of the rectangular logical resources area needed to be configured for the subtask ST_{ij}. ST_{ij}^{texe} and ST_{ij}^{bits} respectively represent the execution time of the subtask and the size of the corresponding bitstream. These characteristics of a subtask except ST_{ij}^{id} can be obtained by analyzing the reports compiled by Vivado High-level synthesis (HLS). We define the subtask configuration time ST_{ij}^{tcon} by

$$ST_{ij}^{tcon} = \frac{ST_{ij}^{bits}}{Rec_{freq}}, \tag{1}$$

where Rec_{freq} represents the time required to configure the bitstream files of unit size on an FPGA. In our system, multiple tasks can be submitted simultaneously and each task is defined as the above form.In addition, the same subtasks are also allowed in these tasks submitted.

Each subtask to be executed on an FPGA must first be configured with the corresponding logical resources, and this process can lead to a certain amount of reconfiguration overhead. However, if it is executed on the reconfiguration region that has already been configured for the same type of subtask, there is no reconfiguration overhead [13]. In addition, for different types of subtasks, if they have similar resource requirements, it is best not to execute them on an FPGA at the same time due to the resource contention which can cause the unbalanced utilization of multidimensional resources. For example, the remaining resource numbers of CLBs, DSPs and BRAMs on an FPGA are 100, 200 and 300 respectively, and

there are two independent subtasks $ST_{.1}$ and $ST_{.2}$. $ST_{.1}$ needs 50 CLBs, 20 DSPs, 20 BRAMs and $ST_{.2}$ needs 50 CLBs, 10 DSPs, 10 BRAMs, respectively. If we assign these two subtasks to the same FPGA, the CLB will be highly utilized, but DSP and BRAM will be largely idle. Therefore, in the multi-FPGA system, we need to first select an appropriate FPGA for each task T_i to reduce the reconfiguration overhead and improve resource utilization.

On the other hand, the inappropriate subtask scheduling sequence may result in the same type of subtasks having to be reconfigured and degrade the performance of system. And the inappropriate placement of subtasks can result in more resource fragments in the reconfiguration region of FPGA and reduce the utilization of resources. Therefore, we need to determine the appropriate scheduling order for the set of subtasks and select the appropriate placement location for each subtask to be configured on FPGA to reduce the reconfiguration overhead and resource fragmentation.

Objective. In this paper, we define a task set as $T = \{T_i | i = 1,.., N\}$ which is submitted by users. Our objective is to minimize the makespan of the task set T by designing the efficient task scheduling and placement approach in the multi-FPGA system considering the task reconfiguration overhead and hardware resource contraints of FPGAs.

IV. THE PROPOSED TASK SCHEDULING AND PLACEMENT APPROACH ON MULTI-FPGAS

A. DESIGN OVERVIEW

In order to minimize the makespan of task set T, we propose a two-stage task scheduling approach in the multi-FPGA system.

1) FIRST SCHEDULING STAGE

At this stage, we regard the whole task as a scheduling unit and choose an appropriate FPGA computing unit for it. We first acquire the subtasks for each FPGA computing unit, including those subtasks in RQ_{FPGA}, WQ_{FPGA} and those have been configured to the FPGA. Then we estimate the subtask similarity between the task and all subtasks assigned to each FPGA using Jaccard coefficients, which is defined as the ratio between the size of the intersection of two sets and the size of the union. The larger the Jaccard coefficient value is, the higher the sample similarity degree will be. The purpose is to allocate the same type of subtasks to the same FPGA as much as possible to reduce unnecessary reconfigurable operations. Then we calculate the resource requirements similarity between the task and the subtasks assigned to each FPGA which can measure the possibility of resource contention between them. The goal is to assign the subtasks with different resource requirements to the same FPGA as much as possible to improve the multidimensional resource utilization. Finally, we combine two similarities with two weighting factors to select an appropriate FPGA considering reconfiguration overhead and resource contention comprehensively. After selecting the FPGA, the subtasks included in the task are assigned to the WQ_{FPGA} and RQ_{FPGA} of the corresponding FPGA according to the dependencies among them.

2) SECOND SCHEDULING STAGE

At this stage, we focus on subtasks scheduling and placement in the RQ_{FPGA} of a single FPGA. The subtasks in the RQ_{FPGA} are independent, and they can execute on the FPGA completely in parallel. However, due to the resource constraints and reconfigurable consumption, the reasonable scheduling order and placement locations are important to minimize the makespan of subtasks. We find the optimal scheduling order based on a combinatorial optimization method combining Genetic Algorithm and Ant Colony Optimization Algorithm, which performs the continuous iteration process to find the appropriate combination in the solution space. The solution space is a set of permutations and combinations of all subtasks in RQ_{FPGA} and the object of each iteration process is to select a more suitable scheduling solution. In order to evaluate the fitness of each scheduling order in the iteration process, we calculate the makespan of subtasks according to each scheduling order found in the current iteration process. This makespan is also influenced by the placement of subtasks in the reconfigurable region of the FPGA. So in order to reduce makespan, we obtain the candidate locations in the reconfigurable region for each subtask, and find the location with the smallest resource fragmentation to place it by calculating 3D-contact value. Ultimately, after the iterative optimization process, we can determine the placement location for each subtask and the scheduling order of subtasks.

B. TASK-LEVEL SCHEDULING ON MULTI-FPGA

In this subsection, we focus on the task-level scheduling, and the objective is to select an appropriate FPGA computing unit for each task.

As the system and workload model described in Section III, every task T_i has a set of subtasks V_i and there is a set of m FPGA computing units in the system. For each FPGA F_k, we can obtain two subtask sets in the WQ_{FPGA_k} and RQ_{FPGA_k} at time t, defined as $S_{w_k(t)} = \{ST_{ki} | i = 1, \ldots, x\}$ and $S_{r_k(t)} = \{ST_{ki} | i = 1, \ldots, y\}$. In addition, the configured subtasks set $F_k(t)$ with p subtasks in the F_k can also be obtained as mentioned in Section III. We define S_k is a combination of $S_{r_k(t)}$, $S_{w_k(t)}$ and $F_k(t)$, that is $S_k = \{ST_{ki} | i = 1, \ldots, x+y+p\}$ which represents the set of all subtasks assigned to the FPGA F_k at time t. Every subtask ST_{ki} in the S_k has these features ST_{ki}^{id}, ST_{ki}^{clb}, ST_{ki}^{bram}, ST_{ki}^{dsp}, ST_{ki}^{w}, ST_{ki}^{h}, ST_{ki}^{texe} and ST_{ki}^{bits} defined in Section III.

In this stage, we need to find an appropriate FPGA for each task and formalize it as a task-to-FPGA mapping M. $M(i) = k$, $k \in \{1, \ldots, m\}$ represents that the task T_i is assigned to the FPGA F_k. In order to find the reasonable mapping $M(i)$, we define our objective function φ considering two aspects of issues: one is to reduce the task reconfiguration, and the other is to reduce the contention of hardware

requirements between subtasks. The objective function is defined as

$$\varphi = \lambda SK(M) + \mu \frac{1}{R(M)}, \tag{2}$$

where $SK(M)$ represents the subtask similarity between the task T_i and the subtasks set S_k, and $R(M)$ represents the resource requirement similarity between them. λ and μ respectively are the weights of $SK(M)$ and $R(M)$. $SK(M)$ is for subtasks with the same subtask type between the subtask set of tasks to be assigned and those in subtask sets assigned to each FPGA, while $R(M)$ is for subtasks with different subtask type in those two kind sets. There is a certain relative relationship between $SK(M)$ and $R(M)$. In order to consider two similarities and assign tasks better, we set weight factors for each similarity λ and μ and we set $\lambda + \mu = 1$. The larger the $SK(M)$, the more subtasks of the same type, which means that it is more likely to reduce the unnecessary reconfiguration operations. The smaller the $R(M)$ is, there is less possibility of resource contention between T_i and S_k, which is more conducive to improve the utilization of multidimensional resources. Our objective is to find a mapping $M(i)$ that maximizes the objective function φ.

To obtain the maximal $SK(M)$ to reduce the unnecessary task reconfiguration operations, we use Jaccard coefficients to estimate subtask similarity under the mapping $M(i)$. The subtask similarity $SK(M(i))$ between the subtasks set V_i and S_k is defined as

$$SK(M(i)) = \frac{Num(V_i \cap S_k)}{Num(V_i \cup S_k)}. \tag{3}$$

In addition, in order to obtain the minimal $R(M)$, we first show the resource requirement similarity between two subtasks. If two different type of subtasks are assigned to the same FPGA and their resource requirements are similar, there may be the resource contention between them. To measure the similarity of resource requirements between any two different type subtasks, we first extract the resource requirement vector as $R_{\cdot j} = (ST_{\cdot j}^{clb}, ST_{\cdot j}^{bram}, ST_{\cdot j}^{dsp})$ for the *j-th* subtask in a set. Then we use following cosine value of the resource requirement feature vectors to measure the similarity of resource requirements between any two subtasks.

$$cos(\theta)_{xy} = \frac{\sum_{i=1}^{n} R_{\cdot x} \cdot R_{\cdot y}}{\sqrt{\sum_{i=1}^{n} R_{\cdot x}^2} \cdot \sqrt{\sum_{i=1}^{n} R_{\cdot y}^2}}, \tag{4}$$

where $R_{\cdot x}$ and $R_{\cdot y}$ are the resource requirement vectors for two different types of subtasks $ST_{\cdot x}$ and $ST_{\cdot y}$. θ is the angle between the two vectors and $cos(\theta) \in [0, 1]$.

For obtaining the minimal $R(M)$ of the sets V_i and S_k, we need to first remove the subtasks of the same type between them and then obtain two subsets V_i' and S_k'. Any two subtasks ST_{ix} and ST_{ky} meet $ST_{ix} \in V_i'$, $ST_{ky} \in S_k'$ and $ST_{ix}^{id} \neq ST_{ky}^{id}$. And we define $R(M(i))$ as

$$R(M(i)) = \frac{\sum_{x=1}^{p} \sum_{y=1}^{q} cos(\theta)_{xy}}{p * q} \tag{5}$$

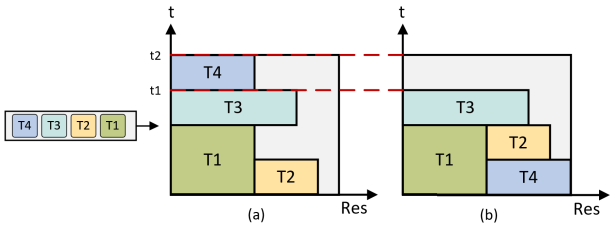

FIGURE 2. (a) Scheduling sequence A. (b) Scheduling sequence B.

where p represents the number of subtasks in V_i' and q represents the number of subtasks in S_k'.

The detailed task-level scheduling method is showed in Algorithm 1.

Algorithm 1 Task-Level Scheduling Method

Input: A subtask set V_i of task T_i to be assigned;
A set of subtasks that have been assigned to all FPGAs at the current moment $S_{tot} = \{S_1, \ldots, S_m\}$ in which each element $S_k, k \in \{1, 2, \ldots m\}$ denotes the set of subtasks assigned to *k-th* FPGA F_k;
Output: The result k of mapping $M(i)$ with the maximized φ for task T_i;

```
for each subtask set S_k in S_tot do
    Compute subtask similarity S(M(i)) of V_i and S_k using Eq3;
    Remove subtasks of the same subtask type ST_..^id between V_i and S_k to obtain two subsets V_i' and S_k';
    Calculate the resource requirement similarity R(M(i)) using Eq5;
    Calculate the objective φ using Eq2;
Return k that maximizes φ under mapping M(i) = k;
```

C. SUBTASK-LEVEL SCHEDULING AND PLACEMENT ON SINGLE FPGA

As section III-B describes, each subtask must be configured with the corresponding logical resources before it is executed which leads us to consider how to place subtasks reasonably under a limited reconfigurable region. In addition, for the subtasks in the queue RQ_{FPGA}, their scheduling order also affects the makespan of the entire set. As shown in Figure 2, for these four subtasks, the scheduling sequence A is T1→T2→T3→T4 and the scheduling sequence B is T1→T4→T2→T3. As we can see that different scheduling order results in different makespan due to the limited resources. To solve this problem, we propose the Subtask-Level Scheduling and Placement (SLSP) method combining Genetic Ant Colony Optimization (GACO) algorithm with MER-3D-Placement algorithm [12].

The basic idea of SLSP method. For the subtask set $S_{r_k}(t)$ with y independent subtasks in the RQ_{FPGA_k} at time t, we need to determine the appropriate scheduling order and placement location for each subtask $ST_{ki}, i \in \{1, \ldots, y\}$.

SLSP method is divided into two main parts. Firstly, we combine Genetic Algorithm (GA) [31], [32] with bi-directional convergent Ant Colony Optimization

(ACO) [33], [34] algorithm to make subtasks scheduling order decision which can overcome the shortcomings of GA and ACO algorithms. It can be better than GA in accuracy and ACO in time efficiency. Secondly, we use the MER-3D-Placement method to obtain the optimal placement location for each subtask. It takes into account the time of each subtask occupying resources, which makes it can reduce the resource fragmentation more than the traditional two-dimensional placement algorithm. In the whole subtasks scheduling and placement process, we first make full use of the randomness and fast global convergence of GA to generate a pheromone distribution of subtask scheduling order problem. This distribution is used as the initial pheromone distribution of the bidirectional convergence ACO. On this basis, we make full use of its positive feedback and high accuracy to find the optimal solution of scheduling order. In addition, during the whole GA and ACO process, we need to evaluate each scheduling order according to the fitness in makespan of subtasks and find a placement loation for each subtask in the reconfigurable region. Therefore, we use MER-3D-placement method to find the optimal location for each subtask according to each scheduling order and then calculate the makespan of the all subtasks to determine the fitness of the scheduling order.

Next we introduce the placement algorithm for a single subtask, and then introduce the overall subtask scheduling and placement method in detail.

1) MER-3D-PLACEMENT METHOD BASED ON MULTI-RESOURCES REQUIREMENTS

It is a common layout strategy to regard the reconfiguration region as a rectangle and consider the remaining resources as MERs. However, there is still much room for improvement in how to allocate MERs reasonably. However, most of the previous algorithms only consider the resource occupation from two-dimensional area and rarely consider the occupied time. Moreover, the previous algorithms usually only consider the CLB resources neglecting other hardware resources, such as BRAM and DSP, which affect the performance of task execution. In our MER-3D-Placement method, we not only consider the three-dimensional space, but also consider different hardware resources to calculate the 3D connection value, and then select the location that produces the smallest fragmentation for allocation. In this way, we can make full use of resources and give full play to the characteristics of different hardware resources (CLB,DSP,BRAM). Next is our detailed description of the algorithm.

In order to better describe our algorithm, we first give a new representation of MER in our method: $MER = (MER_x, MER_y, MER_w, MER_h, MER_r)$, where MER_x, MER_y, MER_w, MER_h respectively represents the horizontal and vertical position of the top-left corner, the width and height of MER. In addition, MER_r indicates whether the resources in this rectangle contain DSP and BRAM. As described in section III, the resources of DSP and BRAM are column-based in reconfiguration region and the columns are fixed.

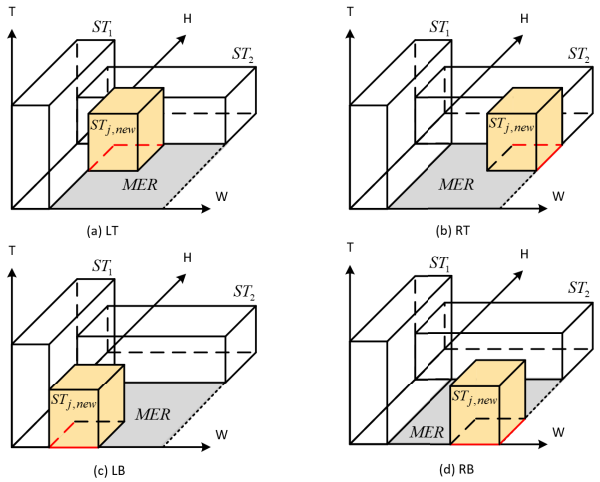

FIGURE 3. Candidate positions of MER-3D-Placement.

Section III shows that each FPGA has a reconfigurable region, which can be formalized as a 2D rectangle to configure different logical functions. At the time t, we can get the configured subtasks set $F_k(t)$ of the k-th FPGA. And any configured subtask has these attributes $TC_{ki} = (TC_{ki}^{id}, TC_{ki}^{clb}, TC_{ki}^{bram}, TC_{ki}^{dsp}, TC_{ki}^{x}, TC_{ki}^{y}, TC_{ki}^{w}, TC_{ki}^{h}, TC_{ki}^{trem})$ defined in Section III-A. When a subtask $ST_{j,new}$ of a task T_j is to be executed, we need to check whether the reconfiguration region has been configured for the same type of subtask with $ST_{j,new}^{id}$. If it has been configured, the subtask only needs to wait untill other subtasks with the same type are completed. Otherwise, we need to find the appropriate location for it.

To get the appropriate location $L = (x, y)$ for $ST_{j,new}$ on the k-th FPGA, we extract the location feature $LF_k = (TC_{ki}^{x}, TC_{ki}^{y}, TC_{ki}^{w}, TC_{ki}^{h}, TC_{ki}^{trem})$ for each configured subtask TC_{ki}, and then use it to compute all the maximum empty rectangles $MERs = \{MER_j | j = 1, ...z\}$ in the current state. Next we traverse all MERs to calculate the 3D-contact value of each candidate position in each MER. As Figure 3 shows, there is only one MER due to the resources occupy of the tasks ST_1 and ST_2. To configure $ST_{j,new}$, we traverse the candidate locations of the MER to calculate 3D-contact value which include top-left corner (TL), top-right corner (TR), bottom-left corner (BL), bottom-right (BR) corner shown in subfigure (a), (b), (c), (d) of figure 3, respectively. Because of the different adjacent edges, different candidate positions have different 3D-contact values. Futher, if $ST_{j,new}$ needs the BRAM or DSP, that is $ST_{j,new}^{bram} \neq 0$ or $ST_{j,new}^{dsp} \neq 0$, we will take into account the two intersections consisting of the column of the specific resource (BRAM or DSP) and the edges of MER. In order to make full use of various resources, we give the priority to two specific intersections in all MERs containing this resource. Finally we select the candidate location with maximized 3D-contact value for new subtask. The 3D-contact value C is defined as

$$C = \sum_{p=0}^{n} L_p \times LT_p \tag{6}$$

where n denotes the number of edges connected to any configured subtask and the edges of reconfiguration region after the new subtask is placed in a candidate location. L_p represents the length of the overlap of the *p-th* contacted edge. In addition, LT_p is a period of time for the *p-th* contacted edge called edge lifetime [8]. In other words, if the edge belongs to $ST_{j,new}$ and a configured subtask TC_{ki}, the LP_p is the shortest remaining time for the two subtasks to execute. In this case, LT_p is defined by

$$LT_p = min\{ST^{texe}_{j,new}, TC^{trem}_{ki} - ST^{tcon}_{j,new}\}, \tag{7}$$

where $ST^{texe}_{j,new}$ is the execution time and $ST^{tcon}_{j,new}$ is its configuration time defined in section III-B. TC^{trem}_{ki} is the remaining execution time of TC_{ki}. In addition, if the *p-th* contacted edge belongs to the reconfiguration region, then $LT_p = ST^{texe}_{j,new}$.

Our MER-3D-Placement method based on different hardware resource requirements is showed in Algorithm 2.

Algorithm 2 MER-3D-Placement Method Based on Multi-Resources

Input: A new subtask to be configured $ST_{j,new}$;
The configuration state $F_k(t)$ of a FPGA F_k at time t;
The column value of BRAM and DSP in FPGA Col_{bram}, Col_{dsp};
Output: Location $L(x, y)$ where $ST_{j,new}$ is configured;

```
Obtain the available MER set MERs according to F_k(t), Col_bram, Col_dsp of the FPGA F_k;
Filter subset MERs' that satisfies resource requirement of the ST_{j,new};
Initialize an array Cs to record 3D-contact values for each candidate location;
if No available MER in MERs' then
    Return null;
for each MER in MERs' do
    if ST_{j,new} needs BRAM or DSP then
        for each MER has the specific resource do
            Calculate the 3D-contact value C of all candidate intersections using Eq6;
    else
        for each MER in MERs' do
            Calculate the 3D-contact value C of the all candidate locations using Eq6;
Return the corresponding location L(x, y) with maximum C in Cs;
```

2) SUBTASK-LEVEL SCHEDULING AND PLACEMENT METHOD

For optimizing the makespan MP of the subtask set S_{kr} with y subtasks, we combine GACO algorithm with MER-3D-Placement algorithm.

In GACO, we can continuously get different scheduling orders. In order to evaluate the optimized order π, we first define the fitness of the order as the makespan of all subtasks in the scheduling order π, which is formalized as $Fit(\pi) = MP_\pi$. To obtain the fitness Fit, we need to combine the MER-3D-Placement algorithm to get the placement location of each subtask. For a scheduling order $\pi = \{\pi^1, \ldots, \pi^y\}$, $\pi^i \in \{1, \ldots, y\}$ and the detailed calculation of its fitness is shown in Algorithm 3.

Algorithm 3 Fitness of a Scheduling Order

Input: A scheduling order π;
The State Vector $F_k(t)$ of a FPGA F_k at the time t;
The subtask set S_{r_k} in the ready queue;
Output: The fitness $Fit(\pi)$;
Every placement location L_i for each subtask ST_{ki};

```
Initialize temp status of FPGA F_temp = F_k(t), Fit(π) = 0;
Sort S_{r_k} according to the scheduling order π;
for each subtask ST_ki of S_{r_k} do
    Get placement location L_i of ST_ki according to Algorithm 2;
    if L_i = null then
        Get the remianing time TC^trem_{temp,ear} of the subtask TC_{temp,ear} that will be earliest completed in F_temp;
        Fit(π) += TC^trem_{temp,ear};
        for each subtask TC_{temp,i} in F_temp do
            TC^trem_{temp,i} = TC^trem_{temp,i} - TC^trem_{temp,ear};
            if TC^trem_{temp,i} = 0 then
                Remove TC_{temp,i} in F_temp;
    else
        Create an element TC_{temp,con} and initialize it with the attributes of ST_ki and add it to F_temp;
        if ST_i is the last element then
            Get the remaining time TC^trem_{temp,last} of the last subtask TC_{temp,last} to be accomplished in F_temp;
            Fit(π) += TC^trem_{temp,last};
Return Fit(π) and every subtask's location L_i;
```

In GA stage, an initial population containing *popsize* chromosomes is firstly initialized, in which each chromosome represents a scheduling order π. Then, the fitness $Fit(\pi)$ of each chromosome in the population is calculated by Algorithm 3 and the chromosomes are crossed, mutated, and duplicated by a certain selection probability $P(\pi)$ to form new N chromosomes. Through continuous evolutionary iteration, we can find the appropriate optimization results. The selection probability $P(\pi_k)$ is defined as

$$P(\pi) = \frac{Fit(\pi)}{\sum_{k=1}^{popsize} Fit(\pi)} \tag{8}$$

When GA merges with ACO, in order to avoid that the

fixed number of GA iterations will affect the efficiency of the algorithm in the traditional algorithm, we use dynamic fusion to ensure the correct conversion time of GA and ACO. We set up the maximum and minimum genetic iterations as Gen_{min}, Gen_{max}, and give the minimum evolutionary rate Evo_{min}. When the evolutionary rate Evo_{die} is less than the Evo_{min}, we transform GA to ACO.

In the stage of ACO, according to the law of ants foraging, the optimal scheduling sequence is found through the swarm intelligence of M ants. Each ant schedules every subtask in $S_{r_k}(\cdot)$ of FPGA F_k and releases a certain pheromone γ_{ij} between the subtask ST_{ki} and the subtask ST_{kj}. In our algorithm, we first initialize pheromone $\gamma_{ij} = \gamma_{ij_G}$, where γ_{ij_G} denotes the proportion of the sequence of ST_{ki} to ST_{kj} in a set of optimal solutions of GA. After an ant A_m schedules all subtasks, in order to accelerate convergence of ACO, the pheromone concentration γ_{ij} is updated as follow:

$$
\begin{aligned}
.\gamma_{ij}(t+n) &= (1-\delta)\gamma_{ij}(t) + \Delta\gamma_{ij}(t+n), \\
\Delta\gamma_{ij}(t+n) &= \begin{cases} \dfrac{Q}{Fit_\pi}, & \text{if } \pi = \pi_{best}; \\ \dfrac{-Q'}{Fit_\pi}, & \text{if } \pi = \pi_{worst}; \\ 0, & others, \end{cases}
\end{aligned} \tag{9}
$$

where $\Delta\gamma_{ij}(t+n)$ is a reward-penalty function which rewards the best order π_{best} and punishes the worst order π_{worse} in n-period. Q is the pheromone constant for reward and Q' is the pheromone constant for punishment. δ is the pheromone volatilization coefficient. When more and more ants pass through the same order, more and more pheromones is found. After endless efforts, they choose the appropriate optimal scheduling order. The probability $\rho_{ij}(t)$ of an ant A_m choosing the next subtask at time t is defined as

$$
.\rho_{ij}^{k}(t) = \begin{cases} \dfrac{[\gamma_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{s\in S}[\gamma_{is}(t)]^\alpha [\eta_{is}(t)]^\beta}, & j \in Next_m; \\ 0, & otherwise, \end{cases} \tag{10}
$$

$$
\eta_{ij}(t) = \frac{1}{Fit_t(\pi_m)} \tag{11}
$$

where α is pheromone heuristic factor and it reflects the strength of random factors in ant colony search. β is expectation heuristic factor which reflects the strength of the priori and deterministic factors of ant colony. And $Next_m$ is the next subtask sets that ant A_m can choose. $Fit_t(\pi_m)$ is the fitness of the order chosen by ant A_m at the time t.

The details of SLSP algorithm are shown in Algorithm 4, which can get the optimized result for the subtask-level scheduling and placement.

Algorithm 4 SLSP Algorithm Based on GACO and MER-3D-Placement

Input: A subtask set S_{r_k} to be scheduled and configured on FPGA F_k;
The state vector $F_k(t)$ of F_k at time t;
Output: The scheduling order π and every subtask's location $L(x, y)$ in S_{r_k};

```
Initialize parameters Gen_min, Gen_max, Evo_die, Evo_min, popsize, M, α, β, Q, Q', δ;
Randomly generate an initial population of popsize chromosomes;
while Iteration number i in [Gen_min, Gen_max] and Evo_die > Evo_min do
    Calculate each Fit(π) of chromosome according to Algorithm 3;
    Calculate the probability ρ_π according to Eq8;
    while chromosomes are not fully traversed do
        Selection,crossover and mutation;
    Replace old population by a new population;
Initialize the pheromones between any two subtasks ST_ki ST_kj using γ_ij = γ_ijG;
Randomly place M ants on different M subtasks ST_k·;
while No optimal solution is found do
    for each ant A_m do
        for each schedule order π_m do
            Select the next subtask based on Eq10, Eq11;
        Evaluate fitness Fit(π_m) using Algorithm3;
        Update pheromone γ_ij using Eq9;
Return the best scheduling order π and the location L_j(x, y) for each subtask ST_kj.
```

V. EXPERIMENTAL RESULTS

A. EXPERIMENT SETUP

We have built our multi-FPGA computing system as described in Section III, including BSR, TLS, SLS and so on. The system is configured with a server with an Intel(R) Xeon(R) CPU E5-2620 v3 working at a frequency of 2.40GHZ and five Xilinx Kintex UltraScale KCU1500 FPGA boards. FPGA is connected with server through PCIE. In this way, we can easily change the number of FPGAs in the system. Each board has 4 DDR4 memory with total 16GB. Moreover, the FPGA has a specific chip, XCKU115-2FLVB2104E, which includes 1,326,720 FlipFlops (FFs), 663,360 Look-Up Tables (LUTs), 5,520 DSP Slices and 2159 Block RAMs with 32Kb each. In Xilinx's UltraScale series, a CLB can be grouped by 8 six-input LUTs and 16 FFs. Therefore, there are about 82920 CLBs in the chip. And in our experiments, the ratio of rows and columns of the reconfiguration region is about 3:2 and every FPGA has about 337 rows and 224 columns hardware resources (CLBs, BRAMs, DSPs) in its reconfiguration region. The reconfigurable region can be reconfigured at runtime while maintaining all other components functional. Once the task is configured, the system records the relevant information.

In addition, we use several real benchmarks to evaluate our task scheduling and placement algorithm. These benchmarks are a mix of different applications from Microelectronic Center of North Carolina [35] benchmark suite and Media-

TABLE 1. Tasks implemented information.

Task	Source
Diffeq	Microelectronic
Finite Impulse Response	Microelectronic
Advanced Recording Format	MediaBench
Discrete Cosine Transforms	MediaBench
JPEG Compand Decomp	MediaBench
G721 Encoder and Decoder	MediaBench
Wavelet	Implemented
Sobel Filter	Implemented
Deriche Filter	Implemented
Gaussian Pyramid	Implemented
Edge Detection	Implemented
Object Tracking	Implemented
GMM Background Modeling	Implemented

Bench [36]. In addition, we implement several different tasks using Vivado Design Suite provided by Xilinx. Table 1 lists these tasks used in our experiments and the workloads are different combinations of randomly generated tasks with different numbers. In our implementation process, each task is divided into 3 to 30 subtasks modules which require different resources (CLBs, BRAMs or DSPs) in advance according to the processing flow of each task. And the same subtask modules may exist in different tasks. For example, there are same modules in edge detection and target detection, such as binarization, corrosion filtering, median filtering and so on. It means that the subtasks of the same type may exist in different tasks. And we store the binary compiled files of these subtasks in BSR of our system.

For comparing the quality of our proposed two-stage scheduling algorithm with other task scheduling algorithm, we have implemented several multi-node scheduling algorithms and single-node scheduling algorithms in the system. Multi-node sheduling methods include Least loaded (LL) method and Round Robin (RR) method which are widely used in multi-node scheduling. And single-node placement methods include MER-2D adjacent placement (MER-2D) method, MER First Fit (MER-FF) method and MER-3D-Contact (MER-3D). We schedule these subtasks by the order of subtask arrival. MER-2D method considers the resource occupancy compactness of subtasks placement in two-dimensional resource plane. MER-FF method is to find the first suitable MER for the coming subtask according to its resource requirements and then chooses the location for it. MER-3D is simlilar to our placement method without scheduling strategy.

B. PARAMETER ANALYSIS

1) SIMILARITY WEIGHT FACTORS ANALYSIS

In task-level scheduling, two important similarity weight factors λ and μ affect the performance of task scheduling. In order to accurately measure the performance of our approach, we first determine them using the different combinations of different tasks in Table 1 with different input data. We design three kind of sets of task: highly similar task sets (HSS), moderately similar task sets (MSS) and low similar task sets (LSS) according to the repeatability of the same

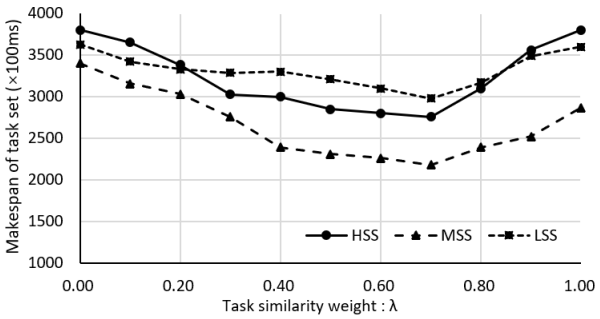

FIGURE 4. Makespan of the task sets with different task similarity weight.

hardware subtasks. Each set has 100 tasks with 300 subtasks. There are only 10 types of subtasks in a set of HSS. Thus, subtasks in the set are highly repetitive. In LSS, there are 120 different types of subtasks, so the repetition of subtasks in the set is very low. And MSS is a compromise between the HSS and LSS with 50 type of subtask. In addition, each set is executed 20 times in our experiment, and the makespan of a set is the average complete time. During the experiment, we continuously submit these 100 tasks of each set to the system with the task arrival rate 10/s. The execution results of three kind of sets under different similarity weight factors are shown in the Figure 4. As shown in the figure, the completion time of the same task set varies under different weighting factors λ and the makespan of three different sets is significantly reduced when λ in the range of $[0.4, 0.8]$. Among them, HSS, MSS, LSS can obtain the minimum completion time when $\lambda = 0.7$. Since subtask similarity weight λ and resource requirement similarity weight μ satisfy $\lambda + \mu = 1$ as mentioned in Section IV, the system can achieve better performance of task scheduling under two similarity weight factors $\lambda = 0.7$ and $\mu = 0.3$, which means that subtask similarity has a greater impact on tasks makespan by reducing task reconfiguration time than resource requirement similarity in our experiments. We use these most appropriate weight factors in the later experiments.

2) CONVERGENCE OF GACO ALGORITHM

In our two-stage scheduling algorithm, GACO algorithm is an important part and the convergence of GACO is an important factor for the performance of our approach. In order to verify the convergence of GACO, we randomly assign 300 subtasks to a subtask ready queue for subtask-stage scheduling simulation and we set GACO parameters as $Gen_{min} = 40$, $Gen_{max} = 80$, $Evo_{min} = 4\%$, $popsize = 30$, $M = 40$, $\alpha = 0.2$, $\beta = 0.8$, $Q = 5$, $Q' = 4$, $\delta = 0.017$, which makes a faster convergence of GACO through our extensive experiments. Then, in the experiment, we compare GACO algorithm with GA algorithm and ACO algorithm. We run each algorithm 15 times and the measured subtask makespan is the average time of 15 executions. The makespan in this process is the time interval from the start of subtask scheduling to the completion of all subtasks. As shown in Figure 5, with the increase of iterations, the total subtask makespan

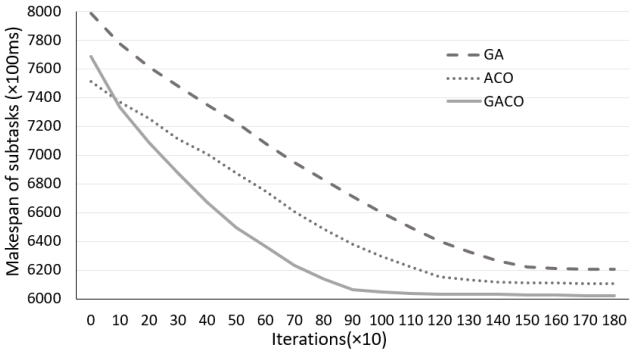

FIGURE 5. Convergence of GACO.

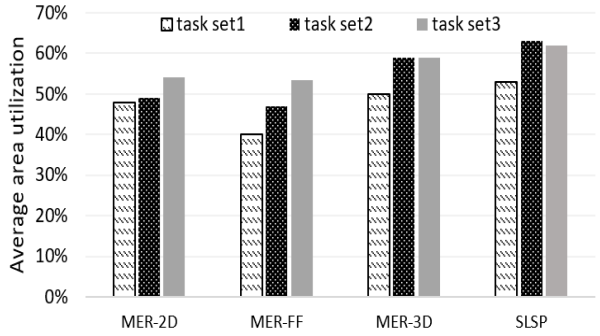

FIGURE 6. Average area utilization of subtask sets on a single FPGA.

of each algorithm converges gradually. When the number of iterations reaches 950, the makespan of GACO curve gradually stabilizes. However, the makespan of GA and ACO algorithms tends to be stable at 1200 and 1500 iterations, respectively. In addition, we can find that the final makespan of GACO is less than that of GA and ACO which means that GACO converges faster than GA and ACO, and the solution obtained is better.

C. PERFORMANCE ANALYSIS

1) EVALUATION OF SLSP METHOD

We verify SLSP method from average area utilization and subtask makespan for the subtask-level scheduling in a single FPGA. We define the average area utilization (AAU) as shown in Eq12.

$$\frac{\sum_{i=1}^{n} ST_i^{texe} \times ST_i^{w} \times ST_i^{h}}{makespan \times F_w \times F_h}, \tag{12}$$

where n represents the number of all the subtasks and *makespan* is the complete time of the subtask set on the FPGA.

In this experiment, we set the number of FPGAs to 1 and there is no task-stage scheduling. In our experiments, we randomly generate three sets with 300 tasks described in table 1 and the tasks arrive at 10/s. Each of set allows that the same type of task can occur multiple times. Figure 6 and Figure 7 represent the result of our SLSP method compared with other methods implemented in our system.

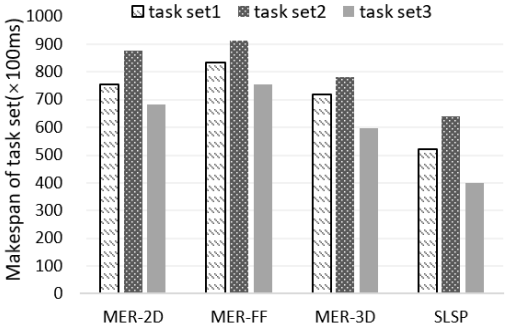

FIGURE 7. Makespan of subtask sets on a single FPGA.

Figure 6 shows the average area utilization of the reconfiguration region caused by different scheduling methods. As we can see, compared with other methods, our method can obtain the largest area resource utilization. Figure 7 analyses the makespan of the whole set, which is the total time from the start of scheduling to the completion of all tasks, including scheduling time, configuration time, and execution time. As shown in Figure 7, SLSP method obtains the minimum makespan for all three different task sets compared with other methods. This is because SLSP method not only considers the placement of a single task to reduce resource fragmentation, but also considers the impact of different scheduling orders on reconfiguration and resource occupancy in multi-task scenarios by using GACO algorithm. It is worth mentioning that although MRE-3D algorithm is very similar to our algorithm in placement, SLSP algorithm shows better performance by considering both scheduling order and placement location.

2) EVALUATION OF OUR TWO-STAGE SCHEDULING APPROACH ON MULTI-FPGA

In order to verify the performance of our two-stage scheduling, we compare it with the hybrid methods which are composed of the common multi-node scheduling methods and the single-FPGA placement methods mentioned above. In order to better reflect different multi-FPGA scenarios, we set the number of FPGAs in the system to 3-5. We randomly submit 400 tasks continously and obtain the performance of the system under different scheduling methods, including the makespan of task sets, resource area utilization and different hardware resources utilization of an FPGA. Note that the makespan of a task set is the time from the start of the first task received by the system to the completion of task set processing.

In our experiments, we obtain the resource area utilization on FPGAs at different time under different scheduling methods. Figure 8 illustrates resource utilization rate at different times on the first FPGA when the number of FPGA is 3. As Figure 8 shows, the resource utilization rate generated by each method is constantly fluctuating. But we can observe that the resource utilization rate of our approach is relatively stable and it basically maintains at about 79% which is significantly higher than the other four methods. This shows that our approach can keep the reconfigurable resources highly utilized for a long time through appropriate task scheduling,

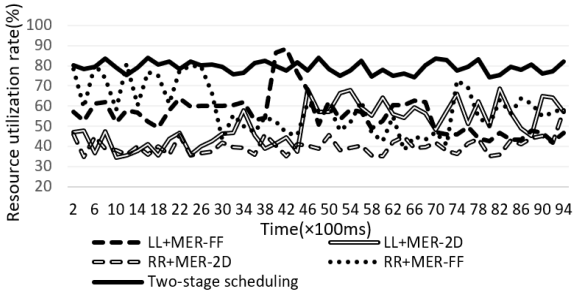

FIGURE 8. Resource utilization rate of the task sets.

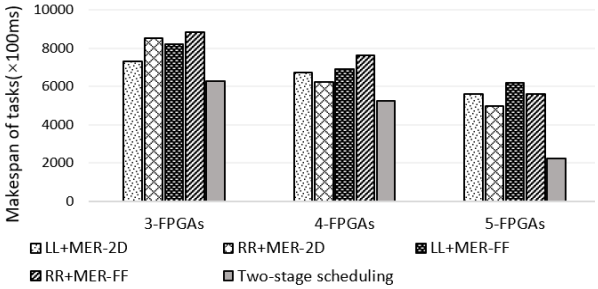

FIGURE 9. Makespan of the task sets.

TABLE 2. Average utilization rate of different resources with three FPGAs.

FPGA	Methods	CLB Util	BRAM Util	DSP Util
1st FPGA	LL+MER-FF	72.3%	52.9%	43.8%
1st FPGA	LL+MER-2D	63.2%	58.7%	36.8%
1st FPGA	RR+MER-FF	56.7%	34.5%	59.1%
1st FPGA	RR+MER-2D	65.3%	45.7%	32.2%
1st FPGA	Our method	74.4%	75.9%	72.3%
2nd FPGA	LL+MER-FF	31.3%	69.9%	20.8%
2nd FPGA	LL+MER-2D	42.2%	32.7%	70.8%
2nd FPGA	RR+MER-FF	56.7%	34.5%	59.1%
2nd FPGA	RR+MER-2D	62.3%	28.7%	46.2%
2nd FPGA	Our method	79.4%	76.9%	73.8%
3rd FPGA	LL+MER-FF	27.3%	45.9%	76.8%
3rd FPGA	LL+MER-2D	34.2%	63.7%	45.8%
3rd FPGA	RR+MER-FF	66.7%	65.5%	20.1%
3rd FPGA	RR+MER-2D	34.3%	45.7%	34.8%
3rd FPGA	Our method	70.4%	72.9%	72.1%

subtask scheduling and placement, which makes multitasking more parallel to some extent. In practice, the utilization of reconfigurable resources in each of our FPGAs is similar to that shown in Figure 8.

In addition, in order to explore the utilization of different hardware resources, we calculate the average utilization rate for each type of resources $r \in R = \{CLB, BRAM, DSP\}$ on the each FPGA as follow:

$$\frac{\sum_{i=1}^{n} ST_{ki}^{texe} \times ST_{ki}^{r}}{makespan \times F_r}, \tag{13}$$

where ST_{ki}^{r} is the amount of the resource r required by subtask ST_{ki} and F_r is the amount of the resource r in the reconfigurable region. Table 2 shows the average utilization rate of different resources in each FPGA when the system is configured with three FPGAs. As shown in Table 2, when the system is configured with three FPGAs, LL+MER-FF, LL+ MER-2D, RR+MER-FF, RR+MER-2D have great differences in resource utilization rates of each resource on each FPGA. However, using our scheduling method, the different resource utilization rates on each FPGA are very close, that is, the resources are used equally. This shows that considering similarity of resource requirements among tasks in task-level scheduling is meaningful, which enables tasks to be scheduled to the appropriate computing unit with the least possibility of resource contention so as to make full use of multi-resources through reasonable task scheduling.

Finally, we analyze the impact of different scheduling methods on the makespan of the whole task set under the same number of FPGA. Figure 9 shows the makespan of all tasks with different scheduling and placement methods on our multi-FPGA system. We can observe that when the system is configured with three, four and five FPGAs respectively, our two-stage scheduling method can always get the minimized makespan which is obviously superior to LL+MER2D, RR+MER2D, LL+MERFF and RR+MERFF with the same task set, which proves that our method can reduce the makespan of multi-task under multi-FPGA.

VI. CONCLUSION

In this paper, we propose a two-stage task scheduling approach in partially reconfigurable multi-FPGA systems. At the first scheduling stage, we use the subtask similarity and resource requirement similarity to measure the suitability between a task and each FPGA, which guides the task to be allocated to an appropriate FPGA computing unit and reduces the possibility of reconfiguration and resource contention. At the second scheduling stage, we propose the SLSP algorithm to select an appropriate scheduling order and location for each subtask, which effectively reduces unnecessary reconfiguration and resource fragmentation. Finally, through extensive experiments, we show that our two-stage scheduling approach is superior to other methods, which reduces the makespan of multi-tasks and improves resource utilization by reducing reconfiguration overhead, resource contention and resource fragmentation.

REFERENCES

[1] C.-C. Kao, "Performance-oriented partitioning for task scheduling of parallel reconfigurable architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 858–867, Mar. 2015.
[2] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network FPGA clusters in a heterogeneous cloud data center," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2017, pp. 237–246.
[3] X. Liu, W. Liu, H. Ma, and H. Fu, "Large-scale vehicle re-identification in urban surveillance videos," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2016, pp. 1–6.
[4] M. D. Santambrogio and D. Sciuto, "Design methodology for partial dynamic reconfiguration: A new degree of freedom in the HW/SW codesign," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2008, pp. 1–8.
[5] J. Strunk, T. Volkmer, K. Stephan, W. Rehm, and H. Schick, "Impact of run-time reconfiguration on design and speed—A case study based on a grid of run-time reconfigurable modules inside a FPGA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2009, pp. 1–8.
[6] A. Purgato, D. Tantillo, M. Rabozzi, D. Sciuto, and M. D. Santambrogio, "Resource-efficient scheduling for partially-reconfigurable FPGA-based systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 189–197.

[7] L. Gong and O. Diessel, "Functionally verifying state saving and restoration in dynamically reconfigurable systems," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2012, pp. 241–244.
[8] J. Tabero, J. Septien, H. Mecha, and D. Mozos, "Task placement heuristics based on 3D-adjacency and look-ahead in reconfigurable systems," in *Proc. Asia South Pacific Conf. Design Autom. (ASPDAC)*, Jan. 2006, pp. 396–400.
[9] S. Roman, H. Mecha, D. Mozos, and J. Septien, "Partition based dynamic 2D HW multitasking management," in *Proc. 9th EUROMICRO Conf. Digit. Syst. Design (DSD)*, Sep. 2006, pp. 61–70.
[10] X. Iturbe, K. Benkrid, T. Arslan, C. Hong, and I. Martinez, "Empty resource compaction algorithms for real-time hardware tasks placement on partially reconfigurable FPGAs subject to fault ocurrence," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, Nov./Dec. 2011, pp. 27–34.
[11] J. A. Clemente, J. Resano, and D. Mozos, "An approach to manage reconfigurations and reduce area cost in hard real-time reconfigurable systems," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, 2014, Art. no. 90.
[12] G. Wang, S. Liu, J. Nie, F. Wang, and T. Arslan, "An online task placement algorithm based on maximum empty rectangles in dynamic partial reconfigurable systems," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2017, pp. 180–185.
[13] R. Cattaneo, R. Bellini, G. Durelli, C. Pilato, M. D. Santambrogio, and D. Sciuto, "PaRA-Sched: A reconfiguration-aware scheduler for reconfigurable architectures," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2014, pp. 243–250.
[14] E. A. Deiana, M. Rabozzi, R. Cattaneo, and M. D. Santambrogio, "A multiobjective reconfiguration-aware scheduler for FPGA-based heterogeneous architectures," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–6.
[15] C. Jing, Y. Zhu, and M. Li, "Energy-efficient scheduling on multi-FPGA reconfigurable systems," *Microprocessors Microsyst.*, vol. 37, nos. 6–7, pp. 590–600, 2013.
[16] G. Dai, Y. Shan, F. Chen, Y. Wang, K. Wang, and H. Yang, "Online scheduling for FPGA computation in the cloud," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2014, pp. 330–333.
[17] Y. He, J. Liu, and H. Sun, "Scheduling functionally heterogeneous systems with utilization balancing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2011, pp. 1187–1198.
[18] M. E. Belviranli, L. N. Bhuyan, and R. Gupta, "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, 2013, Art. no. 57.
[19] T. Bridi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A constraint programming scheduler for heterogeneous high-performance computing machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2781–2794, Oct. 2016.
[20] K. Chronaki, A. Rico, M. Casas, M. Moretó, R. M. Badia, E. Ayguadé, J. Labarta, and M. Valero, "Task scheduling techniques for asymmetric multi-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2074–2087, Jul. 2017.
[21] M. Hakem and F. Butelle, "Dynamic critical path scheduling parallel programs onto multiprocessors," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2005, p. 7.
[22] M. I. Daoud and N. Kharma, "Efficient compile-time task scheduling for heterogeneous distributed computing systems," in *Proc. Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Jul. 2006, p. 9.
[23] Y. Song, L. Liu, H. Ma, and A. V. Vasilakos, "A biology-based algorithm to minimal exposure problem of wireless sensor networks," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 3, pp. 417–430, Sep. 2014.
[24] L. Liu, Y. Song, H. Zhang, H. Ma, and A. V. Vasilakos, "Physarum optimization: A biology-inspired algorithm for the Steiner tree problem in networks," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 818–831, Mar. 2015.
[25] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan, "Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 194–203.
[26] L. Liu, X. Zhang, and H. Ma, "Optimal node selection for target localization in wireless camera sensor networks," *IEEE Trans. Veh. Technol.*, vol. 59, no. 7, pp. 3562–3576, Sep. 2010.
[27] D. Zhao, X.-Y. Li, and H. Ma, "How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr./May 2014, pp. 1213–1221.
[28] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with openstack," in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2014, pp. 109–116.
[29] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, "Virtualized execution runtime for FPGA accelerators in the cloud," *IEEE Access*, vol. 5, pp. 1900–1910, 2017.
[30] G. Charitopoulos, I. Koidis, K. Papadimitriou, and D. Pnevmatikatos, "Run-time management of systems with partially reconfigurable FPGAs," *Integration*, vol. 57, pp. 34–44, Mar. 2017.
[31] H. Yu, "A hybrid GA-based scheduling algorithm for heterogeneous computing environments," in *Proc. IEEE Symp. Comput. Intell. Scheduling (SCIS)*, Apr. 2007, pp. 87–92.
[32] Y. Xu, K. Li, T. T. Khac, and M. Qiu, "A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems," in *Proc. 14th IEEE Int. Conf. High Perform. Comput. Commun. (HPCC)*, Jun. 2012, pp. 639–646.
[33] I. Alaya, C. Solnon, and K. Ghedira, "Ant colony optimization for multi-objective optimization problems," in *Proc. IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, Oct. 2007, pp. 450–457.
[34] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.
[35] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Raleigh, NC, USA: Microelectronics Center of North Carolina, 1991.
[36] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 30th Annu. Int. Symp. Microarchitecture (MICRO)*, Dec. 1997, pp. 330–335.

ZICHANG SUN received the B.S. degree in software engineering from Yanshan University, China, in 2017. She is currently pursuing the M.S. degree in computer science and technology with the Beijing University of Posts and Telecommunications (BUPT), China. Since 2017, she has been with the Beijing Key Laboratory of Intelligent Communication Software and Multimedia, BUPT, and has conducted research on heterogeneous parallel computing and task scheduling.

HAITAO ZHANG received the B.S. degree in mathematics from the Dalian University of Technology, China, in 2006, the M.S. degree in computer science from Northeastern University, China, in 2008, and the Ph.D. degree in computer science from the Beijing University of Posts and Telecommunications, China, in 2012. From 2010 to 2011, he was a Visiting Scholar with the Department of Computer Science, Illinois Institute of Technology, Chicago. He is currently an Associate Professor with the School of Computer Science, Beijing University of Posts and Telecommunications. His research interests include parallel and distributed computing, cloud computing, big data, and image/video analysis. He is the Editor-in-Chief of over ten ITU-T standards related to visual surveillance and cloud computing.

ZEHAN ZHANG received the B.S. degree in computer science and technology from the Wuhan University of Technology (WHUT), China, in 2018. He is currently pursuing the M.S. degree in computer science and technology with the Beijing University of Posts and Telecommunications (BUPT), China. Since 2018, he has been with the Beijing Key Laboratory of Intelligent Communication Software and Multimedia, BUPT, and has conducted research on heterogeneous parallel computing and task scheduling.