electronics

MDPI

Article

Towards Intelligent Edge Computing: A Resource- and Reliability-Aware Hybrid Scheduling Method on Multi-FPGA Systems

Zeyu Li [1,2,*], Yuchen Hao [1], Hongxu Gao [2] and Jia Zhou [3]

[1] School of Computer Science and Technology, North University of China, Taiyaun 030051, China; s202307014@st.nuc.edu.cn
[2] School of Computer Science and Technology, Xidian University, Xi'an 710071, China; hx_gao@stu.xidian.edu.cn
[3] School of Microelectronics, Northwestern Polytechnical University, Xi'an 710071, China; zhoujia@eng-wx.com
* Correspondence: 20230101@nuc.edu.cn; Tel.: +86-1500-343-9992

Abstract: Multi-FPGA systems can form larger and more powerful computing units through high-speed interconnects between chips, and are beginning to be widely used by various computing service providers, especially in edge computing. However, the new computing architecture brings new challenges to efficient and reliable task scheduling. In this context, we propose a resource- and reliability-aware hybrid scheduling method on Multi-FPGA systems. First, a set of models is established based on the resource/time requirements, communication overhead, and state conversion process of tasks to further analyze the constraints of system scheduling. On this basis, the large task is divided into subtasks based on the data dependency matrix, and the Maintenance Multiple Sequence (MMS) algorithm is used to generate execution sequences for each subtask to the Multi-FPGA systems to fully exploit resources and ensure reliable operation. Compared with state-of-the-art scheduling methods, the proposed method can achieve an average increase in resource utilization of 7%; in terms of reliability, it achieves good execution gains, with an average task completion rate of 98.3% and a mean time to failure of 15.7 years.

Keywords: resource management; reliability; hybrid scheduling; Multi-FPGA

check for updates

Academic Editor: Christos J. Bouras

Received: 30 October 2024
Revised: 23 December 2024
Accepted: 25 December 2024
Published: 27 December 2024

Citation: Li, Z.; Hao, Y.; Gao, H.; Zhou, J. Towards Intelligent Edge Computing: A Resource- and Reliability-Aware Hybrid Scheduling Method on Multi-FPGA Systems. *Electronics* **2025**, *14*, 82. https://doi.org/10.3390/electronics14010082

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Commercial Off The Shelf (COTS) Field Programmable Gate Arrays (FPGAs) support partial run-time reconfigurability, which allows some FPGA resources to be configured without affecting the operation of other resources. This technology allows multiple tasks to be executed on the FPGA in a spatially and temporally multiplexed manner [1,2]. However, a single FPGA faces resource constraints. Therefore, the resource requirements of an application can often exceed the resources available on a single FPGA. Even if multiple run-time reconfigurations are allowed, this may not alleviate the resource constraints of end-side intelligent computing applications. Today, many edge computing applications require the simultaneous use of multiple FPGAs (also known as Multi-FPGA systems) to perform their operations [3,4], such as in urban smart transportation applications. There are multiple FPGAs required to perform tasks such as real-time traffic monitoring (camera image processing and traffic monitoring), traffic signal control optimization (signal control and dynamic switching), and abnormal event detection and response (traffic accidents and road construction). The system architecture is shown in Figure 1. A Multi-FPGA system is responsible for submitting and distributing tasks by a client controller, with

multiple FPGAs serving as servers to execute tasks and return processing results to the client. Among them, multiple FPGAs are connected to the central client, and the link is built based on a dual backup method.

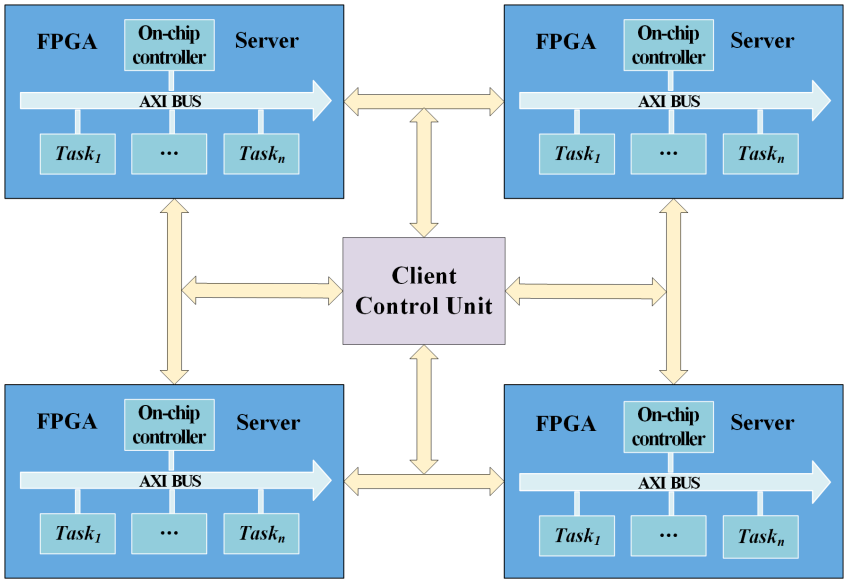


Figure 1. Multi-FPGA systems.

While Multi-FPGA system architectures and Dynamic Partial Reconfiguration (DPR) [5] techniques offer great flexibility, the overheads incurred during system scheduling management and reconfiguration must be carefully considered, as they can easily jeopardize the performance gains achieved through hardware acceleration [6,7]. The Multi-FPGA architecture is like a single FPGA chip, a 'black box' that is invisible to the user. Still, there are some issues that should be taken into account when implementing applications on it: scheduling order, deployment regions, resource constraints, and complex communication overheads between tasks. Combined with the application of DPR technology, further requirements for segmentation and reorganization of large tasks are presented.

Multi-FPGA systems have multiple FPGA resources, and each FPGA can be divided into different reconfigurable blocks for fine-grained resource control. Deploying tasks with different resource requirements and execution times to the corresponding blocks helps to efficiently utilize resources for further deployment of more hardware accelerators working simultaneously. Task scheduling implements resource resilience policies as well as management of task configuration and execution order [8–10]. The DPR technique can reuse the resources of each FPGA in a time-shared manner, providing more options and optimizations for resource management and task deployment. Therefore, it is a challenge to efficiently utilize multiple FPGAs and the resources on each FPGA for more efficient task execution.

Overall, the task scheduling problem is NP-hard because it is a more general optimization version of the Resource Constrained Scheduling Problem (RCSP), which is NP-complete. At the same time, existing research methods for resource scheduling suffer from problems such as low resource utilization, lack of reliable operation guarantees, and inapplicability to the Multi-FPGA system. In this work, we propose a resource- and reliability-aware hybrid scheduling method on a Multi-FPGA system. The proposed method is experimentally verified and the effectiveness of the management and scheduling is demonstrated. In this context, our contributions are summarized as follows.

(1) A resource-aware scheduler to make sure that the final system can make use of the available FPGA resources more efficiently. When scheduling tasks, it can provide the

minimum resources for each task as a service objective, i.e., to schedule as few on-chip resources as possible for all tasks received within a time unit to improve resource utilization.

(2) A reliability-aware scheduler to ensure the the successful execution of tasks and improve the expected lifetime in the Multi-FPGA system. The scheduler implements multi-task scheduling across multiple FPGAs, balancing makespan and reliability by placing tasks in a redundant and load-balanced manner while ensuring execution efficiency.

(3) The experimental results show that the resource management method is able to effectively combine and place the tasks, with an average increase in resource utilization of 7%; the reliability-aware scheduling method achieves good execution gains, with an average task completion rate of 98.3% and a system's lifetime of 15.7 years.

The paper is structured as follows. Section 2 discusses related work on FPGA task scheduling. In Section 3, we provide a detailed description of the resource- and reliability-aware task scheduler. In Section 4, we evaluate the performance of our method. Section 5 concludes the paper and proposes future work.

2. Related Work

Many previous studies have focused on task planning on a single FPGA. A few other studies in the literature specifically focused on task scheduling in Multi-FPGA systems. The research course of task scheduling on Multi-FPGA systems is mainly reflected in the task model, task sequence, and task state management. In application, the main difference between scheduling algorithms is whether the task execution mode is determined and whether the task has a deadline time. The scheduling goals of the algorithms are mostly to achieve shorter task execution time or less resource consumption.

Tang, Y. et al. proposed a method to build task queues based on task deadline and task slack time and to build a task model including required resources and task execution time [11]. Deiana, E. A. et al. proposed an improved method in which resources are requested as late as possible to reduce the amount of resources [12]. Sun, Z. et al. proposed a two-phase task scheduling approach to optimize task execution efficiency in a multi-FPGA system but did not consider load balancing of the system [4]. Unlike a software strategy, the simple hardware real-time scheduler can directly configure the tasks in the ready queue according to priority and achieve fast scheduling [13], which is simple and fast for scheduling, but cannot control global indicators such as load balancing.

Some scheduling algorithms are based on shape modeling, by modeling tasks/schedulable regions as rectangular blocks for task-resource mapping [14–16]. Task scheduling is modeled using Mixed-Integer Linear Programming (MILP) formulas based on an architecture with FPGA resources [17]. The grey correlation degree indicates the degree of correlation between two parameters in the system [18]. Using the grey relation theory, a task model including task size, task shape, execution time, and configuration time is established, which is a way to generate a scheduling sequence with the shortest execution time [19]. Considering the task shape and computation time in the task model [20], a task implementation that is slow to execute but suitable for the current resource is selected during scheduling. Therefore, the on-chip resources can satisfy the resource requirements of more tasks, improve resource utilization, and reduce the task waiting time.

In the above-mentioned studies on task scheduling for the Multi-FPGA system, most of them focus on improving the efficiency of task execution with little consideration of reliability assurance. In this work, we propose a reliability-aware, multi-task multi-device scheduling approach that aims to ensure fault tolerance during task execution and to balance the task load among multiple FPGAs to extend the system lifetime.

3. Resource- and Reliability-Aware Task Scheduler

In this section, we design a resource- and reliability-aware task scheduler that schedules multiple subtasks of a large task for execution in a load-balanced and fault-tolerant manner in a Multi-FPGA system. First, we build a task resource model and a timing model, and analyze the state changes during the task scheduling process. Second, we further analyze the impact of multiple FPGA systems on the communication process and establish a model for inter-FPGA task communication. On this basis, we propose the task hybrid algorithm for solving the multi-task scheduling problem on multiple FPGA chips.

3.1. Model

3.1.1. Resource Model of the Task

The task model includes the input and output of the algorithm and each task needs to be represented in the format of the model to generate a task vector. The task model is divided into a pre-scheduled model m_{pre} and a post-scheduled model m_{pos}, which are, respectively, used to describe the resource requirements of the task and the resource description obtained by the task. The m_{pre} includes width of the resources required by the task W_i and height of the resources required by the task H_i, processing time of the data in the current task $times_{exe}$, specified deadline $Deadline_i$, input data $Data_{in}$, output data $Data_{out}$, and task profile partial BIT bit_i. The m_{pre} is as follows: $m_{pre} = \{W_i, H_i, time_{exe}, Deadline_i, Data_{in}, Data_{out}, bit_i\}$. The m_{pos} includes the FPGA chip number at which the task is placed F_i, the coordinate range of the on-chip resources x_i, y_i the start time of the on-chip resources s_i and end time of the on-chip resources e_i, the required storage resources $Data_{max}$, and the time occupied by the bandwidth B_i. The m_{pos} is as follows: $m_{pos} = \{F_i, x_i, y_i, s_i, e_i, Data_{max}, B_i\}$.

3.1.2. Time Model of the Task

From a time perspective, the on-chip tasks go through four stages: task configuration time $time_{conf}$, data reception time $time_{train-in}$, execution data time $time_{exe}$, and data transmission time $time_{train-out}$, which constitute the shortest time $time_{min}$ for a single task to be on the FPGA chip. At different stages, there is an indefinite length of waiting time $time_{wait}$ due to data dependencies between tasks. So, the global time of the task $time_{total}$ is as follows: $time_{total} = time_{wait} + time_{min}$. When calculating the amount of resources occupied by a task, the amount of resources is multiplied by a $time_{total}$ way of reflecting the impact of resource occupation time on the total amount of system resources.

3.1.3. Overhead of Task Communication

The communication overhead of the tasks depends on the system architecture. This work is directed to a system using the AXI bus structure, where there is a continuously active master node that organizes all communication. The communication overhead can be divided into device internal communication overhead (Figure 2) and device-to-device communication overhead (Figure 1).

Communication between subtasks configured on the same FPGA is relatively simple. After the subtask in the pending receive data state receives the bus, the master node checks the task address of the data to be sent in the address mapping table and sends a read command to the task that is sending data. Then, the master node retrieves the data and sends it to the subtask that has pending received data. The communication overhead corresponding to the communication process is expressed as follows:

$$time_{train-i,j} = (Data_{out-i,j} / Rate_{AXI}) * 2 \quad (1)$$

Communication of subtasks configured on different FPGAs are more complicated. When the subtask in the state of pending receiving data obtains the bus, the master node checks the task location of the data to be sent in address mapping table and send a request of reading data to another device. It is needed to send a bus request to the CPU and the other device when sending data and keep the local bus occupied while waiting for the license. Finally, the communication overhead of local device $time_{tran-receive}$ and the communication overhead of the other device $time_{tran-send}$ are expressed as follows:

$$\begin{aligned} time_{cpu,j} &= Data_{out-i,j}/Rate_{pcie} + Date_{out-i,j}/Rate_{AXI} \\ time_{tran-wait} &= max(Bus_{CPU}, Bus_{FPGA}) \\ time_{tran-send} &= time_{CPU,j} \\ timetran - receive &= \\ max(Bus_{CPU}, Bus_{FPGA}) &+ time_{CPU,j}*2 \end{aligned} \tag{2}$$

By describing the two communication processes above, it can be seen that the communication overhead between devices is significantly higher than the communication overhead within the device. Therefore, by default, when the configuration sequence is generated, the subtasks belonging to a task are configured on the same device. Only if the current device does not have sufficient resources or the task cannot be completed before the deadline, is it necessary to try to configure the other devices. If all devices cannot meet the resource requirements of the subtasks, the subtasks must be reapplied the next time.

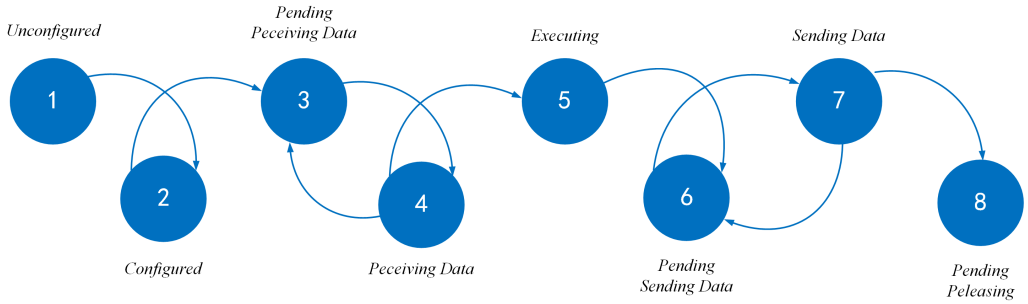


Figure 2. The state transition diagram of subtasks.

3.1.4. State Conversion Process of Task

Depending on the time structure of the subtask, the execution of the subtask can be represented by eight states: unconfigured, configured, pending receiving data, receiving data, executing, pending sending data, sending data, and pending releasing. The initialization state of each task is the unconfigured state, the end state is the releasable state, and the transition relationship between the start and end of the eight tasks is shown in Figure 3.

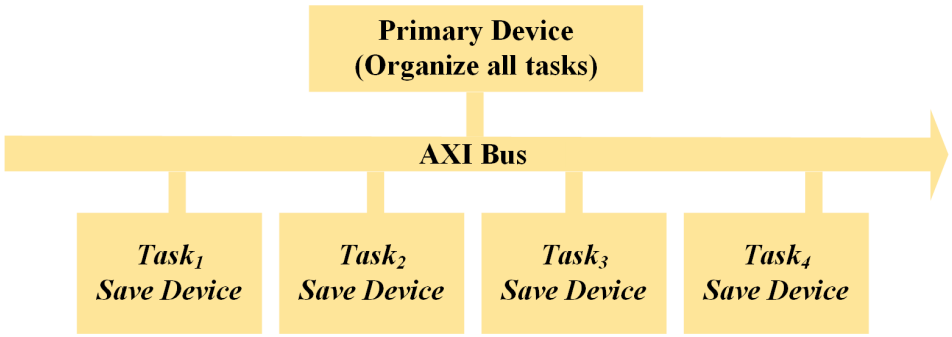


Figure 3. On-chip system structure diagram.

3.2. Subtask Sequence Division

Task scheduling requires time constraints on tasks. The main purpose of topological ordering is to generate a task order that transitions from an unconfigured state to a configured state, which is relied upon to carry the work forward. Because each branch in the DAG of a large task is of different lengths, the different topological ordering will affect the execution time of global tasks. In addition, topological ordering can also prevent resource deadlocks.

The generation of topological sorting is based on a data dependency matrix. According to the data dependency matrix, all feasible solutions for each large task are stored in a tree structure, and the tasks on which the pre-tasks depend are configured before configuration. Suppose there are five tasks A, B, C, D, and E, whose topology is generated by TGFF as shown in Figure 4.

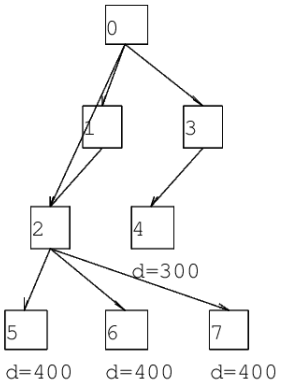


Figure 4. Task topology.

The data dependency matrix between tasks corresponding to the topology is as follows, assuming that the amount of data between tasks is 1:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

By analyzing the data dependency matrix and the task topology, it can be known that if the column vector corresponding to a subtask is a 0 vector, it indicates that there is no unconfigured predecessor task in the subtask, and the row vector of a particular task is a 0 vector, indicating that a subtask has no successor tasks, which signifies the end task of a large task. By decomposing the matrix, all possible task sequences can be obtained, and the decomposition process is as follows:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

There is no data dependency in the startup task that can be executed directly. First, the row and column vector corresponding to the start task is decomposed. From the decomposed row vector, it can be seen that the tasks of the subsequent positions of task A are B and C, and

the column vectors corresponding to tasks B and C become the 0 vectors after decomposing the row and column vectors of task A. Priority to remove the first column:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{5}$$

Priority to remove the second column:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{6}$$

The above corresponds to two decomposition methods, i.e., decomposing tasks B or task C. Here, we select the first corresponding decomposition method since there is no new 0 vector after the decomposition, so the next one to enter the configuration queue can only be task C:

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{7}$$

After decomposing the row and column vector of task C, the column vector of task D becomes a 0 vector. Next, we decompose task D:

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \tag{8}$$

At this point, there is only one task E left in the matrix. After E enters the configuration queue, all tasks enter the configuration state and an effective sequence is obtained through the decomposition of the matrix: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$. It is a safe and effective configuration method to generate task sequences in strict accordance with matrix decomposition.

3.3. Deployment of Tasks on Multi-FPGA Systems

The implementation of dividing the subtask sequence corresponding to several large tasks into the configuration sequence corresponding to several FPGA chips is called maintaining multi-sequences. We propose a Maintain Multi-Sequence (MMS) algorithm as follows (Algorithm 1).

First, we set the default device number for each subtask sequence in a circular fashion. This reduces the overhead of inter-device communication. In a resource-rich environment, subtasks belonging to the same task should be configured in a default device to reduce the amount of data communication between devices. On the other hand, load balancing is beneficial. By distributing large tasks across individual FPGA chips, we prevent multiple large tasks from being concentrated on a single device, resulting in the under-utilization of the system's computing and interface resources. When selecting the default device, first sort the subtask sequence by Deadline, which prevents the larger Deadline task sequence from defaulting to the same device. Secondly, sort the devices in descending order of resources. By allocating resources to a device with more resources, the subtasks in the same large task will be more centralized on the same device, reducing the overhead of communication between devices.

In the next step, a deadline-based as-late-as-possible configuration strategy is used to generate configuration sequences, where each task is configured at the last moment in a

timeframe that does not affect other tasks. Since the goal of the scheduling algorithm is to ensure that each task is completed before the deadline and that the same task is executed on different FPGAs in a dual-mode hot standby for fault tolerance, the time distance (DT) and resource usage between the current time and the deadline of each configuration sequence is first calculated. As shown in Figure 5, the process of requesting devices by subtasks uses back-to-front dynamic allocation. After comparing the deadlines, the largest deadline is used as the starting point for generating the configuration sequence.

Algorithm 1 Maintain Multi-sequences

Input: Task Collection, Device Collection

```
Sort the task sequence by Deadline and initialize the time
Sort devices by resource
for m in Task Collection do
    set the default number for every device
end for
while time- - do
    for tasks in Task Collection do
        if Separate successfully resources for tasks then
            Reduce the Deadline of tasks
        end if
    end for
    Reorder tasks
    if Allocation end then
        Break
    end if
end while{M}ove configuration sequence forward
if time < 0 then
    for mission in Device Collection do
        mission.time -= time;
    end for
end if
```

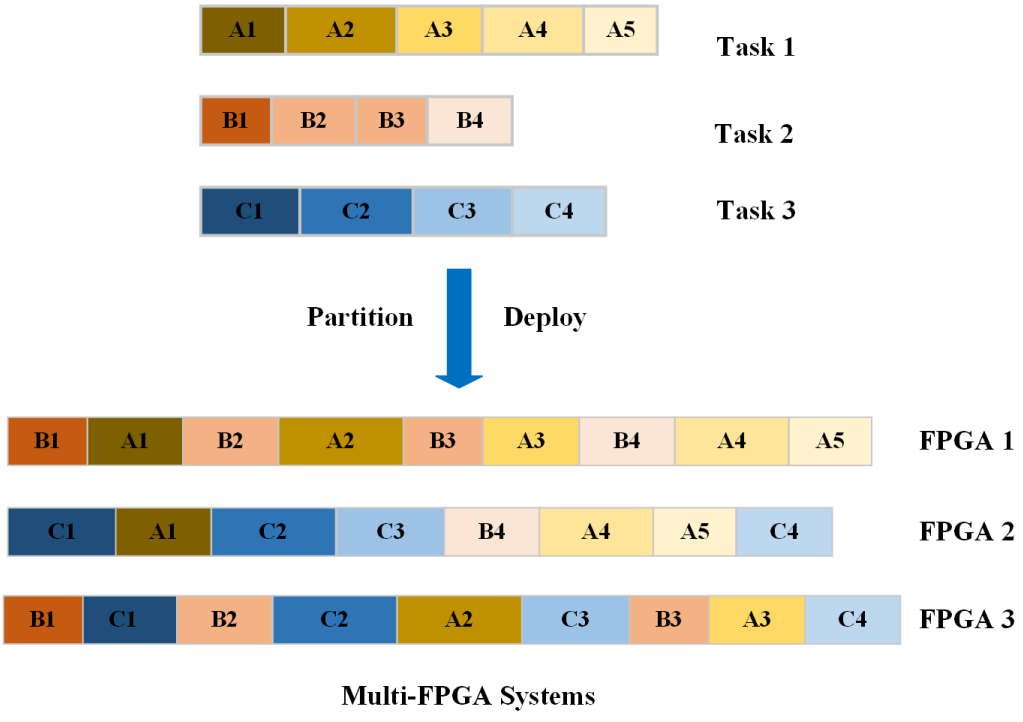


Figure 5. Task sequence partitioning and deployment on Multi-FPGA systems.

All subtasks form a Device Application Sequence and then enter the Device Configuration Sequence. The generation of the device application sequence starts with the last element of each task sequence, i.e., the end node in the task topology. The device application sequence is sorted in descending order of time distance. The task with the largest DT first selects the device and enters the device configuration sequence. After configuration, the next task in the same sequence is added to the Device Application Sequence and reordered. Iterating through the above rules dynamically generates the final configuration sequence. The last action before generating the final configuration sequence is to advance the configure sequence. Check that there is no downtime when all the tasks are scheduled; if there is, then the configuration time of all the tasks should be moved forward to eliminate the downtime.

4. Experimental Evaluation

In this section, we use the Task Graph for Free (TGFF) [21] to generate the Task DAG. The scheduler based on the Multi-FPGA system was tested to verify its performance and fault tolerance when scheduling multiple large tasks.

4.1. Testing Environment

We used C++ to build an experimental platform to provide the system framework and function preset interface for the proposed methods of task sorting, multiple sequence maintenance, resource management, and task combination, based on the algorithm proposed in this work and the comparison algorithm that is implemented. The test task set and device network structure adopt three task topology diagrams and two physical FPGA (XC7A200T-2FBG484l) to implement the scenario of multi-task scheduling on Multi-FPGA. The algorithm program is designed in C++ language and the FPGA project is designed in Verilog.

This work sets the common algorithms for target recognition and image processing as task topology generated by TGFF. The resource requirements of the tasks are shown in Table 1 [22]. The width and height of the task are generated by random numbers based on the amount of logical resources, in order to complete the simulation of the real shape of tasks.

Table 1. List of common algorithm resource requirements.

	Algorithm	Slices	BRAM
Target Recognition	Debayer (2×)	200	2
	Rectifier (2×)	500	30
	Stereo match	2500	30
	Disparity	1000	15
	Flex-SURF	1000	0
	Motor Control (3×)	200	0
Image Processing	FPN correction	100	0
	Dark field corr.	200	1
	FFT	800	7
	Bad pixel/spike	100	2
	CCSDS 122	2500	12
	Binning	300	4
	Hough Transform	1800	14
	Median Filter	800	0

4.2. Evaluation Metrics

In order to evaluate the performance of resource management algorithms, three evaluation metrics are introduced below.

Average Resource Vote (ARV): indicates the resource usage of the task and measures the effect of the scheduling algorithm to save resources. The expression is as follows:

$$ARV = \frac{1}{size_{total} * T_{total}} \sum_{i=0}^{N} (size_i * T_i) \quad (9)$$

Task Completion Rate (TCR): indicates the percentage of successfully executed tasks compared to all scheduled tasks.

Mean Time to Failure (MTTF): indicates the average time before failure. In systems where hard fault repair cannot be achieved, it is equivalent to the system lifetime.

4.3. Result and Analysis

We set up three sets of scheduling strategies for verification in three task DAGs. Regarding the first strategy, we do not use an additional task scheduling algorithm; we just let three large tasks enter the Single-FPGA one by one. In the second strategy, we use task hybrid scheduling to combine three large tasks into one large task set and apply for resources in the Single-FPGA. In the third experiment, we use a load balancing and fault tolerance strategy (resource- and reliability-aware scheduler) to schedule tasks and set default devices for large tasks, then mix subtasks scheduled in the same device and apply for resources in the Multi-FPGA. In the experiment, each device is set to have sufficient resources to respond to resource requests of subtasks.

Firstly, we evaluated the resource utilization and scheduling efficiency of the proposed method, and the experimental results are shown in Table 2 and Figure 6. In Table 2, the ARV, maximum resource utilization rate, and execution time of the three policies when scheduling three groups of tasks are shown. Among them, the Single-FPGA Task Continuous Placement strategy has no advantage in each of the metrics, while the Single-FPGA Task Mix Placement strategy has a higher ARV and maximum resource utilization rate than the Multi-FPGA Task Mix Placement strategy, but the execution time is much higher than the latter. Figure 6 shows the resource margin on each FPGA chip as the task group executes the three scheduling strategies. As a whole, it can be obtained that the task mix algorithm has a significant effect on improving the scheduling metrics.

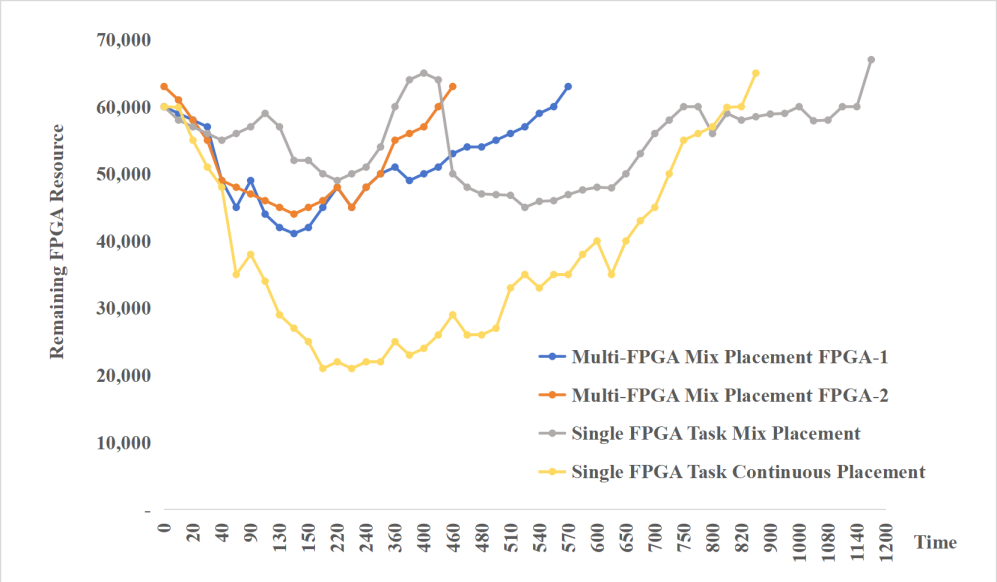


Figure 6. The resourcesurplus variation chart of the first group of experiments.

Table 2. The influence of the task hybrid algorithm and load balancing strategy on the scheduling effect.

Task Sequence Maintenance Approach	ARV			Maximum Resource Utilization Rate			Execution Time		
	1	2	3	1	2	3	1	2	3
Single-FPGA Task Continuous Placement	15.1%	16.2%	14.9%	34.4%	35.1%	31.1%	1141	1389	1312
Single-FPGA Task Mix Placement	42.6%	45.1%	43.0%	68.3%	74.9%	68.0%	858	1138	1140
Multi-FPGA Task Mix Placement	29.6%	30.4%	30.2%	34.3%	35.6%	34.9%	524	689	627

The above reasons are the task successive placement strategy causes the overall execution time of the tasks to become longer when dealing with the bursty task collection, and the much resources remaining indicates that FPGA's ability for parallel computing is not utilized to the extent that it is not possible to guarantee that the tasks that are leaning back will be executed before the deadline. When three tasks are mixed to apply for the resources of a single FPGA, the three tasks form a mixed task set, the overall execution time decreases, the resource utilization rises, and compared to sequential placement, the mixed placement configures more tasks in the FPGA, taking advantage of the FPGA's parallel nature. Three tasks are configured in two FPGAs through complex balancing and task mixing, which reduces the resource residual requirement for a single FPGA compared to mixing three tasks and reduces the total execution time by joint scheduling of the two FPGA devices, which can complete the task before the deadline.

Similarly, we use the above scenarios and three scheduling strategies to verify the reliability performance of the proposed method. The proposed method described in this paper also aims to extend the MTTF by load balancing to alleviate the stress accumulation on each FPGA device for a long time. Therefore, we used the method described in Section 3.3 of reference [23] to measure MTTF. The TCR and MTTF results are shown in Table 3. As shown in the table, the reliability-aware scheduling method proposed in this paper achieves an average of 98.3% on TCR and 15.7 years on MTTF, which is much higher than the other two scheduling methods.

The above reasons are due to: (1) simultaneous execution of redundant tasks through multiple FPGA placements ensures uninterrupted task execution after failures and (2) reduced spike stress on a single FPGA by a load-balanced mix of placement tasks on Multi-FPGA, resulting in extended system lifetime. The experimental results of the Single-FPGA task mix placement strategy are better than the Single-FPGA task continuous placement strategy due to the fact that the hybrid placement avoids fault accumulation and stress accumulation to some extent.

Table 3. The influence of the task hybrid algorithm and load balancing strategy on the scheduling effect.

Task Sequence Maintenance Approach	TCR			MTTF		
	1	2	3	1	2	3
Single-FPGA Task Continuous Placement	66.7%	62.5%	72.5%	5.2	6.0	5.8
Single-FPGA Task Mix Placement	85.0%	82.5%	90.0%	6.3	6.9	6.8
Multi-FPGA Task Mix Placement	100.0%	95.0%	100.0%	15.2	16.0	15.8

In addition, we further investigate the scalability of the scheduling strategy proposed in this paper on top of a multi-FPGA system and design the following experiments. By increasing the number of FPGAs to test the number of task groups that can be carried out by different scheduling strategies in a certain period of time, we have verified the scalability of the method. The scheduling strategies are Multi-FPGA Task Mix Placement, Single-FPGA

Task Continuous Placement, and Single-FPGA Task Mix Placement proposed in this paper. The task groups are then randomly combined from the tasks in Table 1.

From Table 4, it can be seen that as the number of FPGAs increases, the number of schedulable task groups increases for all scheduling policies, but the Multi-FPGA Task Mix Placement policy grows at a much higher rate than the other two policies. The reason is that the Single-FPGA Task Continuous Placement strategy can only schedule tasks sequentially and cannot execute tasks simultaneously on the chip. The Single-FPGA Task Mix Placement strategy can schedule different tasks to be executed on a single slice, so it can flexibly deploy tasks. The Multi-FPGA Task Mix Placement strategy cannot support more tasks when the number of FPGAs is small because it needs to deploy tasks redundantly, but with the increase of FPGAs, its scheduling strategy will be better able to support more tasks. Moreover, as the number of FPGAs increases, the scheduling strategy will make better use of the resources of each FPGA, and thus, the number of tasks completed can be significantly increased. The above experiments also demonstrate that the scheduling method described in this paper has good scalability on Multi-FPGA systems, which improves the adaptability of the system in large-scale edge computing scenarios.

Table 4. The number of task groups that can be scheduled with the increase of the amount of FPGAs under different scheduling strategies.

Task Sequence Maintenance Approach	The Amount of FPGAs				
	2	**3**	**4**	**5**	**6**
Single-FPGA Task Continuous Placement	4	6	9	13	16
Single-FPGA Task Mix Placement	10	14	19	27	38
Multi-FPGA Task Mix Placement	7	12	18	29	40

5. Conclusions and Future Work

In this paper, a set of task models is established based on the resource/time requirements, communication overhead, and state conversion process of the task; based on this, a resource- and reliability-aware hybrid scheduling method on the Multi-FPGA system has been proposed to fully utilize FPGA resources to cope with the deployment of bursty tasks. Experiments show that the method proposed in this paper significantly improves resource utilization by 7%, which achieves a task completion rate of 98.3% and extends system reliability with a mean time to failure of 15.7 years. In future work, we will design an exploration method to solve an appropriate proportion of FPGA usage in the system to balance reliability and makespan for different applications. In general, the more FPGAs are used as redundant resources to achieve fault tolerance, the longer the corresponding makespan. To ensure the main implementation goals of different applications, it is imperative to determine the right proportion of FPGA in the system for fault tolerance or operational tasks. In addition, we will further investigate the scalability and energy of scheduling strategies on Multi-FPGA architectures to more fully evaluate their applicability in large-scale edge computing environments.

Author Contributions: Conceptualization, Z.L. and Y.H.; methodology, Z.L. and H.G.; software, Y.H. and H.G.; validation, Z.L. and J.Z.; formal analysis, Y.H. and J.Z.; investigation, H.G. and J.Z.; resources, Z.L.; data curation, Y.H. and H.G.; writing—original draft preparation, Z.L. and Y.H.; writing—review and editing, Z.L. and J.Z.; funding acquisition, Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Research Start-up Fund in Shanxi Province under Grant 980020066, in part by the Management Fund of North University of China: 2310700037HX, and in part by the Fundamental Research Program of Shanxi Province: 202403021212165.

Data Availability Statement: Data are unavailable due to privacy or ethical restrictions.

Acknowledgments: The authors would like to thank the editors and reviewers for their contributions to our manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bolchini, C.; Sandionigi, C. Design of hardened embedded systems on multi-FPGA platforms. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2014**, *20*, 1–26. [CrossRef]
2. Shan, J.; Lazarescu, M.T.; Cortadella, J.; Lavagno, L.; Casu, M.R. CNN-on-AWS: Efficient allocation of multikernel applications on multi-FPGA platforms. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *40*, 301–314. [CrossRef]
3. Wang, Y.; Liao, Y.; Yang, J.; Wang, H.; Zhao, Y.; Zhang, C.; Xiao, B.; Xu, F.; Gao, Y.; Xu, M.; et al. An FPGA-based online reconfigurable CNN edge computing device for object detection. *Microelectron. J.* **2023**, *137*, 105805. [CrossRef]
4. Sun, Z.; Zhang, H.; Zhang, Z. Resource-Aware Task Scheduling and Placement in Multi-FPGA System. *IEEE Access* **2019**, *7*, 163851–163863. [CrossRef]
5. Wang, Z.; Tang, Q.; Guo, B.; Wei, J.B.; Wang, L. Resource Partitioning and Application Scheduling with Module Merging on Dynamically and Partially Reconfigurable FPGAs. *Electronics* **2020**, *9*, 1461. [CrossRef]
6. Najem, M.; Bollengier, T.; Le Lann, J.C.; Lagadec, L. A cost-effective approach for efficient time-sharing of reconfigurable architectures. In Proceedings of the IEEE 2017 International Conference on FPGA Reconfiguration for General-Purpose Computing (FPGA4GPC), Hamburg, Germany, 9–10 May 2017; pp. 7–12.
7. Iordache, A.; Pierre, G.; Sanders, P.; de F. Coutinho, J.G.; Stillwell, M. High performance in the cloud with FPGA groups. In Proceedings of the 9th International Conference on Utility and Cloud Computing, Shanghai, China, 6–9 December 2016; pp. 1–10.
8. Tianyang, L.; Fan, Z.; Wei, G.; Mingqian, S.; Li, C. A Survey: FPGA-Based Dynamic Scheduling of Hardware Tasks. *Chin. J. Electron.* **2021**, *30*, 991–1007. [CrossRef]
9. Ding, B.; Huang, J.; Xu, Q.; Wang, J.; Chen, S.; Kang, Y. Memory-aware Partitioning, Scheduling, and Floorplanning for Partially Dynamically Reconfigurable Systems. *ACM Trans. Des. Autom. Electron. Syst.* **2023**, *28*, 1–21. [CrossRef]
10. Ramezani, R. Dynamic Scheduling of Task Graphs in Multi-FPGA Systems Using Critical Path. *J. Supercomput.* **2021**, *77*, 597–618. [CrossRef]
11. Tang, Y.; Bergmann, N.W. A Hardware Scheduler Based on Task Queues for FPGA-based Embedded Real-time Systems. *IEEE Trans. Comput.* **2014**, *64*, 1254–1267. [CrossRef]
12. Deiana, E.A.; Rabozzi, M.; Cattaneo, R.; Santambrogio, M.D. A multiobjective reconfiguration-aware scheduler for FPGA-based heterogeneous architectures. In Proceedings of the IEEE 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Riviera Maya, Mexico, 7–9 December 2015; pp. 1–6.
13. Purgato, A.; Tantillo, D.; Rabozzi, M.; Sciuto, D.; Santambrogio, M.D. Resource-efficient scheduling for partially-reconfigurable FPGA-based systems. In Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, USA, 23–27 May 2016; pp. 189–197.
14. Clemente, J.A.; Resano, J.; Mozos, D. An approach to manage reconfigurations and reduce area cost in hard real-time reconfigurable systems. *ACM Trans. Embed. Comput. Syst. (TECS)* **2014**, *13*, 90. [CrossRef]
15. Wang, G.; Liu, S.; Nie, J.; Wang, F.; Arslan, T. An online task placement algorithm based on maximum empty rectangles in dynamic partial reconfigurable systems. In Proceedings of the IEEE 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Pasadena, CA, USA, 24–27 July 2017; pp. 180–185.
16. Kao, C.C. Performance-Oriented Partitioning for Task Scheduling of Parallel Reconfigurable Architectures. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 858–867. [CrossRef]
17. Resano, J.; Mozos, D.; Catthoor, F. A Hybrid Prefetch Scheduling Heuristic to Minimize at Run-time the Reconfiguration Overhead of Dynamically Reconfigurable Hardware. In Proceedings of the IEEE Design, Automation and Test in Europe, Munich, Germany, 7–11 March 2005; pp. 106–111.
18. Liu, S.; Forrest, J.; Vallee, R. Emergence and development of grey systems theory. *Kybernetes* **2009**, *38*, 1246–1256. [CrossRef]
19. Wu, J.O.; Wang, S.F.; Fan, Y.H.; Chien, W. The scheduling and placement strategies for FPGA dynamic reconfigurable system. In Proceedings of the IEEE 2016 International Conference on Applied System Innovation (ICASI), Okinawa, Japan, 26–30 May 2016; pp. 1–2.
20. Wassi, G.; Benkhelifa, M.E.A.; Lawday, G.; Verdier, F.; Garcia, S. Multi-shape tasks scheduling for online multitasking on FPGAs. In Proceedings of the IEEE 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, France, 26–28 May 2014; pp. 1–7.
21. Dick, R.P.; Rhodes, D.L.; Wolf, W. TGFF: Task graphs for free. In Proceedings of the IEEE Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE'98), Seattle, WA, USA, 15–18 March 1998; pp. 97–101.

22. Zhang, H.; Bauer, L.; Kochte, M.A.; Schneider, E.; Wunderlich, H.J.; Henkel, J. Aging resilience and fault tolerance in runtime reconfigurable architectures. *IEEE Trans. Comput.* **2016**, *66*, 957–970. [CrossRef]
23. Li, Z.; Huang, Z.; Wang, Q.; Wang, J. AMROFloor: An Efficient Aging Mitigation and Resource Optimization Floorplanner for Virtual Coarse-Grained Runtime Reconfigurable FPGAs. *Electronics* **2022**, *11*, 273. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.