Chinese Journal of Electronics
Vol.30, No.6, Nov. 2021

A Survey: FPGA-Based Dynamic Scheduling of Hardware Tasks

LI Tianyang[1,2], ZHANG Fan[1,2], GUO Wei[1,2], SUN Mingqian[3] and CHEN Li[1,2]

(1. *PLA Strategic Support Force Information Engineering University, Zhengzhou 450002, China*)

(2. *National Digital Switching System Engineering and Technological R&D Center, Zhengzhou 450003, China*)

(3. *Southeast University, Nanjing 210000, China*)

Abstract — To meet the increasing computing needs of various application fields, Field programmable gate array (FPGA) has been widely deployed. In FPGA-based processing, hardware tasks can be better accelerated by allocating appropriate computing resources. Therefore, FPGA-based hardware task scheduling has become one of the mainstream research directions in academia and industry. However, the optimization objectives of existing FPGA-based hardware task scheduling methods are relatively scattered. In this regard, this paper summarizes the research status of hardware task dynamic scheduling from the three essential elements of FPGA processing: time, resources, and power consumption. This paper analyzes, sorts out, categorizes the ideas and implementations of various scheduling methods and analyzes and evaluates optimization effects of various scheduling methods from multiple dimensions. Then, the shortcomings of the existing methods are summarized and some practical applications are introduced. Finally, the research direction of task scheduling based on FPGA is prospected and summarized.

Key words — Field programmable gate array, Task scheduling, Optimization, Resource management, Dynamic reconfiguration.

I. Introduction

In recent years, artificial intelligence, big data, cloud computing, and other fields have developed rapidly. However, due to the black silicon effect[1], the general computing architecture composed of Central processing units (CPUs) alone can not meet the increasing demand of computing in various fields. The CPU+FPGA heterogeneous computing system has shown great potential in high performance, low power consumption and reconfigurability, and has been widely used in academia and industry. Typical cases such as Microsoft[2], Amazon[3], and other cloud service providers deploy FPGAs in their data centers to support various compute-intensive applications[4]. Modern FPGA supports Dynamic partial reconfiguration (DPR)[5], which allows multi-task parallel processing, and dynamically reconfigures part of FPGA resources at runtime without affecting the task running on other parts. This processing allows limited computing resources to be reused promptly. In addition, this gives FPGA greater flexibility and computing power when handling large-scale compute-intensive applications. Compared with the traditional general architecture, the advantage of CPU+FPGA heterogeneous computing architecture is that the system can dynamically allocate computing resources for each task according to application requirements and resource status. The system improves processing performance through hardware acceleration also ensure high flexibility and adaptability. However, achieving efficient dynamic scheduling of hardware tasks on FPGA is a very challenging problem. It is also the key to giving play to the processing capabilities, flexibility, and adaptability of the entire architecture[6].

FPGA-based hardware task scheduling is an NP-hard problem[7,8]. In the application process, designers need to consider factors such as when to configure, perform tasks, and arrange tasks. At present, many researchers have optimized various problems involved in scheduling methods and achieved some optimization results. However, we need to clearly understand the different scheduling methods' multiple problems and limitations if we want to realize multi-dimensional efficient task scheduling. Therefore, this paper gives a comprehensive overview of the dynamic scheduling

Manuscript Received Dec. 9, 2020; Accepted Apr. 7, 2021. This work is supported by the National Natural Science Foundation of China (No.61521003).

 DOI:10.1049/cje.2021.07.021

method of FPGA computing tasks. It is expected to compare and analyze the advantages and disadvantages of the existing methods from various perspectives, and combining the actual needs in field applications, clarify the future research of FPGA-based task scheduling methods.

The main contributions of this paper are as follows.

1) The model and optimization problem of FPGA-based dynamic scheduling of hardware tasks involved in the current research are analyzed.

2) The latest progress of FPGA-based dynamic scheduling of hardware tasks is summarized from three optimization perspectives: time, resource and power consumption.

3) The typical algorithms for the various optimization techniques under each optimization perspective are analyzed.

4) The researches are compared and summarized from four aspects, some practical applications are introduced, and the future work is discussed.

The rest of this paper is organized as follows. Section II introduces the FPGA-based hardware task dynamic scheduling system model and task model, and analyzes the optimization problems. Section III introduces the latest progress of FPGA-based hardware task dynamic scheduling. Section VI compares and summarizes the scheduling methods. Section V introduces some practical applications. Section VI discusses the challenges and future research. Section VII concludes the paper.

II. Scheduling Model and Optimization Problem

1. System model

The scheduling system model consists of a CPU scheduling module and FPGA computing resources, as shown in Fig.1. The CPU scheduling module includes a scheduler, a placer, and a loader. The scheduler is responsible for receiving application tasks, analyzing task requirements, determining the execution order of tasks and the required computing resources according to task data dependencies and system status, and correctly transitioning the task status during processing. The placer is responsible for determining the appropriate placement position of the hardware task on the FPGA with limited resources and feedback the hardware resource status to the scheduler in real-time to ensure the feasibility of scheduling. If there is an available placement, the loader is responsible for loading the task's configuration data to the designated location on the FPGA. FPGA computing resources are abstracted as heterogeneous resource arrays, including Configurable logic block (CLB), Block random access memory (BRAM), and Digital signal processor (DSP), responsible for accelerating computationally intensive hardware tasks. Different hardware tasks will occupy computing resources in different sizes of rectangular areas on the FPGA during processing. Therefore, task placement can be regarded as a boxing problem[9]. This model can achieve a tight arrangement of tasks and improve FPGA resource utilization. On the other hand, the task placement's high flexibility makes it quite challenging to find the optimal allocation plan for the task. The task scheduling algorithm design will become more complicated.

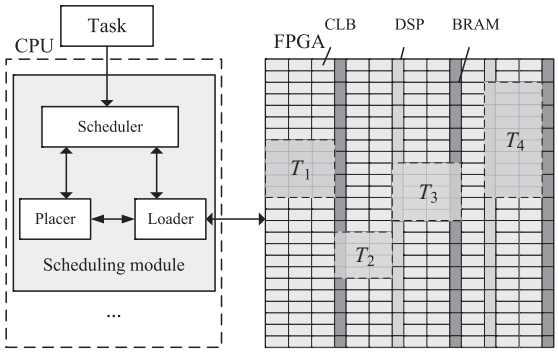


Fig. 1. Scheduling system model

Unfortunately, it is a challenge to achieve the above-mentioned idealized scheduling model in practice. Therefore, the realistic scheduling model should follow many technical bottlenecks in the current FPGA design flow, such as DPR constraints[10,11]: 1) The FPGA is divided into a static area and a dynamic area. The static area will not be reconfigured, including fixed processing structures during the execution of application tasks. The dynamic area is further divided into several Reconfigurable regions (RR). Each RR can be reconfigured independently without affecting the operations of other RRs. 2) RR can only place hardware IP core called "Reconfigurable module (RM)". Each RM can handle subtasks separately, but RR can only place one RM at a time. 3) FPGA Internal configuration access port (ICAP) can only process one reconfiguration request simultaneously and cannot reconfigure multiple RRs at the same time. Under these strict technical constraints, the FPGA system model is shown in Fig.2. After the FPGA is divided into multiple areas, each task can only be placed in the designated region.

2. Task model

FPGA computing resources are limited and cannot meet the needs of large-scale applications at the same time. Therefore, applications are usually divided into many interrelated tasks. The tasks are configured to perform specified functions on a specific area of the FPGA, and the computing resources are time-shared to complete the entire application processing. Generally, a Directed acyclic graph (DAG)[12] is used to represent the application. Fig.3 shows an example of a DAG, in which

each node and directed edge represent tasks and data dependencies between tasks. T_2 and T_3 must be executed after T_1, and T_4 must wait for T_2 and T_3 to complete.

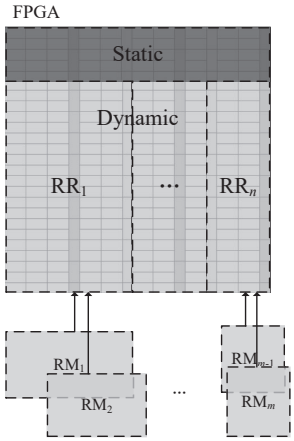


Fig. 2. FPGA system model under strict constraints

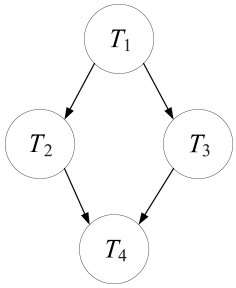


Fig. 3. DAG

3. Scheduling optimization problem

This paper considers FPGA resource constraints, task time-related constraints, and multi-dimensional optimization goals to optimize FPGA-based hardware task scheduling.

1) Resource constraints. The resources occupied by all tasks $\sum T_i$ cannot exceed the total number of resources on the FPGA:

$$\sum w_i \leq W \tag{1}$$

$$\sum h_i \leq H \tag{2}$$

$$\sum (clb_i, dsp_i, bram_i) \leq (CLB, DSP, BRAM) \tag{3}$$

Among them, w_i and h_i represent the width and height of the resource area occupied by T_i; W and H respectively represent the width and height of the FPGA total resource area; $clb_i, dsp_i, bram_i$ respectively represent the CLB, DSP, BRAM occupation by T_i; $CLB, DSP, BRAM$ respectively represent the total amount of various resources on the FPGA.

2) Time constraints. To avoid the task being rejected, the entire processing of the task must be completed before the deadline of the task is reached:

$$T_{\text{total}} = t_{\text{rec}} + t_{\text{exe}} + t_{\text{com}} \tag{4}$$

$$T_{\text{total}} + t_{\text{cur}} \leq t_{\text{dead}} \tag{5}$$

Among them, T_{total} represents the time required for the entire processing of the task; t_{rec} represents the task configuration time; t_{exe} represents the task execution time; t_{com} represents the task communication time; t_{cur} represents the current time; and t_{dead} represents the task deadline.

3) Optimization goals. Different application scenarios need to be optimized for different goals. For example, applications with high real-time requirements need to reduce time overhead; applications with limited FPGA resources need to make full use of resources; and low-cost applications need to reduce power consumption as much as possible. Therefore, in this paper, the three dimensions of time, resources, and power consumption comprehensively consider the optimization problems, including minimizing the processing time T of the application, maximizing the utilization rate U of FPGA resources, and minimizing the power consumption P of the FPGA.

$$\min T \text{ or } \max U \text{ or } \min P \tag{6}$$

On the other hand, in FPGA hardware task scheduling, multi-objective T, U, P can be optimized simultaneously. However, this may compromise time, resource, and power consumption, as discussed in Section IV.

III. FPGA-Based Hardware Task Scheduling Method

This paper selects the hardware task scheduling research work in the past two decades from IEEE, ACM, Elsevier, Springer conferences, and journals to review, classify the papers' scheduling methods. Aiming at the optimization goals described in Section II.3, the paper divides the hardware task dynamic scheduling methods into three categories: reducing time overhead, optimizing resource utilization, and reducing leakage power consumption. These scheduling methods are analyzed in detail in this section. The optimization effects of these scheduling methods are compared in detail in Section IV.

1. Scheduling to reduce time overhead

The time overhead that can be optimized for FPGA hardware task scheduling mainly includes the lengthy task reconfiguration time and the data communication overhead between tasks. According to the different optimization objects, the scheduling method for reducing time overhead is further divided into reducing reconfiguration overhead and optimizing communication overhead.

1) Scheduling to reduce reconfiguration time

Angermeier and Teich[13] compared scheduling meth-

ods that do not consider reconfiguration and consider reconfiguration. The results show the necessity of considering reconfiguration overhead in hardware task scheduling. Therefore, specific methods must be used to reduce or hide the massive overhead of reconfiguration effectively. The current scheduling strategy for reducing reconfiguration overhead is mainly based on the two basic ideas of "configuration prefetching" and "module reuse".

Configuration prefetching[14] means that if the configuration port is available and the task is ready to be placed on the FPGA, the task can be configured in advance during the execution of other tasks to achieve hidden reconfiguration time. This method dramatically reduces the impact of reconfiguration on performance, thereby improving the overall execution efficiency of the application. Fig.4 shows the different scheduling results for the DAG in Fig.3 without prefetching and with prefetching. In the absence of prefetching, T_2 and T_3 can only be configured after T_1 processing is completed. Although T_2 and T_3 can be processed in parallel, FPGAs usually have only one reconfigure port and cannot process two configuration requests simultaneously. T_3 can only be configured after T_2 is configured. When T_2 and T_3 are processed, T_4 starts to configure and execute. In the case of prefetching, once the reconfigure port is free, the next task to be processed can be configured in advance. The configuration phases of T_2, T_3 and T_4 can be completed in advance during the execution of other tasks, thereby hiding the reconfiguration time.

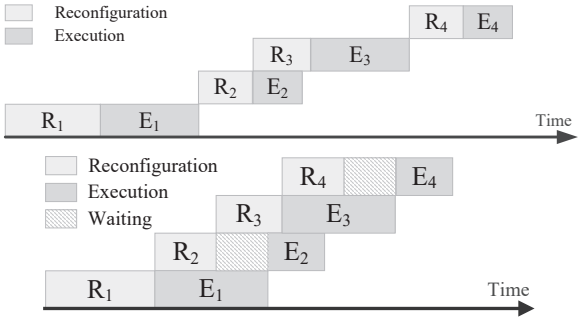


Fig. 4. Without prefetching and with prefetching

Banerjee *et al.*[15] first proposed using configuration prefetching in FPGA hardware task scheduling to hide the reconfiguration time. In their work, each task in the application uses the predecessor's output as its input. The heuristic algorithm is used to configure the next task in advance during the execution of the current task, which reduces the impact of reconfiguration overhead on performance. However, this method is only suitable for applications with linear data dependencies. For more general applications, Resano *et al.*[16] proposed a hybrid configuration prefetch scheduling strategy to avoid runtime overhead caused by high computational complexity[17]. At the beginning of the scheduling, firstly, regardless of the impact of the reconfiguration time on the scheduling results, the branch and bound algorithm is used to determine the initial scheduling table. The As late as possible (ALAP) algorithm is used to calculate the critical path. When the task is running, reconfiguration is considered. According to the initial schedule and critical path, the next task's configuration phase is completed in parallel when the task is executed. In some cases, 93% to 100% of the reconfiguration overhead can be hidden. Khuat *et al.*[18] believed that using the branch and bound algorithm to search all Integer linear programming (ILP) solutions and find the optimal solution also requires a lot of computing time, so they proposed a fast task placement feasible region search method. Considering that the number of BRAM occupied by the task is less than the number of CLB resources, search for a feasible region that meets the BRAM requirements, and all feasible regions can be covered. Then, from the limited scan results, select feasible regions that also meet other resource requirements and place tasks in them. In order to avoid resource competition, a method for avoiding feasible region conflicts is further proposed. It evaluates the situation after the current task occupies each feasible area and selects the position with the most negligible influence on the number of the feasible regions for subsequent tasks. The experimental results show that compared with the method without configuration prefetching, the application's total execution time is reduced by 22%. However, the above scheduling algorithms are only suitable for single application processing. Aiming at the FPGA hardware task scheduling problem in continuous multiple application scenarios, Ramezani[19] proposed a Forefront-fetch technology. During an application's execution, some tasks of the following application to be processed are packaged and configured in advance. However, this method has a sizeable reconfiguration granularity and can only hide part of the reconfiguration overhead.

The above algorithms implicitly implement configuration prefetching, but they cannot completely hide the task configuration overhead. Using queues to manage task configuration in hardware task scheduling can effectively hide the configuration phase of hardware tasks. PATS[20] introduces the concept of task efficiency in task scheduling. It maintains Not released queue (NRQ), Low efficiency queue (LEQ), and Full efficiency queue (FEQ) to manage waiting tasks. The three queues are inserted into inefficient tasks that are not completed by predecessors, inefficient tasks that have not been configured, and high-efficiency tasks that have completed configuration. PATS prioritizes tasks with high task efficiency. During the execution of the current task, the waiting task's status is updated, such as prefetching

the task in the LEQ, which can dynamically change the task efficiency at runtime. Compared with the three classic scheduling algorithms of Early deadline first (EDF), Rate monotonic scheduling (RMS), and round-robin, PATS shortens the total application execution time by an average of 1.45 times. However, if a successor task in the LEQ configures first, and the FPGA does not have free resource space to configure its predecessor task, which results in both the predecessor task and the successor task cannot be executed, and deadlock occurs. As shown in Fig.5, taking Fig.3 DAG as an example, if T_4 is configured before T_3, and the FPGA resource space after configuration cannot configure T_3, both T_3 and T_4 cannot be executed.

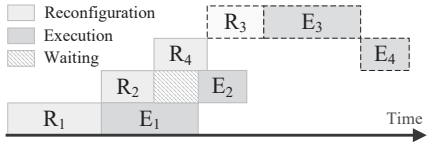


Fig. 5. Scheduling deadlock

To avoid this problem, CPSA[21] modifies the management mode of the unconfigured task queue, and only prefetches all the tasks whose predecessor tasks have completed the configuration, effectively avoiding the scheduling deadlock problem. PaRA-Sched[22] is an automated design method for FPGA hardware task scheduling, which includes three steps: preprocessing, scheduling and post-processing. Preprocessing is to use the task mapping information obtained by the A2B[23] algorithm, explicitly introduce the reconfiguration task in the DAG, and use the dynamic programming algorithm to calculate the critical path information. Scheduling is responsible for assigning start and end time to each task under the premise of satisfying the task data dependency. Postprocessing uses additional edges in the task graph to clarify the execution order of all tasks, which automatically hides the reconfiguration overhead during the scheduling process and improves the system's overall performance.

Configuration prefetching hides the reconfiguration overhead during the scheduling and execution of FPGA hardware tasks, and module reuse[24] is to reduce the reconfiguration overhead further. It reuses as many tasks as possible that have been configured on the FPGA to reduce hardware task configuration times. In the research of many applications, it can be observed that different hardware task modules may have the same or similar functions. For this reason, it is very efficient to reuse the configured tasks, which helps to reduce the total execution time of the application and keep the impact of configuration data transfer on memory bandwidth at a low level.

When the task arrives, RCSched[25] first checks whether there is a reusable task module in an idle state; if there is, it will be reused; otherwise, the Least recently used (LRU) strategy will be used for reconfiguration. When there is no reusable task module and no configuration space, the task is migrated to the software processing unit for execution. Similar work is also a scheduling algorithm proposed by Hariharan and Kannan[26] that uses LRU and optimal replacement strategy. They try to avoid using software processing units and pay more attention to using more efficient hardware resources to process all tasks. When the number of waiting tasks is higher than the total number of hardware processing units, the LRU strategy is used for reconfiguration, and the least recently used task module is replaced; otherwise, the optimal replacement strategy is used to replace the task module with less reconfiguration overhead. However, both of these methods keep the task modules on the FPGA as much as possible, resulting in excessive, unnecessary power consumption. RBS[27] uses the Least probability of reappearance (LPR) strategy to retain only a part of the tasks and select non-important tasks with a small reappearance probability and are large enough to accommodate the new task for replacement. RBS regards tasks with high reconfiguration overhead and high probability of reappearing as important tasks. After execution, it retains its task modules on the FPGA and divides the FPGA into two parts, one for important tasks and the other for non-important tasks. It uses a scan-based algorithm to schedule tasks and dynamically adjusts the two FPGA partitions' size to reduce the possibility of task rejection. As shown in Fig.6, the FPGA is divided into Part A and Part B that can be dynamically adjusted. Part A places non-important tasks, and Part B reserves the task modules to be reused. Taking Fig.3 DAG as an example, suppose T_2 module will be reused by T_4, Part A is used to load T_1 and T_3, Part B loads T_2, when T_2 and T_3 are executed, T_4 directly reuses T_2 module for processing. However, this method is expensive in scanning calculation and relies on the artificially set LPR threshold.

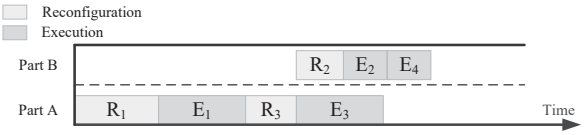


Fig. 6. RBS

In FPGA hardware task scheduling, considering both configuration prefetching and module reuse can reduce the impact of reconfiguration overhead on performance to a greater extent. Deiana *et al.*[28] proposed a scheduling method based on Mixed integer linear programming (MILP). They designed the IS-k algorithm to schedule k

tasks in each iteration to reduce computational overhead. IS-k was verified in a real image analysis case study, and some tasks in the case study were preconfigured and reused task modules. Compared with PaRA-Sched based on the ant colony optimization algorithm, IS-k reduces the total application execution time by 16%–27.7% on average under different k values.

2) Scheduling to reduce communication overhead

When the FPGA reloads the task configuration, the on-chip interconnect structure will also change. Therefore, in FPGA-based hardware task scheduling, it may not be possible to guarantee correct data exchange between tasks or between tasks and external devices if data communication is not considered. On the other hand, data communication between tasks also takes a long time during task processing, especially for compute-intensive tasks with a large amount of data. Therefore, some researchers have researched optimizing communication overhead.

Fekete *et al.*[29] first considered the impact of communication overhead on FPGA hardware task scheduling, regarded communication overhead as a constant, and proposed a model that implicitly expresses communication overhead by increasing task execution time. However, the various assumptions based on the model significantly reduce the reliability of the method. CASA[30] is the first scheduling algorithm that explicitly handles the communication overhead between FPGA hardware tasks. It uses a modified Flow scanning (mFS) algorithm to search for FPGA free computing resources to place tasks. Data communication between tasks and between tasks and external devices is regarded as specific constraints in the scheduling. A Locked communication scheduling (LCS) method is proposed to deal with communication constraints: When a task is scheduled, the required communication bus is immediately locked. The bus is released until the end of the data communication, ensuring the data communication demand. The communication overhead problem in task scheduling will be more complicated for FPGAs with heterogeneous computing resources. Abdessamad *et al.*[31] carried out mathematical modeling for the scheduling problem in the CPU+FPGA architecture, proposed a nonlinear model and a linear model of heterogeneous communication scheduling, and used the data dependency relationship to reduce the number of constraints of the linear model and realize the solution of smaller-scale applications in a short time. The MILP model's limitation is that when the scale of the application becomes larger and larger, the algorithm's performance will drop sharply or even impossible to solve. In response to this problem, Abdallah *et al.*[32] proposed two heuristic algorithms: Multithreaded simulated annealing (MSA) algorithm and Multithreaded genetic algorithm (MGA) realize parallel task processing and solve the scheduling problem of larger-scale applications considering the communication between tasks.

However, the above method only regards communication overhead as a constraint condition for FPGA-based hardware task scheduling and does not substantially reduce communication overhead. Ahmadinia *et al.*[33] developed a dynamic scheduling algorithm that reduces communication overhead. It minimizes the communication overhead between task modules by placing tasks with data dependencies as close as possible. The simulation experiment uses task sets of different scales. Compared with the First-fit (FF) and Best-fit (BF) methods, the communication overhead of this method is reduced by 12.1%–90.2%. However, this method does not consider FPGA on-chip interconnect resources. CA-MAE[34] considers the actual communication channel, local bus, system bus, and peripheral bus of FPGA, as shown in Fig.7. A communication coefficient is introduced in the scheduling to measure the relationship between task location and communication overhead, and task scheduling is optimized by placing hardware tasks closer to the communication channel, which avoids excessive occupation of the system bus due to tasks being far away from the channel and can reduce communication overhead.

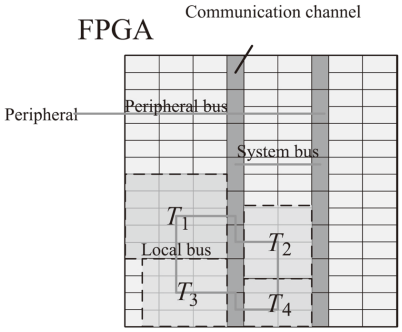


Fig. 7. CA-MAE task communication model

RDMS[35] considers data communication and resource utilization between tasks in scheduling, and configures tasks in groups, especially tasks with data dependencies, to minimize communication overhead between tasks from reducing data movement. However, if tasks with data dependencies are not grouped into the same group, it may cause more significant communication overhead between groups. Mathematical analysis shows that the worst-case time complexity of the algorithm is $O(n^3W^2)$, n is the total number of tasks, and W is the upper bound of the number of groups. To avoid this problem, Yoosefi and Naji[36] introduced the idea of task clustering into FPGA-based hardware task scheduling and proposed a communication-aware

clustering scheduling algorithm. Similar to task grouping in Ref.[35], tasks are clustered into the same cluster. Then a penalty-based merging strategy is used to merge the two clusters with the least cost of merging operation to eliminate communication between tasks in different clusters. However, the experimental results show that when the communication overhead between task clusters is too significant, the number of clusters that can be merged is small. The effect of reducing the communication overhead between clusters is not apparent.

2. Scheduling to optimize resource utilization

Due to the limited number of FPGA computing resources, improper use of resources is one of the critical reasons why tasks are rejected. If the task placement that has been placed on the FPGA is relatively scattered, it will cause the FPGA to lack available computing resources with a large enough area. At this point, even if the FPGA has enough resources to meet the resource requirements of the newly arrived task, the task cannot be configured because the resources are fragmented, resulting in the task being rejected. Using efficient resource or task management strategies in the FPGA hardware task scheduling can effectively avoid this problem. According to different management objects, the scheduling method for optimizing resource utilization is further divided into free resource management scheduling and task management scheduling.

1) Scheduling of free resource management

The free resource management scheduling method maintains an FPGA free resource area list, which contains the location information of each continuously available computing resource on the current FPGA, which is dynamically updated during the scheduling. When placing tasks, select the positions where the tasks can be placed in the FPGA free resource area list, effectively avoiding resource fragmentation.

Bazargan and Kastner[37] first studied this kind of problem. They proposed a Keeping maximal empty rectangle (KMER) algorithm to manage FPGA free resources, divide the FPGA free area, save it as a non-overlapping rectangular set, and assign each task to an appropriate rectangular area. However, this method sacrifices placement quality because tasks cannot be arranged tightly, resulting in rejection of tasks that could be placed. As shown in Fig.8(a), assuming that T_1 and T_2 have been placed on the FPGA, T_3 and T_4 are waiting tasks, and the free area of the FPGA is divided into four areas A, B, C, and D, where T_3 can be placed on area D. Although the hardware resources are sufficient, T_4 will be rejected because it does not have a large enough free area. On the other hand, as the FPGA area gradually increases, the cost of searching out all the free areas of the FPGA will increase. Aiming at the high cost of time for KMER to search for free areas, Walder *et al.*[38] proposed a hash matrix method, which can find a suitable rectangular area in constant time, but introduces the additional cost updating the hash matrix. To solve the high rejection rate of KMER tasks, Marconi *et al.*[39] proposed an intelligent merging algorithm to extend KMER. When there is no large enough free area to place the newly arrived task, the adjacent non-overlapping free areas can be merged to form a larger rectangular area until there is enough space to place the new task. As shown in Fig.8(b), T_4 can be placed by merging area A and area B to avoid rejection of T_4. At the same time, the performance of the algorithm has been improved. Compared with KEMR, the speed of the algorithm is increased by 1.72 times.

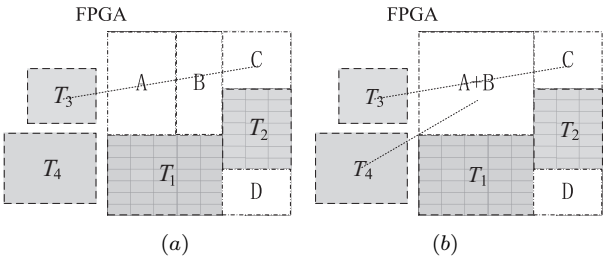


Fig. 8. KEMR and Intelligent merging KEMR

Handa and Vemuri[40] modified the management method of free resources, saving FPGA's free area as rectangles that overlap each other and identify the Maximal empty rectangle (MER). As shown in Fig.9, assuming that 1 and 2 have been laid out on the FPGA, MER_1, MER_2, MER_3 can be identified. Compared with the KEMR as mentioned above and other algorithms, under the same scheduling situation, the free area where tasks can be placed is more substantial, which reduces the probability of task rejection. Similar work is the TT-KAMER[41]– an algorithm that calculates and manages MER based on the upper boundary of the task and the resource's lower boundary. It uses a linked list to store the largest free rectangles after placing different tasks, updates the linked list while tasks are being executed, and achieves better resource utilization. Wang *et al.*[42] proposed two heuristic methods based on MER: MER-3D-Contact and MER-Resource-Contact. The former puts newly arrived tasks on the resource boundary or adjacent to other tasks as much as possible. The latter sorts and prioritizes tasks according to resource type requirements. Both methods improve resource utilization.

2) Scheduling of task management

Ahmadinia *et al.*[43] believe that the number of resource areas usually occupied by tasks is less than the number of FPGA free areas. The location of management tasks can be more efficiently avoided FPGA resource fragmentation. The task management scheduling strategy

is mainly developed around the two basic ideas of "non-preemptive" and "preemptive".

Non-preemptive FPGA hardware task scheduling is to determine the location of the task before the task is loaded on the FPGA. Once the task is configured, it will not be changed. This method avoids multiple reconfigurations of the same task and reduces computational complexity. However, it may only improve resource utilization to a lesser extent[44].

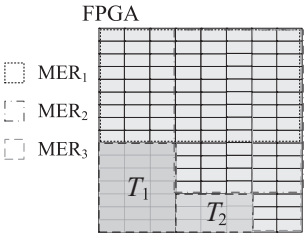


Fig. 9. MER

Quad-Corner[45] considers the 2D resource model of FPGA and places the tasks at the four corners of the FPGA as far as possible. A large free area is reserved in the center for tasks that may occupy a large area in the future. As shown in Fig.10, assuming that T_1, T_2, T_3 and T_4 have been configured on the FPGA, and T_5 is a waiting task. Task scheduling that does not use Quad-Corner may cause T_5 to be rejected due to resource fragmentation. Quad-Corner places T_1, T_2, T_3 and T_4 in the four corners of the FPGA respectively, and then T_5 can be placed in the central free area for processing. However, this method disperses the tasks, resulting in a huge communication overhead between tasks.

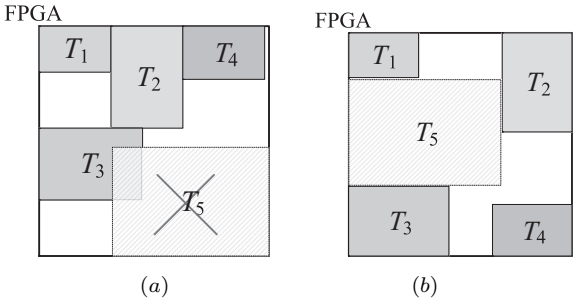
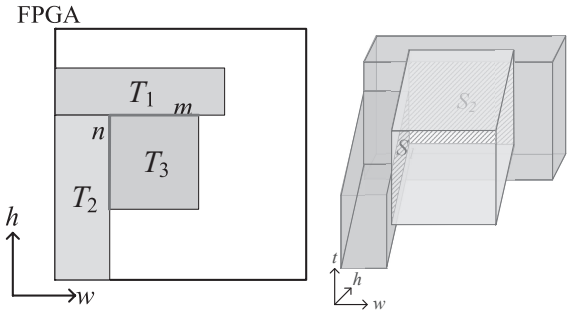


Fig. 10. Quad-Corner algorithm

In order to avoid the introduction of additional communication overhead while minimizing resource fragmentation, some research workplaces multiple tasks closely. Tabero *et al.*[46] designed a scheduling method based on the adjacency value of task modules, in which the sum of the length of the contact between the placement position of the new task and the adjacent placed task or resource boundary is the 2D adjacency value, and 2D adjacency value is multiplied by the length of time the new task is adjacent to other tasks or boundaries to obtain the 3D adjacency value. As shown in Fig.11, assuming that the newly arrived T_3 placement position is adjacent to T_1 and T_2, the sum of adjacent edges m and n is the 2D adjacency value; considering the time dimension, the sum of the contact areas S_1 and S_2 is the 3D adjacency value. The larger the adjacency value, the more compact the task is placed, and the less resource fragmentation. Therefore, the task will be preferentially placed in the candidate position with a large 3D adjacency value. However, this method needs to calculate the 3D adjacency values of all the candidate placement positions, and the calculation cost is relatively high. In order to solve this problem, 3DC[47] associates the newly arrived task with the scheduled previous task and the next task. It only calculates the candidate position with the best starting time of the new task, reducing the computational overhead. Similar to the scheduling strategy based on 3D adjacency values, Xu *et al.*[48] proposed a scheduling algorithm based on 3D fragmentation, which considered the contact surface area between the task and the free area after placing, and reduced the probability of task rejection without adding additional time overhead. However, the anti-fragmentation scheduling method measured by adjacency value is not suitable for FPGAs with heterogeneous resources. The FAReP[49] algorithm calculates the Fragmentation coefficient (FC) based on the "independence" of the candidate placement position of the task on the FPGA, and also considers the module reuse. According to the historical multiplexing times num, reconfiguration time t, and FC value, the task module with the smaller reconfiguration cost $c = num \times t$ will be deleted first. If the reconfiguration cost c of multiple tasks is similar, the task module with the larger FC will be deleted first.



(a) 2D-Adjacency (b) 3D-Adjacency

Fig. 11. Adjacency value

Preemptive FPGA hardware task scheduling is task relocation[50], which can release the resources occupied by the current task in advance and move the task from one reconfigurable area to another, effectively avoiding resource fragmentation. This allows the computing system to process more tasks before the application task deadline

arrives, reduces the probability of task rejection, and maximizes FPGA resource utilization. However, it will perform multiple reconfigurations for the same task, severely reducing system performance.

Diessel and Elgindy[51] reorder the tasks to be processed and the tasks currently being executed, placing the high-priority waiting tasks in a position that minimizes resource fragmentation. If the position is occupied by the task currently being executed, move the task being processed in the same direction to arrange the tasks together closely. However, this method assumes that all tasks can be suspended at any time, which is unacceptable for real-time applications. Gericota *et al.*[52] introduced a dynamic relocation technology that allows moving task modules in execution without suspending tasks but did not analyze the computational overhead of this method.

Unfortunately, none of the above scheduling methods have been implemented in actual hardware due to existing technical bottlenecks. But some researchers considered the real world's technical limitations and carried out related research work. RTSM[53] implements hardware implementation on the ZedBoard platform, saves the hardware task configuration information in the bitstream library, divides the FPGA into reconfigurable areas of different sizes, connects to the system bus. After the hardware tasks are scheduled, they are placed in the most suitable reconfigurable area, and the hardware tasks can be relocated between the reconfigurable areas. Morales-Villanueva and Gordon-Ross[54] did similar work on Virtex-5. They seized and restored hardware task modules on resource areas of different sizes, allowing designers to customize reconfigurable areas according to application task types. But they don't restore the state of the BRAM, which may result in data loss. Enemali *et al.*[55] implemented a low-cost resource-aware scheduling strategy on ZYNQ XC7Z100. This method uses the FAReP algorithm to calculate the FC fragmentation coefficient for each task module in the idle state on the FPGA. Suppose the FC value is greater than the set fragmentation threshold, and the time used to relocate the task module does not exceed the time when the module is used next time. In that case, the idle task module can be migrated without affecting the configuration or execution of other tasks. This method improves resource utilization and, at the same time, avoids the impact of multiple configurations of the same task on system performance.

3. Scheduling to reduce leakage power consumption

"Configuration prefetching", which is described in Section III.1.1), hides the hardware task configuration time by parallelizing the task's configuration phase with the execution phase of the current task, which brings about the problem of power leakage[56]. As shown in Fig.12, taking the schedule of Fig.4 as an example, T_2 and T_4 need to wait for the data-dependent T_1 and T_3 processing to be completed after they are configured on the FPGA before they can be executed, resulting in the separation of configuration and execution. To avoid the loss of task configuration information, the configured Static random access memory (SRAM) unit cannot be powered off, thus causing leakage power consumption. In order to reduce the leakage power consumption caused by configuration prefetching, the optimization goal of scheduling is to minimize the interval between the task configuration phase and the execution phase.

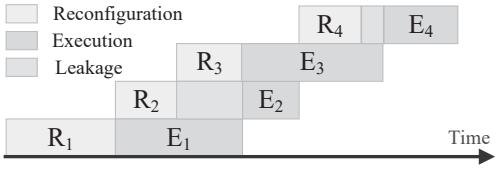


Fig. 12. Leakage power consumption

Yuh *et al.*[57] put forward the idea of mitigating FPGA leakage power consumption in scheduling. The author gave a two-stage scheduling method. The first stage aims at performance optimization and minimizes the DAG scheduling length to generate the initial schedule. In the second stage, a heuristic scheduler is designed to optimize the initial schedule. Under the premise of ensuring that the performance is not affected, the interval between task configuration and execution is shortened, and the leakage power consumption is minimized, which proves the feasibility of reducing power consumption by optimizing task scheduling. However, this method considers the leakage power consumption after the schedule is generated, so the optimization space is limited and may not significantly reduce the leakage power consumption. As shown in Fig.13(a)[57,59], taking Fig.12 as an example, this method can eliminate the leakage power between R_4 and E_4. Since the schedule has been generated, in order not to affect the total execution time of the application, the difference between R_2 and E_2 The interval will not be shortened. Similar work also Ref.[58], Li *et al.* proposed a post-placement leakage-aware scheduling algorithm. It reduces the leakage power consumption caused by configuration prefetching by optimizing the scheduling in the layout stage after the initial scheduling is completed. However, the algorithm has poor schedulability and cannot reduce the delay between task reconfiguration and execution in some cases. In response to this problem, ELAA[59] binds the configuration and execution phase of tasks before scheduling, and then assigns priority to each task according to the data dependency between tasks, and

generates an initial schedule. Finally, check the deadline of the task. If the task cannot be executed before its deadline, split the bound configuration and execution phases. As shown in Fig.13(b), assume that the end time of E_4 in Fig.12 is the deadline of T_4. ELAA can further reduce leakage power consumption by changing the configuration sequence of T_2 and T_3, and splitting the configuration and execution phase of T_2. However, ELAA cannot meet the needs of real-time applications by using static prioritization tasks.

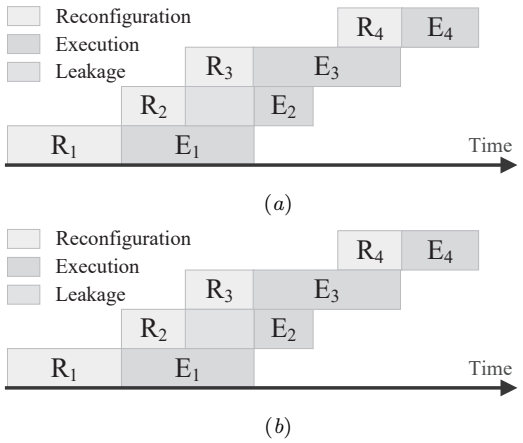


Fig. 13. Leakage power optimization[57,59]

In LRMA[60], a priority function is designed to reduce leakage power consumption, and a cost function is derived from managing the trade-off between performance optimization and leakage power optimization. In order to further improve the scheduling results, a heuristic algorithm is proposed. This method provides developers with design flexibility and can balance performance and power consumption. Experimental results show that compared with ELAA when the application scale gradually increases, the performance advantage of this algorithm becomes more and more obvious under the premise of ensuring the reduction of leakage power consumption. LBHS2D[61] has been modified on this basis to apply to the 2D FPGA resource model. Pham *et al.*[62] proposed a scheduling strategy using genetic algorithm based on LRMA, further solve the trade-off problem between performance and power consumption. This method can traverse the design space and obtain the Pareto optimum of performance or leakage power consumption, allowing developers to choose the best solution for different optimization goals. At the same time, in order to solve the problem of excessive calculation time, the author developed a machine learning component that uses linear regression and clustering algorithms. Compared with the scheduling strategy using a genetic algorithm, it can get the approximate Pareto optimum in a short time. However, this component sacrifices about 10% of the scheduling quality. TBLA[63] is a time-based leakage-aware algorithm. The method includes three stages: scheduling, pre-placement and optimization. TBLA obtains time-related task parameters, and tasks with long configuration time and short deadlines are scheduled first. Then, similar to ELAA, the task configuration and execution phase are considered as a whole for pre-placement. If the scheduling result cannot hide the task configuration time, split the configuration and execution phases, and sacrifice part of the leakage power to shorten the total execution time of the application. Experimental results show that the leakage power generated by TBLA is far less than the As soon as possible (ASAP) algorithm without considering leakage power, and the effect of reducing FPGA leakage power is better than LBHS2D.

In Refs.[64,65], the mathematical models for reducing leakage power consumption are presented respectively. Ghorbani[64] proposed energy yield and formulated the task scheduling under process variation as a MILP problem. In this work, the author considered leakage-power by defining some variables that depend on the calculation method of energy yield. However, the method produced too many variables that result in a MILP that requires high computational time to obtain the optimal solution. This is unacceptable in practical applications. Khodabandeloo *et al.*[65] proposed a novel formula considering leakage power for hardware task scheduling problems. It uses a uniform formula to capture parameter uncertainty. A given random parameter in the formula is used to solve the feasible region, thus optimizing the objective function. Compared with the method in Ref.[64], the number of formula parameters is significantly reduced. Experimental results show that this can improve performance by many times.

On the other hand, as the manufacturing process improves, integrating more components on an FPGA chip will cause the chip to heat up rapidly. The temperature of the chip has significant effects on leakage power. Therefore, it is very important to design a task scheduling method that can reduce the chip temperature and the energy consumption of the chip. Yang *et al.*[66] studied the leakage-aware energy efficiency scheduling when the leakage power depends on the temperature. To simplify the dependency between leakage and temperature, the authors designed a quadratic leakage model and proposed a pattern-based task scheduling method. The method achieves periodic task scheduling and reduces leakage power by periodically switching between active mode and dormant mode. However, this pattern-based scheduling method cannot guarantee the maximum temperature constraint. In response to this problem, Huang *et al.*[67] incorporated the interdependence of leakage, temperature, and supply voltage into the

analysis. They designed a novel method to estimate the overall energy consumption quickly. The authors proposed a scheduling method to minimize the overall energy consumption under the maximum temperature constraint. Compared with Ref.[66], the performance of power reduction is improved by 11.2%. Similarly, Saha *et al.*[68] proposed a genetic algorithm-based task scheduling method that satisfies thermal constraints to reduce energy consumption and simultaneously meet the maximum temperature and task deadlines. But this method does not consider the heterogeneity of real-time tasks. Zhou *et al.*[69] proposed a two-stage energy-efficient temperature-aware task scheduling method, which minimizes the dynamic energy consumption by a heuristic task scheduling approach in the first stage and minimizes the leakagepower by reducing the chip temperature through slack distribution. The simulation results show that the proposed algorithms consume up to 11.1% less energy than Ref.[68]. Ekhtiyari *et al.*[70] proposed a partitioned-based dynamic task scheduling method. The fuzzy-based technology proposed by the authors determines the priority of task assignment according to the utilization rate and temperature of chip, which can reduce the leakage power and the overall system energy consumption at the same time. Simulation results show that compared with Ref.[69], the proposed method reduces energy consumption by 9%.

IV. Comparison and Summary

FPGA-based hardware task scheduling is an important research topic in the field of high-performance computing. This paper analyzed the FPGA-based hardware task scheduling methods published in the literature between 2000 and 2020. Each method has advantages and disadvantages. In this section, we summarize the literature reviewed and its implementation methods. We compare these scheduling methods from different dimensions in detail and compare their optimization effects.

According to the different optimization goals, the existing research work is classified from three perspectives: time, resource, and leakage power. We split the reviewed literature on task scheduling into four different time periods: 2000–2005, 2006–2010, 2011–2015, and 2016–2020.

Table 1 (Refs.[15,16,18–22,25–42,45–49,51–55,57–70]) lists the research work done by researchers for different optimization goals in each time period. During the period 2000−2005, 9 papers were published. Due to the low integration of FPGA at that time, most papers focused on optimizing FPGA resource utilization. During the period 2006−2010, 10 papers were published. It can be noticed that some researchers have begun to optimize the FPGA leakage power consumption during scheduling. During the period 2011−2015, 16 papers were published. There are more and more researches focusing on optimizing time overhead and leakage power consumption. It can be attributed to the fact that FPGAs have begun to be widely used in the field of artificial intelligence and deployed on a large scale in data centers, so scholars are paying more and more attention to computing performance and cost. During the period 2016–2020, 14 papers were published. Artificial intelligence algorithms such as linear regression and clustering are gradually applied to the research of FPGA-based hardware task dynamic scheduling.

Table 1. Overall analysis of Survey 2000–2020

Optimization goal	2000–2005	2006–2010	2011–2015	2016–2020
Time overhead	Refs. [15, 16]	–	Refs. [18, 20, 22, 25, 27, 28]	Refs. [19, 21, 26]
Resource utilization	Refs. [29, 33, 37, 38, 40, 51, 52]	Refs. [30, 35, 39, 41, 45–47]	Refs. [31, 34, 53, 54]	Refs. [32, 36, 42, 48, 49, 55]
Leakage power	–	Refs. [57, 58, 66]	Refs. [59–61, 64, 67, 68]	Refs. [62, 63, 65, 69, 70]

Table 2 briefly summarizes the implementation methods of the above-mentioned FPGA-based hardware task scheduling research work. The scheduling strategies for optimizing time overhead mainly use configuration prefetching and module reuse technology to hide or avoid task reconfiguration. On the other hand, cluster tasks or place them close to communication channels to reduce communication overhead. The scheduling strategies for optimizing resource utilization are developed around FPGA free resource management and task management. The central idea of scheduling strategies to optimize leakage power consumption is to shorten the interval between the task configuration stage and execution stage as much as possible.

Table 3 analyzes the optimization effects of the investigated research work in four aspects: reconfiguration overhead, communication overhead, resource utilization, and leakage power consumption. "√" indicates that there are some positive optimization effects. "-" indicates that there is no optimization effect or optimization in this aspect is not considered. "×" means that it will not only have no optimization effect but will reduce the performance in this aspect. It can be concluded from Table 3 that different scheduling strategies are effective

Table 2. Summary of implementation methods

Optimization goal	Reference	Implementation methods
Time overhead	Refs.[15, 18, 32]	Heuristic algorithm
	Ref.[16]	Branch bound algorithm + ALAP
	Ref.[19]	Forefront-fetch
	Refs.[20, 21]	List-based scheduling
	Ref.[22]	Variations of DAG
	Refs.[25, 26]	LRU
	Ref.[27]	LPR
	Refs.[28, 31]	Linear programming
	Refs.[29, 30]	Additional communication constraints
	Refs.[33, 34]	Task proximity placement
	Refs.[35, 36]	Task grouping and clustering
Resource utilization	Refs.[37–39]	Based on KMER
	Refs.[40–42]	Based on MER
	Ref.[45]	Decentralized placement of tasks
	Refs.[46–48]	Based on the adjacency value between tasks
	Ref.[49]	Based on fragmentation coefficient
	Ref.[51]	Task reordering and relocation
	Ref.[52]	Dynamic task relocation
	Refs.[53–55]	Hardware implementation of task relocation
Leakage power	Refs.[57, 58, 68–70]	Heuristic algorithm
	Refs.[59, 63]	Configuration and execution of bundled tasks
	Refs.[60, 61]	Cost function to balance performance and leakage power
	Ref.[62]	Genetic algorithm + linear regression and clustering algorithm machine learning components
	Refs.[64, 65]	MILP
	Refs.[66, 67]	Mathematical modeling of leakage power and temperature

to some extent for different optimization goals and may have a positive impact on other aspects of performance. For example, FAReP considers the possibility of task reuse in the future when calculating the fragmentation coefficient FC and reserves hardware task modules with smaller FC on the FPGA, which improves resource utilization and reduces the number of task reconfigurations. 3DC links tasks with their related predecessors and successors and lays them out tightly to improve resource utilization while reducing communication overhead. However, most FPGA hardware task scheduling that only considers optimizing a single goal usually harms other performance aspects.

For example, to avoid resource fragmentation, Quad-Corner places task modules as close as possible to FPGA resource corners. Although resource utilization is improved, scattered placement will result in substantial inter-task communication overhead. The RTSM relocation task improves resource utilization while performing secondary configuration on the same task, resulting in a waste of reconfiguration and power consumption. In addition, restoring data after relocation also increases communication costs.

The survey found that most algorithms often fail to consider time overhead, resource utilization, and power consumption, and it is difficult to obtain optimal scheduling results. Therefore, it is necessary to continue further research in this field to achieve more efficient scheduling.

V. Application

With the rapid development of artificial intelligence, more and more FPGA-based Task Scheduling technology is used in neural network FPGA accelerators. Since Convolutional neural network (CNN) massively parallel processing is usually limited by the storage and bandwidth of FPGA, MEM-OPT[71] optimizes the storage utilization on a chip through computational task scheduling and data reuse, reducing the memory required for input feature graph by up to 80%. To improve the scheduling efficiency of Xilinx Deep learning processor unit (DPU), Zhu *et al.*[72] integrated and optimized it and proposed a high-performance task scheduling strategy. For CNNs of different sizes, the acceleration ratio increased by 22-30 times. Ref.[73] demonstrated that task partitioning in CNN's sublayers could improve overall energy efficiency, and proposed a CNN-based task partitioning and scheduling method, which reduces system power consumption while ensuring computing performance. Carreras *et al.*[74] applied a batch-based convolution scheduling method to the Temporal convolutional network (TCN) FPGA accelerator and referenced the software implementation, which increased the inference speed of TCN by 10 times and the energy efficiency by 3 times. From the perspective of ensuring the universality of FPGA accelerators, Jiang *et al.*[75] and Ma *et al.*[76] designed scheduling strategies that can apply to neural networks of different types and different precision.

Table 3. Comparison of optimization effects

Method	Reconfiguration	Communication	Resource-utilization	Leakage	Method	Reconfiguration	Communication	Resource-utilization	Leakage
Banerjee[15]	√	–	–	×	Marconi[45]	–	×	√	–
Resano[16]	√	–	–	×	Tabero[46]	–	√	√	–
Khuat[18]	√	–	√	×	Marconi[47]	–	√	√	–
Ramezani[19]	√	–	–	×	Xu[48]	–	√	√	–
Bauer[20]	√	–	–	×	Enemali[49]	√	–	√	–
Dai[21]	√	–	√	×	Diessel[51]	×	–	√	×
Cattaneo[22]	√	–	√	×	Gericota[52]	×	–	√	×
AlWattar[25]	√	–	–	×	George[53]	×	×	√	×
Hariharan[26]	√	–	–	×	MoralesVillanueva[54]	×	×	√	×
Mansub[27]	√	–	√	√	Enemali[55]	–	–	√	–
Deiana[28]	√	–	–	×	Yuh[57]	√	–	–	√
Fekete[29]	–	√	–	–	Li[58]	√	–	–	√
Marconi[30]	–	√	–	–	Hsieh[59]	√	–	–	√
Cadi[31]	√	√	–	–	Pham[60]	√	–	–	√
Abdallah[32]	√	√	–	–	Wang[61]	√	–	–	√
Ahmadinia[33]	–	√	–	–	Pham[62]	√	–	–	√
Sheng[34]	–	√	–	–	Zhou[63]	√	–	–	√
Huang[35]	–	√	√	×	Ghorbani[64]	–	–	–	√
Yoosefi[36]	–	√	–	–	Khodabandeloo[65]	√	–	–	√
Bazargan[37]	–	–	√	–	Yang[66]	–	–	–	√
Walder[38]	–	–	√	–	Huang[67]	–	–	–	√
Marconi[39]	–	–	√	–	Saha[68]	–	–	–	√
Handa[40]	–	–	√	–	Zhou[69]	–	–	–	√
Li[41]	–	–	√	–	Ekhtiyari[70]	–	–	–	√
Wang[42]	–	√	√	–	–	–	–	–	–

FPGA-based task scheduling techniques can be used to optimize the performance and power consumption of cloud data centers. Dai *et al.*[77] applied a revenue-based scheduling algorithm in the OpenStack-based cloud environment to efficiently use FPGA in the cloud. Compared with traditional methods, the FPGA accelerated cloud system can save 60.32% of computing resources and increase the speed by 1.386 times. Liu *et al.*[78,79] conducted energy optimization for cloud data centers under time constraints and Quality of service (QOS) constraints, respectively. Their software/hardware co-scheduling optimization strategy is applied to the heterogeneous hardware architecture supporting Dynamic voltage and frequency scaling (DVFS), which can significantly reduce the energy cost of the cloud data center. Kachris *et al.*[80] optimized the scheduling of FPGA-based MapReduce in the data center. Compared with general-purpose processors, the proposed method can reduce energy consumption by up to 2 orders of magnitude.

FPGA-based task scheduling techniques are also applicable to video codec. Li *et al.*[81] applied a fuzzy dynamic scheduling algorithm in their parallel H.264 encoder, which increased the speedup to 12.69 times. Osorio *et al.*[82] optimized the control scheduling of video entropy decoding and realized high-performance decoding that processes 1 bin per cycle.

In addition, the practical FPGA-based hardware task scheduling methods are also widely used in other research fields, such as edge computing[83], satellite[84], biology[85], sensors[86], *etc.*

VI. Challenge and Future Research

Future work can learn from the idea of mimic computing[87] and artificial intelligence methods, intelligently and dynamically perceive the state of heterogeneous computing resources and task characteristics, allocate appropriate resources to each task in advance and configure it on the FPGA, while optimizing time overhead, resource utilization, and power consumption. The following summarizes future work of FPGA-based hardware task scheduling in terms of hardware resources, task matching, versatility, memory access bottlenecks, and power consumption.

1) Hardware resources. In task scheduling, the system's resource status information will dynamically change with the task status. Modern FPGAs integrate heterogeneous resources with different functions. The industry's most advanced UltraScale+ FPGA architecture additionally supports dynamic reconfiguration of I/O, clock, and other resources. Therefore, how to parameterize FPGA's complex heterogeneous resources, real-time dynamic perception of resources, and realize efficient resource management needs further research.

2) Task matching. Task requirements and resource status dynamically change during processing. In addition,

the performance and efficiency of tasks on different hardware computing resources are different. As the scale of applications increases and FPGA integration becomes higher and higher, it will become more difficult to find suitable computing resources for the task. Therefore, artificial intelligence algorithms can be combined to build the best matching rules between tasks and heterogeneous resources and intelligently allocate the most appropriate computing resources to each task. Simultaneously, how to avoid resource competition, achieve dynamic adaptive matching of tasks and resources, and ensure the effectiveness of task scheduling is a difficult point worth studying.

3) Practicability. The reconfigurable time of FPGA is too long, which will seriously affect the computing performance in the practical application. Therefore, the FPGA design in practical applications is almost all customized, and only a few applications use the FPGA-based hardware task scheduling method. However, with the emergence of high-level FPGA development environment and the increase of reconfigurable speed in recent years, the application feasibility of the FPGA-based hardware task scheduling methods has gradually increased. Therefore, how to improve the feasibility of FPGA-based hardware task scheduling method by improving the speed of FPGA reconfiguration and optimizing the design flow will be a considerable challenge.

4) Memory access bottleneck. At present, the main bottleneck of FPGA acceleration design is bandwidth. Due to the low communication efficiency between hardware, the access speed cannot keep up with the calculation speed. Therefore, stack bandwidth or architectural innovations such as 3D storage and further optimization of the FPGA communication mechanism are also problems that need to be solved urgently.

5) Power consumption. Most of the existing algorithms do not consider power consumption or are limited to the leakage power generated by configuration prefetching. With the increasing scale of data center computing resources and the popularity of embedded applications, the power consumption problem has become increasingly prominent. Therefore, exploring new physical devices such as light diffraction, memristor, *etc.*, to achieve "integration of storage and computing" may effectively solve this problem.

VII. Conclusions

In recent years, with the widespread deployment of FPGAs in various fields such as artificial intelligence, data centers, and cloud computing, the dynamic scheduling of FPGA-based hardware tasks directly affects the overall performance of the computing system and is of great significance in practical applications. This paper has made a comprehensive survey of some representative FPGA-based hardware task scheduling methods in the past two decades. Various methods are analyzed from the perspectives of time, resources, and power consumption, and the implementation methods of task scheduling are classified and summarized. Then some practical applications are introduced. On this basis, it highlights the characteristics and limitations of various research work and finally summarizes the challenges faced by the current dynamic scheduling of hardware tasks and feasible solutions. This paper makes a reasonable judgment on the future research of FPGA-based hardware task dynamic scheduling, which has specific reference significance for the further development of this field.

References

[1] I.L. Markov, "Limits on fundamental limits to computation", *Nature*, Vol.512, No.7513, pp.147–154, 2014.

[2] A. Putnam, A.M. Caulfield, E.S. Chung, *et al.* "A reconfigurable fabric for accelerating large-scale datacenter services", *Proc. of International Symposium on Computer Architecture*, USA, pp.13–24, 2014.

[3] X. Liu, W. Liu, H. Ma, *et al.,* "Large-scale vehicle re-identification in urban surveillance videos", *Proc. of IEEE International Conference on Multimedia and Expo*, City, State, USA, pp.1–6, 2016.

[4] N. Tarafdar, T. Lin, E. Fukuda, *et al.,* "Enabling flexible network FPGA clusters in a heterogeneous cloud data center", *Proc. of ACM/SIGDA International Symposium*, City, State, USA, pp.237–246, 2017.

[5] D. Koch, "Partial reconfiguration on FPGAs: Architectures, tools and applications", *Springer Science & Business Media*, 2012.

[6] S. Saha, A. Sarkar and A. Chakrabarti, "Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms", *IEEE Embedded Systems Letters*, Vol.7, No.1, pp.23–26, 2015.

[7] R. G. Michael and S. J. David, "Computers and intractability: A guide to the theory of np-completeness", W. H. Freeman and Company, San Fr, pp.90–91, 1979.

[8] H. Murata, K. Fujiyoshi, S. Nakatake, *et al.,* "Rectangle-packing-based module placement", *Proc. of IEEE/ACM International Conference on Computer-aided Design*, Springer, Boston, MA, pp.535–548, 2003.

[9] M. M. Bassiri and H. S. Shahhoseini, "On-line HW/SW partitioning and co-scheduling in reconfigurable computing systems", *Proc. of IEEE International Conference on Computer Science & Information Technology*, pp.557–562, 2009.

[10] Xilinx, "Vivado design suite user guide: Partial reconfiguration", *https://china.xilinx.com/support documentation/sw_manuals/xilinx2019_2/ug909-vivado-partial-reconfiguration.pdf*, 2019.

[11] P. Lysaght, B. Blodget, J. Mason, *et al.,* "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs", *Proc. of International Conference on Field Programmable Logic and Applications*, Madrid, Spain, pp.1–6, 2006.

[12] R.P. Dick, D.L. Rhodes and W. Wolf, "TGFF: Task graphs for free", *Proc. of International Workshop on Hardware/software*

Codesign, Seattle, WA, USA, pp.97–101, 1998.

[13] J. Angermeier and J. Teich, "Heuristics for scheduling reconfigurable devices with consideration of reconfiguration overheads", *Proc. of IEEE International Symposium on Parallel & Distributed Processing*, Miami, FL, USA, pp.1–8, 2008.

[14] Hauck and Scott, "Configuration prefetch for single context reconfigurable coprocessors", *International Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, pp.65–74, 1998.

[15] S. Banerjee, E. Bozorgzadeh and N. Dutt, "Considering run-time reconfiguration overhead in task graph transformations for dynamically reconfigurable architectures", *Proc. of IEEE Symposium on Field-programmable Custom Computing Machines*, Napa, CA, USA, pp.273–274, 2005.

[16] J. Resano, D. Mozos and F. Catthoor, "A hybrid design-time/run-time scheduling flow to minimise the reconfiguration overhead of FPGAs", *Microprocessors and Microsystems*, Vol.28, No.5, pp.291–301, 2004.

[17] J. Resano, D. Verkest, D. Mozos, *et al.,* "A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware", *Proc. of Design, Automation and Test in Europe*, Munich, Germany, pp.106–111, 2005.

[18] Q.H. Khuat, D. Chillet and M. Hubner, "Considering reconfiguration overhead in scheduling of dependent tasks on 2D Reconfigurable FPGA", *Proc. of NASA/ESA Conference on Adaptive Hardware & Systems*, Leicester, UK, pp.1–8, 2014.

[19] R. Ramezani, "A prefetch-aware scheduling for FPGA-based multi-task graph systems", *The Journal of Supercomputing*, Vol.76, pp.7140–7160, 2020.

[20] L. Bauer, A. Grudnitsky, M. Shafique, *et al.,* "PATS: A performance aware task scheduler for runtime reconfigurable processors", *Proc. of IEEE International Symposium on Field-Programmable Custom Computing Machines*, Toronto, ON, Canada, pp.208–215, 2012.

[21] Z. Dai and T. Qu., "Task scheduling technology for coarse-grained dynamic reconfigurable system based on configuration prefetching and reuse", *Journal of Electronics & Information Technology*, Vol.41, No.6, pp.1458–1465, 2019. (in Chinese)

[22] R. Cattaneo, R. Bellini, G. Durelli, *et al.,* "PaRA-sched: A reconfiguration-aware scheduler for reconfigurable architectures", *Proc. of IEEE International Parallel & Distributed Processing Symposium Workshops*, Phoenix, AZ, USA, pp.243–250, 2014.

[23] A.A. Nacci, C. Pilato, R. Cattaneo, *et al.*, "A2B: An integrated framework for designing heterogeneous and reconfigurable systems", *Proc. of NASA/ESA Conference on Adaptive Hardware & Systems*, Turin, Italy, pp.198–205, 2013.

[24] Y. Lu, T. Marconi, K. Bertels, *et al.*, "Online task scheduling for the FPGA-based partially reconfigurable systems", *Proc. of International Workshop on Applied Reconfigurable Computing*, Vol.5453, pp.216–230, 2009.

[25] A. Al-Wattar, S. Areibi and F. Saffih, "Efficient on-line hardware/software task scheduling for dynamic run-time reconfigurable systems", *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, Shanghai, China, pp.401–406, 2012.

[26] I. Hariharan and M. Kannan, "Algorithms for reducing reconfiguration overheads using prefetch, reuse, and optimal mapping of tasks", *Concurrency & Computation Practice & Experience*, Vol.33, No.7, 2018.

[27] M. Mansub, B. Hadi and S. Shahhoseini, "Configuration reusing in on-line task scheduling for reconfigurable computing systems", *Journal of Computer Science & Technology*, Vol.26, No.3, pp.463–473, 2011.

[28] E.A. Deiana, M. Rabozzi, R. Cattaneo, *et al.,* "A multiobjective reconfiguration-aware scheduler for FPGA-based heterogeneous architectures", *Proc. of International Conference on ReConFigurable Computing and FPGAs*, Riviera Maya, Mexico, pp.1–6, 2015.

[29] S. Fekete, E. Kohler and J. Teich, "Optimal FPGA module placement with temporal precedence constraints", *Proc. of Design, Automation and Test in Europe*, Munich, Germany, pp.658–665, 2001.

[30] Y. Lu, T. Marconi, K. Bertels, *et al.,* "A communication aware online task sheduling algorithm for FPGA-based partially reconfigurable systems", *Proc. of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, Charlotte, NC, USA, pp.65–68, 2010.

[31] A.E. Cadi, O. Souissi, R.B. Atitallah, *et al.,* "Mathematical programming models for scheduling in a CPU/FPGA architecture with heterogeneous communication delays", *Journal of Intelligent Manufacturing*, Vol.29, No.3, pp.629–640, 2018.

[32] F. Abdallah, C. Tanougast, I. Kacem, *et al.,* " A comparison of two metaheuristic algorithms for scheduling problem on a heterogeneous CPU/FPGA architecture with communication delays", *Proc. of International Conference on Control*, Barcelona, pp.0294–0299, 2017.

[33] A.Ahmadinia, C.Bobda, D.Koch, *et al.,* "Task scheduling for heterogeneous reconfigurable computers", *Proc. of Symposium on Integrated Circuits and System Design*, Pernambuco, Brazil, pp.22–27, 2004.

[34] Y. Sheng, Y. Liu, R. Li, *et al.,* "A communication-aware scheduling algorithm for hardware task scheduling model on FPGA-based reconfigurable systems", *Journal of Computers*, Vol.9, No.11, pp.65–68, 2014.

[35] M. Huang, V.K. Narayana, H. Simmler, *et al.,* "Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing", *ACM Transactions on Reconfigurable Technology and Systems*, Vol.3, No.4, pp.1–25, 2010.

[36] A. Yoosefi and H.R. Naji, "A clustering algorithm for communication-aware scheduling of task graphs on multi-core reconfigurable systems", *IEEE Transactions on Parallel & Distributed Systems*, Vol.28, No.10, pp.2718–2732, 2017.

[37] K. Bazargan and R. Kaster, "Fast template placement for reconfigurable computing systems", *IEEE Design & Test of Computers*, Vol.17, No.1, pp.68–83, 2017.

[38] H. Walder, C. Steiger and M. Platzner, "Fast online task placement on FPGAs: Free space partitioning and 2D-hashing", *Proc. of International Parallel and Distributed Processing Symposium*, Nice, France, 8 pages, 2003.

[39] T. Marconi, Y. Lu, K. Bertels, *et al.,* "Intelligent merging online task placement algorithm for partial reconfigurable systems", *Proc. of Design, Automation and Test in Europe*, Munich, Germany, pp.1346–1351, 2008.

[40] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement", *Proc. of Annual Design Automation Conference*, San Diego, CA, USA, pp.960–965, 2004.

[41] T. Li and Y. Yang, "Algorithms of reconfigurable resource management and hardware task placement", *Journal of Computer Research and Development*, Vol.2, No.22, pp.171–178, 2008. (in Chinese)

[42] G. Wang, S. Liu, J. Nie, *et al.,* "An online task placement algorithm based on maximum empty rectangles in

dynamic partial reconfigurable systems", *Proc. of NASA/ESA Conference on Adaptive Hardware and Systems*, Pasadena, CA, USA, pp.180–185, 2017.

[43] A. Ahmadinia, C. Bobda, M. Bednara, *et al.,* "A new approach for on-line placement on reconfigurable devices", *International Parallel and Distributed Processing Symposium*, Santa Fe, NM, USA, 7 pages, 2004.

[44] R. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems", *ACM Comput. Surv.*, Vol.43, No.4, 44 pages, 2011.

[45] T. Marconi, Y. Lu, K. Bertels, *et al.*, "A novel fast online placement algorithm on 2D partially reconfigurable devices", *International Conference on Field-Programmable Technology*, Sydney, NSW, Australia, pp.296–299, 2009.

[46] J. Tabero, J. Septien, H. Mecha, *et al.,* "Task placement heuristics based on 3D-adjacency and look-ahead in reconfigurable systems", *Asia and South Pacific Conference on Design Automation*, Yokohama, Japan, 6 pages, 2006.

[47] T. Marconi, Y. Lu, K. Bertels, *et al.,* "3D Compaction: A novel blocking-aware algorithm for online hardware task scheduling and placement on 2D partially reconfigurable devices", *International Symposium on Applied Reconfigurable Computing*, Bangkok, Thailand, pp.194–206, 2010.

[48] J. Xu, L. Liu, W. Li, *et al.*, "Reconfigurable hardware task scheduling algorithm based on 3D fragmentation layout strategy", *Journal of Electronics & Information Technology*, Vol.40, No.8, pp.247–254, 2018. (in Chinese)

[49] G. Enemali, A. Adetomi and T. Arslan, "FAReP: Fragmentation-aware replacement policy for task reuse on reconfigurable FPGAs", *IEEE International Parallel and Distributed Processing Symposium Workshops*, Lake Buena Vista, FL, USA, pp.202–206, 2017.

[50] K. Compton, Z. Li, J. Cooley, *et al.,* "Configuration relocation and defragmentation for run-time reconfigurable computing", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.10, No.3, pp.209–220, 2002.

[51] O. Diessel, H. ElGindy, M. Middendorf, *et al.,* "Dynamic scheduling of tasks on partially reconfigurable FPGAs", *IEE Proceedings - Computers and Digital Techniques*, Vol.147, No.3, pp.181–188, 2000.

[52] M.G. Gericota, G.R. Alves, M.L. Silva, *et al.,* "On-line defragmentation for run-time partially reconfigurable FPGAs", *International Conference on Field Programmable Logic and Applications*, Montpellier, France, pp.302–311, 1998.

[53] G. Charitopoulos, I. Koidis, K. Papadimitriou, *et al.,* "Hardware task scheduling for partially reconfigurable FPGAs", *International Symposium on Applied Reconfigurable Computing*, Bochum, Germany, pp.487–498, 2015.

[54] A. Morales-Villanueva and A. Gordon-Ross, "HTR: On-chip hardware task relocation for partially reconfigurable FPGAs", *International Symposium on Applied Reconfigurable Computing*, Los Angeles, California, USA, pp.185–196, 2013.

[55] G. Enemali, A. Adetomi and T. Arslan, "A placement management circuit for efficient realtime hardware reuse on FPGAs targeting reliable autonomous systems", *International Symposium on Circuits and Systems* , Baltimore, MD, pp.1–4, 2017.

[56] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA", *Custom Integrated Circuits Conference*, San Jose, CA, USA, pp.57–60, 2003.

[57] P.H. Yuh, C. Yang, C. Li, *et al.,* "Leakage-aware task scheduling for partially dynamically reconfigurable FPGAs", *ACM Trans. Des. Autom. Electron. Syst.*, Vol.14, No.4, 26 pages, 2009.

[58] C. Li, P. Yuh, C. Yang, *et al.,* "Post-placement leakage optimization for partially dynamically reconfigurable FPGAs", *International Symposium on Low Power Electronics and Design*, Portland, OR, USA, pp.92–97, 2007.

[59] J. Hsieh, Y. Chang and W. Lee, "An enhanced leakage-aware scheduler for dynamically reconfigurable FPGAs", *Asia and South Pacific Design Automation Conference*, Yokohama, Japan, pp.661–667, 2011.

[60] N.K. Pham, A.K. Singh and A. Kumar, "A multi-stage leakage aware resource management technique for reconfigurable architectures", *Edition of the Great Lakes Symposium on VLSI*, Houston, Texas, USA, pp.63–68, 2014.

[61] S. Wang, N.K. Pham, A.K. Singh, *et al.,* "Leakage and performance aware resource management for 2D dynamically reconfigurable FPGA architectures", *International Conference on Field Programmable Logic and Applications*, Munich, Germany, pp.1–4, 2014.

[62] P.N. Khanh, A. Kumar, A.K. Singh, *et al.,* "Leakage aware resource management approach with machine learning optimization framework for partially reconfigurable architectures", *Template*, Vol.47, pp.231–243, 2016.

[63] T. Zhou, T. Pan, Z. Bao, *et al.,* "A time-based leakage-aware algorithm for task placement and scheduling problem on dynamic reconfigurable FPGA", *International Conference on Systems and Informatics*, Nanjing, China, pp.501–506, 2018.

[64] M. Ghorbani, "A variation and energy aware ILP formulation for task scheduling in MPSoC", *International Symposium on Quality Electronic Design*, Santa Clara, CA, USA, pp.772–777, 2012.

[65] B. Khodabandeloo, A. Khonsari, A. Majidi, *et al.,* "Task assignment and scheduling in MPSoC under process variation: A stochastic approach", *Asia and South Pacific Design Automation Conference*, Jeju, Korea (South), pp.690–695, 2018.

[66] C. Yang, J. Chen, L. Thiele, *et al.,* "Energy-efficient real-time task scheduling with temperature-dependent leakage", *Design, Automation & Test in Europe Conference & Exhibition*, Dresden, Germany, pp.9–14, 2010.

[67] H. Huang and G. Quan, "Leakage aware energy minimization for real-time systems under the maximum temperature constraint", *Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, France, pp.1–6, 2011.

[68] S. Saha, Y. Lu and J. S. Deogun, "Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems", *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Seoul, Korea (South), pp.41–50, 2012.

[69] J. Zhou, T. Wei, M. Chen, *et al.,* "Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.35, No.8, pp.1269–1282, 2016.

[70] Z. Ekhtiyari, V. Moghaddas and H. Beitollahi, "A temperature-aware and energy-efficient fuzzy technique to schedule tasks in heterogeneous MPSoC systems", *The Journal of Supercomputing*, Vol.75, 22 pages, 2019.

[71] G. Dinelli, G. Meoni, E. Rapuano, *et al.,* "MEM-OPT: A scheduling and data re-use system to optimize on-chip memory usage for CNNs on-board FPGAs", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol.10, No.3, pp.335–347, 2020.

[72] J. Zhu, L. Wang, H. Liu, *et al.,* "An efficient task assignment framework to accelerate DPU-based convolutional neural

network inference on FPGAs", *IEEE Access*, Vol.8, 14 pages, 2020.

[73] E. Oh, W. Han, E. Yang, *et al.,* "Energy-efficient task partitioning for CNN-based object detection in heterogeneous computing environment", *International Conference on Information and Communication Technology Convergence*, Jeju, Korea (South), pp.31–36, 2018.

[74] M. Carreras, G. Deriu, L. Raffo, *et al.,* "Optimizing temporal convolutional network inference on FPGA-based accelerators", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol.10, No.3, pp.348–361, 2020.

[75] J. Wei, Z. Song, J. Zhan, *et al.,* "Optimized co-scheduling of mixed-precision neural network accelerator for real-time multitasking applications", *Journal of Systems Architecture*, Vol.110, 12 pages, 2020.

[76] Y. Ma, Y. Cao, S. Vrudhula, *et al.,* "Automatic compilation of diverse CNNs onto high-performance FPGA accelerators", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.39, No.2, pp.424–437, 2020.

[77] G. Dai, Y. Shan, F. Chen, *et al.,* "Online scheduling for FPGA computation in the Cloud", *International Conference on Field-Programmable Technology*, Shanghai, China, pp.330–333, 2014.

[78] X. Liu, P. Liu and L. Hu, "Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters", *Concurrency and Computation: Practice and Experience*, Vol.32, No.18, 17 pages, 2020.

[79] X. Liu, P. Liu, H. Li, *et al.,* "Energy-aware task scheduling strategies with QoS constraint for green computing in cloud data centers", *Proc. of the 2018 Conference on Research in Adaptive and Convergent Systems*, Honolulu, Hawaii, USA, pp.260–267, 2018.

[80] C. Kachris, D. Diamantopoulos, G.C. Sirakoulis, *et al.,* "An FPGA-based integrated mapReduce accelerator platform", *Journal of Signal Processing Systems*, Vol.87, pp.357–369, 2017.

[81] D. Li, Y.Hou, Z.Huang, *et al.,* "A framework of multi-characteristics fuzzy dynamic scheduling for parallel video processing on MPSoC architecture", *IEEE International Conference on Fuzzy Systems*, Taipei, China, pp.627–634, 2011.

[82] R. Osorio and J. Bruguera, "High-speed FPGA architecture for CABAC decoding acceleration in H.264/AVC standard", *Journal of Signal Processing Systems*, Vol.72, pp.119–132, 2013.

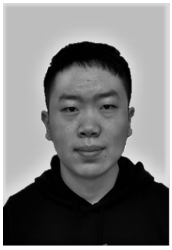
[83] Z. Zhu, J. Zhang and J. Zhao, "A hardware and software task-scheduling framework based on CPU+FPGA heterogeneous architecture in edge computing", *IEEE Access*, Vol.7, 14 pages, 2019.

[84] Y. Liu, Y. Shen and Z. Sun, "Task scheduling algorithm of FPGA for on-board reconfigurable coprocessor", *International Conference on Computer Science and Artificial Intelligence*, Beijing, China, pp.207–214, 2019.

[85] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis", *IEEE Transactions on Parallel and Distributed Systems*, Vol.21, No.9, pp.1267–1280, 2010.

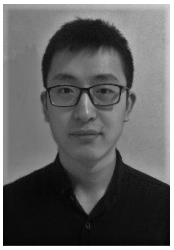
[86] Y. Li, Z. Jia, S. Xie, *et al.,* "Dynamically reconfigurable hardware with a novel scheduling strategy in energy-harvesting sensor networks", *IEEE Sensors Journal*, Vol.13, No.5, pp.2032–2038, 2013.

[87] J. Wu, "Meaning and vision of mimic computing and mimic security defense", *Telecommunications Science*, Vol.30, No.7, pp.2–7, 2014.



LI Tianyang was born in 1996. He received the B.E. degree from Harbin Institute of Technology in 2018. He completed the M.S. degree from PLA Strategic Support Force Information Engineering University in 2020 and is currently pursuing the Ph.D. degree in information and communication engineering. His research interests include high performance computing and task scheduling. (Email: tyli0107@163.com)

ZHANG Fan (corresponding author) was born in 1981. He received the Ph.D. degree in information and communication engineering in 2013 from Information Engineering University. He is currently an Associate Researcher. His research interests include proactive defense, high-performance computing, and big data processing. (Email: 17034203@qq.com)



GUO Wei was born in 1990. He received the Ph.D. degree in information and communication engineering in 2013 from PLA Strategic Support Force Information Engineering University. He is currently an Assistant Researcher. His research interests include information security, distributed storage system, and big data processing.

SUN Mingqian was born in 1995. He received B.E. degree from Wuhan University of Technology in 2017, and M.S. degree from Army Engineering University of PLA in 2020. He is currently pursuing the Ph.D. degree in Southeast University. His research areas include network security, big data processing and deep packet inspection.

CHEN Li was born in 1997. He received the B.E. degrees in network engineering in 2019 from PLA Strategic Support Force Information Engineering University. His research interests include computer vision, small scale object detection and big data processing.