An Efficient Task Assignment Framework to Accelerate DPU-Based Convolutional Neural Network Inference on FPGAs

Jiang Zhu1,2, Lizan Wang1,2, Haolin Liu2, Shujuan Tian2,
Qingyong Deng2,3, (Member, IEEE), and Jianqi Li1,4

1School of Automation and Electronics Information, Xiangtan University, Xiangtan 411105, China
2Key Laboratory of Hunan Province for Internet of Things and Information Security, Xiangtan University, Xiangtan 411105, China
3Science and Technology on Parallel and Distributed Processing Laboratory, China
4Hunan Province Cooperative Innovation Center for the Construction and Development of Dongting Lake Ecological Economic Zone, Hunan University of Arts and Science, Changde 415000, China

Corresponding author: Jianqi Li (jianqi_li@126.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003702, in part by the National Natural Science Foundation of China under Grant 61902336 and Grant 61703157, in part by the Hunan Province Science and Technology Project Funds under Grant 2018TP1036, in part by the Natural Science Foundation of Hunan Province, China, under Grant 2019JJ50592, and in part by the CERNET Innovation Project under Grant NGII20160310.

ABSTRACT Field Programmable Gate Array (FPGA) has become an efficient accelerator for convolutional neural network (CNN) inference due to its high performance and flexibility. To further improve the performance of CNN inference on FPGAs, an Intellectual Property core (IP core) called Deep Learning Processor Unit (DPU) is released by Xilinx. Unlike previous FPGA-based hardware designs focusing on specific functions or CNNs, the DPU IP supports ample basic functions of deep learning, and the developers can take advantage of DPUs to accelerate CNN inference conveniently. In DPU-based CNN acceleration platform, an encapsulated scheduler plays a crucial role in task scheduling between heterogeneous ARM and multiple DPUs. However, the current scheduler is unsatisfactory because its low schedule efficiency. This paper thus presents a high performance task assignment framework built upon Xilinx hybrid CPU-FPGA MPSoC devices. We first evaluate the main causes of low schedule efficiency problem. Then, we explore the scheduler rules and improve schedule efficiency through purposeful observations and analysis. Finally, we integrate our optimizations, and propose an efficient task assignment framework to maximize performance on DPU-based CNN acceleration platform. Experimental results on Xilinx Zynq UltraScale+ MPSoC zcu104 show that our efficient task assignment framework significantly boosts schedule efficiency for small-scale CNNs (from 36% to 70%), medium-scale CNNs (from 65% to 95%), and large-scale CNNs (from 77% to 99%) compared with original schedule strategy.

INDEX TERMS Field programmable gate array (FPGA), deep learning processor unit (DPU), convolutional neural network (CNN) accelerator, schedule efficiency.

I. INTRODUCTION

A. BACKGROUND

Convolutional neural network (CNN) has gradually replaced traditional machine vision methods in image recognition, object detection, image segmentation and many other machine vision applications because of its excellent performance [1]–[6]. Generally, each CNN consists of multiple layers with different functions, which requires suitable hardware to accelerate its inference process. Meanwhile, many emerging fields, such as intelligent robots, unmanned aerial vehicles, autopilot cars and space probes, have imposed strict restrictions on power, delay and physical size of hardware accelerators, and traditional GPUs are hard to satisfy their requirements [7], [8]. To satisfy the above strict requirements, Field Programmable Gate Array (FPGA) has become a high performance and flexibility accelerator of CNN inference in many emerging fields [9]–[13].
In actual applications, the developers tend to run multiple CNNs in parallel to complete different tasks because these CNNs have different network structures, model sizes and their own advantages. In response to increasingly complex application scenarios, the Intellectual Property core (IP core) called Deep Learning Processor Unit (DPU) is released by Xilinx Inc. Different from previous FPGA-based hardware accelerators focusing on specific functions or CNNs [2], [5], [6], [13]–[16], the DPU IP supports ample basic functions of deep learning, which allows for the efficient implementation of many CNNs on FPGAs.

To strike a balance between flexibility (of being able to handle different functional layers of CNNs) and performance, a more subtle acceleration platform should be designed as heterogeneous architecture, which can take advantage of both CPU and FPGA, like Xilinx Zynq SoC, a so-called hybrid CPU-FPGA device. This type of hybrid devices has been employed in accelerating a wide range of application-specific algorithms, including CNN inference. In our study, the DPU IP is implemented in the programmable logic (FPGA) of the selected Xilinx Zynq UltraScale+ MPSoC zcu104 and integrated into the processing system (ARM) through an AXI interconnect to perform CNN inference, such as image classification, object detection, and semantic segmentation.

**TABLE 1.** CNN features and parameters supported by the DPU-based acceleration platform (channel\_parallel = 16).

| Features                | Parameters |
|-------------------------|------------|
| **Convolution**         |            |
| Kernel Sizes            | W1–16 H1–6 |
| Strides                 | W1–H1–4   |
| Padding\_w              | 1 kernel\_w–1 |
| Padding\_h              | 1 kernel\_h–1 |
| Input Size              | Arbitrary  |
| Input Channel           | 1–256 channel\_parallel |
| Output Channel          | 1–256 channel\_parallel |
| Activation              | ReLU or ReLU\_6 |
| **Depthwise Convolution** |          |
| Kernel Sizes            | W1–16 H1–4 |
| Strides                 | W1–H1–4   |
| Padding\_w              | 1 kernel\_w–1 |
| Padding\_h              | 1 kernel\_h–1 |
| Input Size              | Arbitrary  |
| Input Channel           | 1–256 channel\_parallel |
| Output Channel          | 1–256 channel\_parallel |
| Activation              | ReLU or LeakyReLU |
| **Deconvolution**       |            |
| Kernel Sizes            | W1–H1–4   |
| Stride\_w               | stride\_w * output\_channel ≤ 256 * channel\_parallel |
| Stride\_h               | Arbitrary  |
| Padding\_w              | 1 kernel\_w–1 |
| Padding\_h              | 1 kernel\_h–1 |
| Input Size              | Arbitrary  |
| Input Channel           | 1–256 channel\_parallel |
| Output Channel          | 1–256 channel\_parallel |
| Activation              | ReLU or LeakyReLU |
| **Max Pooling**         |            |
| Kernel Sizes            | W1–H1–4   |
| Strides                 | W1–H1–4   |
| Padding\_w              | 1 kernel\_w–1 |
| Padding\_h              | 1 kernel\_h–1 |
| Input size              | Arbitrary  |
| **Element-Wise-sum**    |            |
| Input channel           | 1–256 channel\_parallel |
| Output channel          | 1–256 channel\_parallel |
| Activation              | Arbitrary  |
| **Concat**              |            |
| Input channel           | 1–256 channel\_parallel |
| Output channel          | 1–256 channel\_parallel |
| Activation              | Arbitrary  |
| **Reshape**             |            |
| Input channel           | 1–256 channel\_parallel |
| Output channel          | 1–256 channel\_parallel |
| Activation              | Arbitrary  |

**FIGURE 1.** The DPU-based CNN acceleration platform.

Fig. 1 shows the DPU-based CNN acceleration platform, where the DPU IP is instantiated as DPU0 and DPU1 in zcu104. The process of CNN inference in this platform is as follows. Firstly, the quantizer (Xilinx DECENT tool) is used to optimize the system performance by reducing the model size, and the compiler (Xilinx DNNC tool) is designed to implement CNN inference and memory access by generating the DPU-related instructions. Then, the CNN inference tasks are initialized in ARM and assigned to multiple DPUs by using an encapsulated scheduler. Finally, the results are obtained. Table 1 shows CNN features and parameters supported by the DPU-based acceleration platform, the parameter channel\_parallel = 16 is determined by the DPU configuration. In addition, the functions, such as average pooling, ReLU and softmax, are optional and determined during DPU configuration.

Therefore, it is more convenient to accelerate CNN inference by using DPU-based CNN acceleration platform, especially for developers without FPGA knowledge, because it integrates many excellent hardware functions on FPGAs compared with the FPGA hardware accelerators for specific functions or CNNs proposed in the previous studies. To further demonstrate the advantages of the DPU-based acceleration platform, we introduce a representative CNN accelerator (NullHop [17]) released recently as a comparison in Table 2.

**B. MOTIVATION**

The DPU-based CNN acceleration platform affords ample basic functions of deep learning, supports the fast development flow and provides convenient operation to accelerate different CNN inferences in parallel for developers. In this platform, an encapsulated scheduler assigns tasks to multiple DPUs, and the concept of schedule efficiency is proposed to reflect the utilization of DPUs over a period of time, which can be detected by using Xilinx performance monitoring tool Dsight [18], [19]. In fact, the differences of structures and models of CNNs result in different inference times, and a flexible schedule strategy is required to deal with the complex...
and changeable situations. However, the low schedule efficiency problem is exposed as the increasing complexity of application scenarios, which means that the inference tasks take longer to complete their execution. The core ideas of the original schedule strategy (S_{original}) are as follows: (1) there is a priority between DPU{s}, that is, inference tasks are assigned to DPU0 if DPU0 and DPU1 are idle (tasks are assigned to DPU1 only when DPU0 is busy); (2) inference tasks simply run on DPU{s} by using up to concurrent two threads in the order of initialization completion, rather than more efficient method [18], [19]. Fig. 2 to Fig. 4 reveal how the S_{original} leads to low schedule efficiency. Note that an inference in this paper represents the whole calculation process of a feature map (single-frame) from the first layer to the last layer of an CNN.

Fig. 2 shows the unbalanced task assignment problem by using one thread and running single network. In Fig. 2, the network A consists of two parts, where initialization part runs on ARM and inference part runs on DPU, and the number of inferences is 3. Obviously, the S_{original} cannot evenly assign those tasks to multiple DPU{s} according to the actual number of inferences, which results in DPU1 being idle during above process. An ideal schedule strategy (our optimization goal) should assign those tasks to different DPU{s}, thereby improving schedule efficiency.

Fig. 3 exposes the unused interval time between two inferences problem by using two threads and running single network. In Fig. 3, although DPU1 is utilized to inferences, the schedule efficiency is still low because the interval time between two inferences is wasted. An ideal schedule strategy (our optimization goal) should take advantage of the interval time on each DPU.

Fig. 4 reveals the scheduling confusion problem by using two threads and running multiple networks named A and B. In Fig. 4, although DPU1 is utilized to inferences, the schedule efficiency is still low because the interval time between two inferences is wasted. An ideal schedule strategy (our optimization goal) should take advantage of the interval time on each DPU.

Table 2. Comparison between the DPU-based acceleration platform and NullHop CNN accelerator.

|                        | DPU          | NullHop [17] |
|------------------------|--------------|--------------|
| Parallelism            | Network-level| MAC operation-level |
| Simplicity             | Support for APIs | -           |
| Functionality          | See Table 1  | Convolution (ReLU, Mat-mul) |
| Resources              | BRAM: (69-249)*1-(3) | BRAM: 386 |
|                        | DSP: (66-64)*1-(3) | DSP: 128 |
| Compatibility (Configurability) | Support for low-end and high-end devices | -      |
optimization goal) should ensure that the inference tasks of different networks are controllable.

In summary, the $S_{\text{original}}$ has two problems: unbalanced task assignment and unused interval time, which lead to low schedule efficiency.

C. CONTRIBUTIONS

In this study, we aim to improve schedule efficiency on DPU-based CNN acceleration platform. Although the principles of the encapsulated scheduler cannot be modified directly, the execution sequence and the trigger time of each task are controllable by exploring the rules of the scheduler. Therefore, schedule efficiency has the potential to be improved. Our contributions are as follows:

1. We evaluate the main causes of low schedule efficiency problem, and explore the rules of the encapsulated scheduler.
2. We improve schedule efficiency through observations and analysis, and propose an efficient task assignment framework.

To the best of our knowledge, this is the first work that supports an efficient task assignment framework on DPU-based CNN acceleration platform, which can take advantage of both CPU’s generality and FPGA’s flexibility. Furthermore, the proposed efficient task assignment framework also presents a remarkable improvement in terms of schedule efficiency in comparison with $S_{\text{original}}$.

The rest of this paper is organized as follows. Section II shows the related work. Section III evaluates the main causes of low schedule efficiency problem, and explores the rules of the encapsulated scheduler. Section IV proposes an efficient task assignment framework to improve schedule efficiency. Section V verifies the performance (schedule efficiency) of our efficient framework and presents the experimental results. Section VI concludes this study.

II. RELATED WORK

Considering that this study focuses on improving schedule efficiency to accelerate CNN inference on DPU-based platform, this section mainly reviews the researches on the optimizations related to CNN inference on FPGAs.

Many researches have accelerated CNN inference by optimizing the network structure and model. In [20], the authors proposed a novel layer-based structured design method for full scalability in constructing CNNs, in which all the CNN layers are optimized and deployed separately and independently. In [21], the authors reduced the parameters and model sizes of CNNs. In [22], all the CNN layers were optimized and deployed to accelerate face feature extraction.

Meanwhile, many excellent algorithms have designed to accelerate the basic functions of CNN inference. In [23], two dedicated computing engines named Conv Engine and Dwcv Engine were designed for pointwise convolution and depthwise convolution to improve the efficiency. In [15], the authors aimed to accelerate sparse CNNs. In [24], an energy-efficient and high-throughput FPGA-based CNN accelerator was proposed. In [25], the fast Winograd algorithm was proposed to reduce the arithmetic complexity and improve the performance of CNNs on FPGAs.

To further improve the performance of CNN inference, the researchers have focused on the FPGA-based SoC architecture. In [26], a new idea was explored and implemented for basic processing element of CNN. In [27], the authors analyzed the on-chip and off-chip memory resources of FPGA, and proposed a memory optimization method. In [28], the accelerator has adopted full parallel pipeline structure to improve the computational efficiency for CNN inference. In [29], the authors proposed a framework to solve boundary problem and connect CNN accelerator with ARM processors and DDR4 memory through dual AXI bus. In [30], the authors proposed a new approach utilizing bit-width partitioning of FPGA DSP resources to improve the performance and resource utilization efficiency of CNN accelerator. In [9], the authors proposed a framework to enable fine-grained customization for CNNs with dynamic reconfiguration on FPGAs. In [17], the authors exploited the sparsity of neuron activations in CNNs to accelerate the computation and reduce memory requirements, and proposed NullHop accelerator.

Because of many excellent optimizations and designs, FPGA has been widely used in actual applications. In [31], an ADAS system was designed and FPGA was used to accelerate its CNN inference. In [3], [32] and [33], FPGA-based real-time reliable target detectors were implemented. In [34], the optimizations for CNN cascade face detection algorithm were proposed. However, until recently FPGAs were mostly
TABLE 3. Schedule efficiency of DPU1 is lower than that of DPU0.

|                | MobileNet-v2 | Inception-v1 | ResNet-50 | Yolo-v3 | VGG-16 |
|----------------|--------------|--------------|-----------|---------|--------|
| **DPU cores**  | 0            | 1            | 0         | 1       | 0      |
| 1 thread       | 29.1%        | -            | 37.9%     | -       | 57.4%  |
| (40 inferences per thread) |            |              |           |         | 68%    |
| 2 thread       | 55.8%        | 25.9%        | 40.6%     | 15.7%   | 61.8%  |
| (20 inferences per thread) |           |              |           |         | 58.6%  |
| 4 thread       | 48.3%        | 28.1%        | 60.1%     | 40.5%   | 93.4%  |
| (10 inferences per thread) |         |              |           |         | 64%    |
| 8 thread       | 59.8%        | 29.5%        | 51.5%     | 36.9%   | 71.2%  |
| (5 inferences per thread)    |            |              |           |         | 66.9%  |
| **Schedule efficiency**       |             |              |           |         |        |
| DPU0            | 68%          | -            | 84.7%     | -       |        |
| DPU1            | 42.9%        | -            | 76.5%     | -       |        |

programmed in hardware description languages (HDLs), which imply longer time-to-market and require from designers knowledge about hardware details in order for performance to be optimized, resulting in FPGAs being less used than other solutions. Therefore, they are not easy to design and optimize the FPGA-based accelerator of CNN inference in actual projects, especially for users without FPGA development experience. To solve above problem, the DPU-based CNN acceleration platform is used to deal with increasingly complex application scenarios. It is worth mentioning that the DPU-based acceleration platform supports ample basic functions of deep learning compared with previous optimizations and designs. The developers can take advantage of DPUs to accelerate CNN inference efficiently. Therefore, the problem of low schedule efficiency on DPU-based CNN acceleration platform should be studied.

III. THE EVALUATIONS AND EXPLORATIONS FOR SCHEDULE EFFICIENCY PROBLEMS

In this section, we evaluate the main causes of low schedule efficiency problem by running the typical lightweight MobileNet-v2, small-scale Inception-v1, medium-scale ResNet-50, large-scale Yolo-v3 and VGG-16, which are widely used in machine vision. We also explore the rules of the encapsulated scheduler, and propose an observation-based task assignment framework ($F_{\text{observation}}$). Because of the variety of CNNs, we select above representative CNNs with different scales for following evaluations and explorations.

A. THE PROBLEM OF UNBALANCED TASK ASSIGNMENT

Observation 1: When running inferences with one or multiple threads, the $S_{\text{original}}$ assigns too many tasks to DPU0 rather than DPU1, which exposes the problem of unbalanced task assignment.

In DPU-based CNN acceleration platform, schedule efficiency reflects the utilization of DPUs over a period of time proposed by Xilinx Inc, and low schedule efficiency means that the inference tasks take longer to complete their execution [18], [19]. Specifically, for each CNN, schedule efficiency of each DPU is expressed as:

$$SE = \frac{T_{\text{inference}}}{T_{\text{inference}} + T_{\text{interval}}} \times 100\%$$  (1)

where $T_{\text{inference}}$ and $T_{\text{interval}}$ represent the total inference time and total interval time for each DPU, respectively. For example, one inference time of network A is 3ms and the interval time between two inferences of network A is 8ms. Assume that the number of inferences is 2 for each DPU, that is, $T_{\text{inference}} = 6ms$, $T_{\text{interval}} = 8ms$ and $SE = \frac{6}{6+8} \times 100\% = 42.9\%$ for each DPU.

Table 3 shows schedule efficiency of 40 inferences on different CNNs under 1, 2, 4 and 8 threads. In Table 3, we observe that schedule efficiency of DPU1 is lower than that of DPU0, especially for lightweight MobileNet-v2 and small-scale Inception-v1. Moreover, the improvement of schedule efficiency of lightweight and small-scale CNNs by increasing the number of threads is not satisfactory. That is, although multithreading is an effective method to improve schedule efficiency, the degree of improvement varies with the different CNNs. For example, the maximum schedule efficiency of DPU0 is only 48.3% for MobileNet-v2 (4 threads), it means that DPU0 is idle for more than half of time during 40 inferences. Meanwhile, multithreading can improve schedule efficiency when the number of threads changes from 1 to 4. However, schedule efficiency declines when the number of threads increases from 4 to 8. In other words, the ideal performance cannot always be achieved by increasing the number of threads without limitation.

The analysis of above observations is as follows: (1) Firstly, the problem of unbalanced task assignment is ultimately caused by the DPU priority rule of the $S_{\text{original}}$ as mentioned in Section I-B. (2) Secondly, a complete inference process consists of two parts, that is, ARM is responsible for the task’s initialization, and then the initialized task is sent to DPU for execution. For small-scale CNNs, the initialization of the next inference may not be completed at the end of previous inference, and the opposite is true for large-scale CNNs. It rationalizes the difference of schedule efficiency for CNNs with various scales. (3) Thirdly,
the upper bound on threads is usually related to the number of CPU’s cores (ARM Cortex-A53 is a quad-core processor in zcu104), and the concurrency (parallelism) of four threads is perfectly supported in zcu104. As the number of threads continues to increase, it may aggravate the competition for CPU between threads, resulting in performance degradation.

B. THE PROBLEM OF UNUSED INTERVAL TIME BETWEEN TWO INFERENCES

Observation 2: In DPU-based CNN acceleration platform, the interval time between two inferences on DPUs for CNNs is relatively fixed (initialization time on ARM). And the interval time caused by the communication and data transmission time between FPGA and ARM affects schedule efficiency according to equation (1), especially for small-scale CNNs. In small-scale CNNs, schedule efficiency is low and the interval time even exceeds the inference time.

Fig. 5 shows the proportion of average inference time and average interval time (between two inferences) on DPUs for different CNNs. We find that interval time between two inferences of different networks is about the same (5-9ms). Furthermore, the interval time is caused by the communication and data transmission time between FPGA and ARM before the next inference. Meanwhile, the inference time of small-scale CNNs is less than the corresponding interval time, and multithreading cannot effectively improve schedule efficiency. However, the inference time of large-scale CNNs is much longer than the corresponding interval time. If small-scale CNNs can be inserted into the interval time of large-scale CNNs, schedule efficiency can be improved.

Table 4 shows the performance of inserting small-scale CNNs (MobileNet-v2 and Inception-v1) into the interval times of large-scale CNN (Yolo-v3). The number of threads is 2 for each CNN and the number of inferences is 20 for each thread. The second column (Yolo) in Table 4 is used as a comparison experiment. From Fig. 5, the total time to complete an CNN inference consists of interval time (initialization time on ARM) and inference time. Assume that DPU0 and DPU1 are responsible for 20 inferences, respectively. The total time for MobileNet-v2 and Inception-v1 to complete 20 inferences on DPUs is (7ms + 4ms) × 20 = 220ms and (8ms + 7ms) × 20 = 300ms, respectively. However, when running Yolo-v3 and MobileNet-v2, the total time of DPU0 and DPU1 only increased by 47ms and 17ms compared to comparison experiment, respectively. Similar results can be observed when running Yolo-v3 and Inception-v1. It is worth noting that the total time of DPU0 and DPU1 only increased by about 60ms even when running Yolo-v3, MobileNet-v2 and Inception-v1 simultaneously, which is much smaller than the theoretical total time of 20 inferences for MobileNet-v2 and Inception-v1 ((220 + 300) ms). The above results show that inserting small-scale CNNs into the interval time of large-scale CNNs is a feasible method to improve total schedule efficiency (hybrid CNNs with multithreading).
C. THE OBSERVATION-BASED TASK ASSIGNMENT FRAMEWORK

In the previous subsections, we evaluated main causes of low schedule efficiency problem. To satisfy the requirements of the complex and changeable application scenarios, we improve schedule efficiency based on our observations and analysis in this subsection.

Fig. 6 shows the relationship between threads and performance when the number of inferences is 40 for each CNN. In each case, 40 inferences are equally assigned to each thread. When the number of threads increases from 1 to 4, schedule efficiency increases and total time of 40 inferences decreases. However, this trend disappears as the number of threads continues to increase for each CNN. The related reasons for the above conclusions have been analyzed in Section III-A, namely, the upper bound on threads is related to the number of processor cores.

Since each CNN has the same trend, the upper bound on threads of each CNN is set to 3 in our observation-based task assignment framework (minimize the number of threads while ensuring sufficient performance for each CNN according to Fig. 6). Furthermore, the total threads of our framework is set to 6, because for each network, increasing the number of threads on this basis can no longer improve schedule efficiency (the upper bound on threads is related to the number of processor cores).

The details of the observation-based task assignment framework \( F_{\text{observation}} \) are explained as follows:

1. Firstly, the maximum threads on each CNN and framework are determined according to actual user requirements (single CNN or multiple CNNs). As mentioned earlier, the upper bound on threads of each CNN and framework is 3 and 6, respectively.

2. Secondly, the computing tasks are transferred to ARM, DPs or a candidate queue if multiple CNNs need to be processed.

3. Thirdly, if the actual project requirements include CNNs with various scales, each large-scale CNN with 3 threads is selected one by one, and then combined with one small-scale or medium-scale CNNs with 3 threads to insert into the interval time of large-scale CNN (according to the order of entering the queue). Note that the inference tasks of each CNN are assigned equally to each thread to reduce the difference of schedule efficiency between DPU0 and DPU1.

4. Finally, the inferences are running on ARM and DPs until all computing tasks in candidate queue are completed.

IV. THE EFFICIENT TASK ASSIGNMENT FRAMEWORK

Although the \( F_{\text{observation}} \) improves schedule efficiency from observations and analysis, it still has three shortcomings:

1. It fails to guarantee ideal performance by just increasing the number of threads.

2. It fails to improve schedule efficiency based on the features of CNNs with various scales, because the \( F_{\text{observation}} \) is based on coarse-grained observations.

3. It fails to support efficient inferences of multiple CNNs, because inferences are performed with multiple threads in out of order.

In this section, we tailor different strategies to improve schedule efficiency for small-scale, medium-scale and large-scale CNNs based on their corresponding inference time and interval time. Then, we integrate our optimizations and propose an efficient task assignment framework to maximize performance on DPU-based CNN acceleration platform. Our framework provide software designers with much more high-level processing power and an efficient way to accelerate CNN inference.

A. SCHEDULE EFFICIENCY IMPROVEMENT FOR SMALL-SCALE CNNs

Due to data transfer between FPGA and ARM, the interval time between two inferences is unavoidable for CNNs. Although the \( F_{\text{observation}} \) has improved schedule efficiency, the interval time is still hard to be fully utilized. For small-scale CNNs, the inference time is far less than the interval time. If some tasks are inserted into the interval time between two inferences, the schedule efficiency
can be effectively increased. Note that an inference in this paper represents the whole calculation process of a feature map (single-frame) from the first layer to the last layer of an CNN.

Fig. 7 shows our schedule efficiency improvement strategy for small-scale CNNs, and six threads are set up to accelerate the inferences of small-scale CNNs, called three groups round-robin strategy (ThGRR). The details of the ThGRR strategy are as follows:

(1) Firstly, all inference tasks are equally assigned to six threads, and six threads are divided into three groups.

(2) Secondly, group one (thread 1 and thread 2) are launched concurrently to assign inference tasks to DPU0 and DPU1 in parallel. According to the rules of the \( S_{original} \), DPU0 has a higher priority than DPU1, but as long as DPU0 is busy, DPU1 immediately accepts subsequent tasks for execution. Therefore, setting up two concurrent threads as a group can ensure that the two DPUs work in a nearly synchronous manner. As the number of threads increases, the task assignment tends to be chaotic.

(3) Thirdly, three groups of threads round-robin between ARM and DPUs until all inference tasks are completed.

Assume that only two threads (one group) are used, the tasks of two concurrent threads are executed either in ARM or in DPUs at the same time, it is difficult to make full use of the parallel computing power of heterogeneous ARM and DPUs to initializations and inferences simultaneously (the \( S_{original} \) in the top half of Fig. 7). Furthermore, in the case of four threads (two groups), when the inference tasks of a group are completed, the initializations of another group are not completed, which still lead to low schedule efficiency. Our ThGRR strategy for small-scale CNNs improves the utilization of heterogeneous ARM and DPUs, and provides a good balance between threads and performance (the \( F_{efficient} \) in the bottom half of Fig. 7). Meanwhile, a inference task can be inserted into the interval time if its inference time is less than this interval time. For small-scale CNNs, \( T_{inference} < T_{interval} \) is certain, but \( 2T_{inference} < T_{interval} \) is not guaranteed. Therefore, the ThGRR strategy ensures high schedule efficiency and avoids competition between threads. As the thread groups continue to increase, many tasks that have completed initialization will not have the appropriate interval time to be inserted. Otherwise, threads compete for DPUs, which usually leads to poor schedule efficiency.

FIGURE 7. Schedule efficiency improvement for small-scale CNNs (interval time - inference time).
B. SCHEDULE EFFICIENCY IMPROVEMENT FOR MEDIUM-SCALE CNNs

For medium-scale CNNs, the inference time is about equal to the interval time. Different from the ThGRR strategy adopted by small-scale CNNs, medium-scale CNNs has longer inference time. Therefore, the three groups with delay round-robin strategy (ThGDRR) is proposed to avoid competition for DPUs between threads.

Fig. 8 shows our ThGDRR strategy for medium-scale CNNs. Six threads (three groups) are set up to accelerate the inferences of medium-scale CNNs. Similar to small-scale CNNs, the three groups round-robin strategy is adopted as a basis, which has been introduced in the previous subsection. But the difference is that initialization time is about equal to inference time for medium-scale CNNs, and the delay between thread groups should be added to ensure that there is no competition between inference tasks. The delay should satisfy:

\[ T_{\text{interval, new}} = (T_{\text{interval}} + \text{delay}) > T_{\text{inference}} \]  

where \( T_{\text{interval}} \) and \( T_{\text{inference}} \) can be determined by Xilinx performance monitoring tool Dsight, so as to ensure that inference tasks of medium-scale CNNs can be completely inserted into new interval time.

If no delay is added between thread groups, the competition among tasks is inevitable, which affects performance (schedule efficiency cannot be guaranteed). Meanwhile, it is unnecessary to continue to add thread groups, because at most one inference task can be inserted in interval time after adding a delay.

C. SCHEDULE EFFICIENCY IMPROVEMENT FOR LARGE-SCALE CNNs

For large-scale CNNs, the inference time is much longer than the interval time. Therefore, four threads are set up to accelerate the inference tasks of large-scale CNNs, and it is enough to satisfy the requirement of high schedule efficiency.

Fig. 9 shows our two groups round-robin strategy (TwGRR) for large-scale CNNs.

1) Firstly, group one (thread 1 and thread 2) are launched simultaneously to execute inference tasks on DPU0 and DPU1 in parallel.

2) Secondly, when inference tasks of group one are completed, inference tasks of group two (thread 3 and thread
D. AN EFFICIENT TASK ASSIGNMENT FRAMEWORK

In this subsection, an efficient task assignment framework \(F_{\text{efficient}}\) is proposed. Different from the \(F_{\text{observation}}\), where multiple CNNs are mixed and executed in out of order and competition between inference tasks is inevitable. In fact, our \(F_{\text{observation}}\) only considers the impact of thread’s increase on performance. The proposed \(F_{\text{efficient}}\) takes the competition among threads into consideration, thus further improving the schedule efficiency. The \(F_{\text{efficient}}\) pays more attention to high performance implementation of each CNN (all CNNs are executed one by one in \(F_{\text{efficient}}\)). The details of \(F_{\text{efficient}}\) are as follows:

1. Firstly, the \(F_{\text{efficient}}\) determines large-scale, medium-scale and small-scale CNNs according to the proportion of inference time and interval time. It is easily determined by using Xilinx performance monitoring tool Dsight.

2. Secondly, the \(F_{\text{efficient}}\) adopts different strategies (ThGRR, ThGDRR and TwGRR) according to CNNs with various scales.

3. Finally, all computing tasks in candidate queue are completed.

V. EXPERIMENTS

This study aims to address the low schedule efficiency problem on DPU-based CNN acceleration platform by proposing an efficient task assignment framework \(F_{\text{efficient}}\). The performance metrics selected for comparison is total time by running experiments of small-scale, medium-scale, large-scale and multiple scales CNNs. The compared frameworks with the presented \(F_{\text{efficient}}\) are the official original strategy \(S_{\text{original}}\) and our observation-based task assignment framework \(F_{\text{observation}}\). All inference tasks are executed on Xilinx UltraScale+ MPSoC zcu104 platform with heterogeneous Cortex A53 ARM and ZU7 FPGA (two DPUs).

A. INFERENCE FOR SMALL-SCALE CNNs

In our study, CNNs are divided into three types: large-scale, medium-scale and small-scale. Our \(F_{\text{efficient}}\) supports corresponding task assignment strategies (ThGRR, ThGDRR and TwGRR) for different CNNs.
Table 5 shows the total time of lightweight MobileNet-v2 and small-scale Inception-v2 in difference task assignment frameworks, and inference time is less than interval time for small-scale CNNs. In Table 5, forty inference tasks are evenly assigned to two threads, three threads and six threads for $S_{original}$, $F_{observation}$ and $F_{efficient}$, respectively.

According to Fig. 5, the total time of one inference of MobileNet-v2 is composed of inference time of 4ms and interval time of 7ms, while the total time of one inference for Inception-v1 is composed of inference time of 7ms and interval time of 8ms. If using the $S_{original}$ with single thread, it takes 440ms and 600ms to complete 40 inferences for MobileNet-v2 and Inception-v2, that is, ARM (initializations) and DPU0 (inferences) execute serially, while DPU1 is idle. When using the $S_{original}$ with two threads, the total time to complete 40 inference tasks is reduced to 220ms and 300ms for each DPU, respectively. Apparently, multithreading strategy accelerates inferences of MobileNet-v2 and Inception-v2, and improves the schedule efficiency significantly. However, the schedule efficiency of MobileNet-v2 and Inception-v2 can still be improved. Although the $S_{original}$ makes use of DPU1, the interval time between two inferences still has potential to be exploited. Specifically, in the $S_{original}$, 40 inference tasks are equally assigned to two DPU (each DPU undertakes 20 inference tasks). For above two small-scale CNNs, the total inference time of each DPU is 4 × 20.80ms and 7 × 20.140ms, and the total interval time of each DPU is 7 × 20.140ms and 8 × 20.160ms, respectively. Schedule efficiency is improved in our proposed $F_{observation}$ and $F_{efficient}$. In our $F_{observation}$, the total time of each DPU is reduced to 164ms and 245ms for above two small-scale CNNs, respectively. That is, the unused interval time of each DPU is shortened to 84ms and 104ms, while in the $S_{original}$, the number is 140ms and 160ms respectively. In our $F_{efficient}$, the total time of each DPU is further reduced to 115ms and 210ms for MobileNet-v2 and Inception-v2. And the total interval time

|                | MobileNet-v2  | Inception-v2 |
|----------------|---------------|--------------|
| $S_{original}$ | 220ms (SE:36%) | 300ms (SE:46%) |
| $F_{observation}$ | 164ms (SE:48%) | 245ms (SE:57%) |
| $F_{efficient}$ | 115ms (SE:70%) | 210ms (SE:67%) |

* SE: schedule efficiency.
of each DPU is 35ms (140ms in the $S_{\text{original}}$) and 70ms (160ms in the $S_{\text{original}}$) for MobileNet-v2 and Inception-v2. Compared with the $S_{\text{original}}$, the total time of each DPU is reduced by about 100ms and 90ms, respectively.

### B. INFERENCES FOR MEDIUM-SCALE CNNs

Table 6 shows the total time of medium-scale ResNet-50 in difference task assignment frameworks, and inference time is about equal to interval time for medium-scale CNNs. In Table 6, forty inferences are evenly assigned to two threads, three threads and six threads for $S_{\text{original}}$, $F_{\text{observation}}$, and $F_{\text{efficient}}$, respectively.

**TABLE 6.** Medium-scale CNNs running on different task assignment frameworks (40 inferences).

|                  | ResNet-50 |
|------------------|-----------|
| $S_{\text{original}}$ | 420ms (SE:65%) |
| $F_{\text{observation}}$ | 341ms (SE:77%) |
| $F_{\text{efficient}}$ | 299ms (SE:95%) |

* SE: schedule efficiency.

According to Fig. 5, the total time of one inference of ResNet-50 is composed of inference time of 14ms and interval time of 7ms. If using the $S_{\text{original}}$ with single thread, it takes 840ms to complete 40 inferences for ResNet-50. When using the $S_{\text{original}}$ with two threads, the total time to complete 40 inference tasks is reduced to 420ms for each DPU. In the $S_{\text{original}}$, the total inference time of each DPU is $14 \times 20.280ms$, and the total interval time of each DPU is $7 \times 20.140ms$. Schedule efficiency is improved in our proposed $F_{\text{observation}}$ and $F_{\text{efficient}}$. In our $F_{\text{observation}}$, the total time of each DPU is reduced to 361ms for ResNet-50. That is, the unused interval time of each DPU is shortened to 81ms, while in the $S_{\text{original}}$, the number is 140ms. Furthermore, in our $F_{\text{efficient}}$, the total time of each DPU is reduced to 295ms, and the total interval time of each DPU is 15ms for ResNet-50. Compared with the $S_{\text{original}}$, the total time of each DPU is reduced by about 125ms.

### C. INFERENCES FOR LARGE-SCALE CNNs

Table 7 shows the total time of large-scale Yolo-v3 and VGG-16 in difference task assignment frameworks, and inference time is much larger than interval time for large-scale CNNs. In Table 7, forty inferences are evenly assigned to two threads, three threads and four threads for $S_{\text{original}}$, $F_{\text{observation}}$, and $F_{\text{efficient}}$, respectively.

According to Fig. 5, the total time of one inference of Yolo-v3 is composed of inference time of 28ms and interval time of 8ms, while the total time of one inference for VGG-16 is composed of inference time of 36ms and interval time of 6ms. In the case of using the $S_{\text{original}}$ with single thread, it takes 1440ms and 1680ms to complete 40 inferences for Yolo-v3 and VGG-16. When using the $S_{\text{original}}$ with two threads, the total time to complete 40 inference tasks is reduced to 720ms and 840ms for each DPU, respectively. For above two large-scale CNNs, the total inference time of each DPU is $28 \times 20.560ms$ and $36 \times 20.720ms$, and the total interval time of each DPU is $8 \times 20.160ms$ and $6 \times 20.120ms$, respectively. In our $F_{\text{observation}}$, the total time of each DPU is reduced to 602ms and 755ms for above two large-scale CNNs, respectively. That is, the unused interval time of each DPU is shortened to 42ms and 35ms, while in the $S_{\text{original}}$, the number is 160ms and 120ms respectively. In our $F_{\text{efficient}}$, the total time of each DPU is further reduced to 578ms and 725ms for Yolo-v3 and VGG-16. And the total interval time of each DPU is 18ms (160ms in the $S_{\text{original}}$) and 5ms (120ms in the $S_{\text{original}}$) for Yolo-v3 and VGG-16. Compared with the $S_{\text{original}}$, the total time of each DPU is reduced by about 142ms and 115ms, respectively.

### D. INFERENCES FOR CNNs WITH MULTIPLE SCALES

Table 8 shows the total time of five CNNs with various scales in difference task assignment frameworks. In Table 8, twelve inferences of each CNN are assigned to two threads, three threads and four (or six) threads for $S_{\text{original}}$, $F_{\text{observation}}$, and $F_{\text{efficient}}$, respectively. Note that the upper bound on threads is 6 in the $F_{\text{observation}}$, and schedule efficiency is not listed here because it involves multiple CNNs (the results of the first three experiments can be used as a reference).

**TABLE 7.** Large-scale CNNs running on different task assignment frameworks (40 inferences).

|                  | Yolo-v3 | VGG-16 |
|------------------|---------|--------|
| $S_{\text{original}}$ | 720ms (SE:77%) | 840ms (SE:85%) |
| $F_{\text{observation}}$ | 602ms (SE:93%) | 755ms (SE:95%) |
| $F_{\text{efficient}}$ | 578ms (SE:96%) | 725ms (SE:99%) |

* SE: schedule efficiency.

In the case of using the $S_{\text{original}}$ with single thread, it takes 1500ms to complete 12 $\times$ 5.60 inferences for five CNNs with various scales. When using the $S_{\text{original}}$ with two threads, the total time to complete 60 inference tasks is reduced to
750ms for each DPU. In our $F_{\text{observation}}$, the total time of each DPU is reduced to 638ms, while in our $F_{\text{efficient}}$, the total time of each DPU is further reduced to 598ms. Compared with the $S_{\text{original}}$, the total time of each DPU is reduced by about 152ms.

VI. CONCLUSION

This study aims to present a high performance task assignment framework built upon Xilinx hybrid CPU-FPGA SoC devices with DPU IP. We first evaluate the main causes of low schedule efficiency problem. Then, we explore the scheduler rules and improve schedule efficiency through observations and analysis. Finally, we integrate our optimizations, and propose an efficient task assignment framework to maximize performance on DPU-based CNN acceleration platform. Experimental results on Xilinx Zynq UltraScale+ MPSoC zcu104 show that the efficient task assignment framework significantly boosts schedule efficiency for inferences of CNNs with various scales compared with original schedule strategy.

In the future, we will optimize the problems that still exist on the current DPU-based acceleration platform: (1) the existing platform only supports the acceleration of deep learning-related functions even if there are still a lot of computing resources available (the computing resources occupied by DPU IP are adjustable according to actual needs), which requires software-hardware re-development and provides related driver support for the entire platform; (2) on the basis of (1), an efficient task scheduler between ARM, DPs, and custom hardware accelerators should be designed to maximize system performance.

REFERENCES

[1] E. Delays, A. Srisrao, C. Duda, and S. Das, “Deep learning challenges and solutions with Xilinx FPGAs,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), Nov. 2017, pp. 908–913.
[2] C. Hao and D. Chen, “Deep neural network model and FPGA accelerator co-design: Opportunities and challenges,” in Proc. 14th IEEE Int. Conf. Solid-State Integ. Circuit Technol. (IISICIT), Oct. 2018, pp. 1–4.
[3] S. Fang, L. Tian, J. Wang, S. Liang, D. Xie, Z. Chen, L. Sui, Q. Yu, X. Sun, Y. Shan, and Y. Wang, “Real-time object detection and semantic segmentation hardware system with deep learning networks,” in Proc. Int. Conf. Field-Program. Technol. (FPT), Dec. 2018, pp. 389–392.
[4] X. Li, L. Ding, L. Wang, and F. Cao, “FPGA accelerates deep residual learning for image recognition,” in Proc. IEEE 2nd Inf. Technol., Netw., Electron. Automat. Control Conf. (ITNEC), Dec. 2017, pp. 837–840.
[5] A. Shawahma, S. M. Sait, and A. El-Maleh, “FPGA-based accelerators of deep learning networks for learning and classification: A review,” IEEE Access, vol. 7, pp. 7823–7859, 2019.
[6] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, “Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs,” in Proc. 54th Annua. Design Automat. Conf. (DAC), 2017, pp. 1–6.
[7] T. V. Huynh, “Deep neural network accelerator based on FPGA,” in Proc. 4th NAFOSTED Conf. Inf. Comput. Sci., Nov. 2017, pp. 254–257.
[8] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou, “A deep learning prediction process accelerator based FPGA,” in Proc. 15th IEEE/ACM Int. Symp. Cluster. Cloud Grid Comput., May 2015, pp. 1159–1162.
[9] Q. Xiao and Y. Liang, “Fuse: An FPGA tuning framework for CNN acceleration,” IEEE Des. Test, vol. 37, no. 1, pp. 46–55, Feb. 2020.
[10] L. Lu and Y. Liang, “SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGA,” in Proc. 55th ACM/ESDA/IEEE Design Automat. Conf. (DAC), Jun. 2018, pp. 1–6.
[11] Z. Lin, J. M. Ota, J. D. Owens, and P. Muyan-Ozcelik, “Benchmarking deep learning frameworks with FPGA-suited models on a traffic sign dataset,” in Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2018, pp. 1197–1203.
[12] H. Hegde, Siddhartha, N. Ramasamy, V. Buddhha, and N. Kapre, “Evaluating embedded FPGA accelerators for deep learning applications,” in Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), May 2016, p. 25.
[13] Y. Zhao, X. Zhang, X. Fang, L. Li, X. Li, Z. Guo, and X. Liu, “A deep residual networks accelerator on FPGA,” in Proc. 11th Int. Conf. Adv. Comput. Intell. (ICACI), Jun. 2019, pp. 13–17.
[14] L. Stornaiuolo, M. Santambrogio, and D. Sciento, “On how to efficiently implement deep learning algorithms on FYNQ platform,” in Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI), Jul. 2018, pp. 587–590.
[15] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, “An efficient hardware accelerator for sparse convolutional neural networks on FPGAs,” in Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), Apr. 2019, pp. 17–25.
[16] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, “Frequency improvement of systolic array-based CNNs on FPGAs,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2018, pp. 1–5.
[17] A. J. K. Mostafa, E. Calabrese, R. Rios-Navarro, R. Tapia-Morales, I.-A. Lunga, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, “NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 3, pp. 644–656, Mar. 2019.
[18] G. Peng, Z. Hu, S. Chen, and F. Wu, “Energy-efficient and high-throughput FPGA-based accelerator for computational neural networks,” in Proc. 13th IEEE Int. Conf. Solid-State Integ. Circuit Technol. (ICSICT), Oct. 2017, pp. 1037–1040.
[19] M. Hailesellie, S. R. Hasan, F. Khalid, F. A. Wad, and M. Shafique, “FPGA-based convolitional neural network architecture with reduced parameter requirements,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2018, pp. 1–5.
[20] R. Ding, G. Su, G. Bai, W. Xu, N. Su, and X. Wu, “A FPGA-based accelerator of convolutional neural network for face feature extraction,” in Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC), Jun. 2019, pp. 1–3.
[21] Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, “A low latency and high performance CNN processor based on FPGA for MobileNets,” in Proc. 29th Int. Conf. Field Program. Leg. Appl. (FPL), Sep. 2019, pp. 136–143.
[22] G. Feng, Z. Hu, S. Chen, and F. Wu, “Energy-efficient and high-throughput FPGA-based accelerator for convolutional neural networks,” in Proc. 13th IEEE Int. Conf. Solid-State Integ. Circuit Technol. (ICSICT), Oct. 2016, pp. 624–626.
[23] Y. Liang, L. Lu, Q. Xiao, and S. Yan, “Evaluating fast algorithms for convolutional neural networks on FPGAs,” IEEE Trans. Comput.-Aided Design Integ. Circuits Syst., vol. 39, no. 4, pp. 857–870, Apr. 2020.
[24] F. U. D. Farnukh, T. Xie, C. Zhang, and Z. Wang, “Optimization for efficient hardware implementation of CNN on FPGA,” in Proc. IEEE Int. Conf. Integ. Circuits, Technol. Appl. (ICTA), Nov. 2018, pp. 88–91.
[25] X. Zhang, H. Pan, D. Zhang, Q. Sun, and W. Lin, “A memory-optimized and energy-efficient CNN architecture based on FPGA,” in Proc. IEEE 28th Int. Symp. Ind. Electron. (ISIE), Jun. 2019, pp. 2137–2141.
[26] L. Zong-Ling, W. Lu-Yuan, Y. Ji-Yang, C. Bo-Wen, and H. Liang, “The design of lightweight and multi parallel CNN accelerator based on FPGA,” in Proc. IEEE 8th Joint Int. Inf. Technol. Artif. Intell. Conf. (ITAIc), May 2019, pp. 1521–1528.
[27] X. Li, Y. Cai, J. Han, and X. Zeng, “A high utilization FPGA-based accelerator for variable-scale convolutional neural network,” in Proc. IEEE 12th Int. Conf. ASCON (ASICON), Oct. 2017, pp. 944–947.
[28] J. Guo, S. Yin, P. Ouyang, L. Liu, and S. Wei, “Bit-width based resource partitioning for CNN acceleration on FPGA,” in Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), Apr. 2017, p. 31.
[29] Y. Shan, “ADAS and video surveillance analytics system using deep learning algorithms on FPGA,” in Proc. 28th Int. Conf. Field Program. Leg. Appl. (FPL), Aug. 2018, pp. 465–468.
[32] M. Shimoda, Y. Sada, R. Kuramochi, and H. Nakahara, “An FPGA implementation of real-time object detection with a thermal camera,” in Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL), Sep. 2019, pp. 413–414.

[33] A. Jinguiji, Y. Sada, and H. Nakahara, “Real-time multi-pedestrian detection in surveillance camera using FPGA,” in Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL), Sep. 2019, pp. 424–425.

[34] B.-S. Yu, Y. Tsao, S.-W. Yang, Y.-K. Chen, and S.-Y. Chien, “Architecture design of convolutional neural networks for face detection on an FPGA platform,” in Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS), Oct. 2018, pp. 88–93.

JIANG ZHU received the M.S. and Ph.D. degrees in control science and engineering from Hunan University, Changsha, China, in 2005 and 2011, respectively. He is currently an Associate Professor with the School of Automation and Electronics Information, Xiangtan University, Xiangtan. His current research interests include intelligent information processing, mobile communication networks, and parallel distributed systems.

LIZAN WANG is currently pursuing the M.S. degree with Xiangtan University, Xiangtan, China. His current research interests include parallel distributed systems and computer architecture.

HAOLIN LIU received the B.Eng. and Ph.D. degrees from Sichuan University, Chengdu, China, in 2010 and 2015, respectively. He is currently an Associate Professor with the College of Information Engineering, Xiangtan University, Xiangtan, China. His research interests include mobile edge computing, wireless sensor networks, the Internet of Things, and parallel distributed systems. He is a member of the CCF.

SHUJUAN TIAN received the B.Eng. and M.Eng. degrees from Xiangtan University, in 2004 and 2009, respectively, and the Ph.D. degree from Central South University, in 2016. She is currently a Vice Professor with the College of Information Engineering, Xiangtan University. Her research interests include VANETs, the Internet of Things, wireless signal control and processing, compressed sensing, and parallel distributed systems.

QINGYONG DENG (Member, IEEE) received the master’s degree in signal and information processing from Xiangtan University, China, in 2009, and the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), China, in 2019. He is currently an Associate Professor with the College of Information Engineering, Xiangtan University. He has published more than ten refereed journal articles. His current research interests include the Internet of Things, compressed sensing, wireless networks, and 5G and parallel distributed systems. He is a member of the CCF.

JIANQI LI received the Ph.D. degree in control science and engineering from Central South University, Changsha, China, in 2013. He is currently a Professor with the Department of Communication and Electrical Engineering, Hunan University of Arts and Science. His main research interests are in intelligent information processing, image processing, and pattern recognition.

***