A Multi-Tenant Resource Management System for Multi-FPGA Systems

Miho YAMAKURA††, Nonmember, Ryousei TAKANO†††, Akram BEN AHMED†††, Midori SUGAYA†††, Members, and Hideharu AMANO††, Fellow

SUMMARY FPGA (Field Programmable Gate Array) based accelerators are attracting significant interest in cloud computing systems. Combining multi-FPGA systems with cloud computing brings a new perspective to the reconfigurable computing research. However, the multi-tenancy of a multi-FPGA system has not been fully discussed in the previous researches. In this paper, we propose a multi-tenant resource management system, named FiC-RM, for a multi-FPGA cloud system. FiC-RM provides users with a set of FPGA resources according to their requirements and allows them to exclusively access FPGA boards and the interconnection network. To achieve this, we propose a placement algorithm which is a key to efficiently share the limited resources. We demonstrate FiC-RM controls a practical scale multi-FPGA system. Moreover, our simulation study shows that our placement algorithm achieved 3 to 4% improvement in the average resource usage and a 20-second reduction in the response time, compared to other existing naive algorithms.

key words: multi-FPGA, FPGA, multi-tenant

1. Introduction

FPGAs (Field Programmable Gate Arrays) are used in various fields that require large computational power with a low energy budget, such as machine learning. However, many high-performance FPGAs are expensive, and the number of users who can purchase and use them individually is limited. Therefore, various organizations and institutions are conducting research and development to enable the use of high-performance FPGAs in the cloud so that many users can easily use them [1], [2]. In such environments, multiple users must be able to use FPGA resources at the same time. In other words, it is necessary to build a system that operates with multiple tenants. Virtualization is a popular approach to realize multi-tenancy in cloud computing. Various methods have been proposed for FPGA virtualization [3], [4], but there is no established method unlike the computer virtualization technology used in the cloud.

Especially in large-scale cloud computing in which the enormous amount of resources available, it is important to enable multiple users to use large amounts of FPGA resources simultaneously. In the cases of arranging multiple tenants on a single FPGA such as [3], ensuring the independence of each tenant has become an issue, and various methods have been proposed. On the other hand, in a multi-FPGA system, independence can be easily realized because it is possible to provide different FPGAs for each user without sharing a single FPGA. However, to enable one user to use multiple FPGAs, a high-speed and secure inter-board network is essential. Therefore, to realize multi-tenancy in a multi-FPGA system, a system that manages both FPGA resources and communication between them is required.

In this research, we realize multi-tenancy in a multi-FPGA system that we have developed, Flow-in-Cloud (FiC). FiC is a multi-FPGA system in which multiple mid-range FPGAs are directly connected with a circuit switching network. The STDM (Static Time Division Multiplexing) method [5] is adopted in this network and the communication paths are set for each slot. The advantage of the FiC system can be summarized as follows. First, since FiC is composed of many FPGAs, users can use large FPGA resources overall. Second, since the FPGAs are directly connected without the host PC, it is possible to improve the resource utilization rate and scalability of the entire system compared to the system in which the host PC is connected to each FPGA. Multi-FPGA systems like FiC has received a lot of attention as a server for Multi-access Edge Computing (MEC). MEC is being standardized at ETSI (European Telecommunications Standards Institute) as a standard of 5G mobile network. It provides an IT service environment and cloud-computing capabilities at the edge of an access network which contains one of more type access technology, and in close proximity to its users [6]. Edge devices can offload heavy tasks to MEC, and they can be executed without suffering the delay and network congestion on uplink to the cloud. Thus, timing critical services: factory control, traffic management of a smart city, and security control are expected.

Multi-FPGA computing is advantageous for MEC, because it can accept the requests from IoT devices directly, and execute timing critical jobs with hard-wired logic. Especially, network with the STDM switches contributes to keep the bandwidth and delay time within a certain limitation for timing critical jobs. Moreover, the consumed energy is enough small to provide base stations on the roof of buildings. For such usage, a system-wide resource management
system to support multi-tenancy is essential.

In FiC, there are four challenges that must be solved in order to realize multi-tenancy: (i) exclusive board assignment, (ii) separation of communications, (iii) constrained and efficient board assignment, and (iv) management of slots. To solve these challenges and realize multi-tenancy in FiC, we have developed FiC-Resource Manager (FiC-RM). The main functions of FiC-RM are as follows.

- Automation of board assignment: FiC-RM determines the board assignment so that multiple users can efficiently share FPGA resources.
- Abstraction of the board and slot ID: The users of FiC-RM do not need to be aware of which board or slot they are using.

With these functions, FiC-RM enables multiple tenants to operate simultaneously in FiC. There are a lot of systems for introducing virtualization to FPGAs in a cloud computing [7]. FiC-RM targets bare-metal style stand-alone multi-FPGA systems unlike other systems in which each node of the cloud provides FPGAs as component. FiC-RM focuses on supporting multiple users with a group of FPGA boards connected with the STDM switch. Timing critical jobs required for MEC can be done with the group once it is assigned and the job is launched. On the contrary, FiC-RM can only support a limited shape of FPGA groups exclusively separated from each other. Also, resources on a single FPGA cannot be shared with multiple users unlike most FPGA virtualization systems.

The organization of this paper is as follows. Section 2 gives an overview of the FiC system, and Sect. 3 describes the four challenges to achieve multi-tenancy in FiC. Section 4 describes FiC-RM. Then, the evaluation result is shown in Sect. 5, and the conclusion is stated in Sect. 6.

2. Flow-in-Cloud (FiC)

2.1 Overview of FiC

FiC consists of multiple mid-range FPGA boards called FiC-SW tightly connected by a number of high-speed GTH serial links as shown in Fig. 1.

A FiC-SW consists of an FPGA, two 16 GB DRAM cards, high-speed serial links, and a Raspberry Pi3 daughter-board. The FPGA mounted on the FiC-SW is either Xilinx Kintex UltraScale XCKU095 or XCKU115, and we call the board with XCKU095 Mark1 board, while the board with XCKU115 Mark2. The current FiC prototype has 12 Mark1 boards and 12 Mark2 boards, 24 boards in total. Raspberry Pi3 is connected to the FPGA with GPIO and the host server via Ethernet to manage the configuration and I/O for the FPGA. For interconnection, 32 serial links, each of which provides 9.9 Gbps, are connected to each board.

2.2 The STDM Switch

In order to cope with timing-critical jobs for MEC, FiC adopts a circuit-switched network with STDM (Static Time Division Multiplexing) switches. In the STDM method, each link is time-divided into multiple slots. Figure 2 shows the outline of the STDM switch. In this example, the number of ports and slots are both three. The destination of the data coming into each port are treated depending on the slot. For example, the data that enters port-2 is once stored in the FIFO IP and then transferred to port-0 in slot-0. In slot-1, the data from port-1 is sent to port-1. In slot-2, the data from port-2 is broadcasted to all output ports.

The port to which data are transferred in each slot is determined by a routing table provided to each output port of the switch. By referring the table with the current slot number, the output port selects the transferred input port data. Since the slot number is incremented every two clock cycles, the input data and output data are time multiplexed every two clock cycles. The duration of the slot is depending on the circuit switching network required for the application. We use the slot optimization algorithm [8] to reduce the number of duration as much as possible, since large duration degrades the network performance. Since time slot is statically allocated, the back pressure is not needed for the board-to-board communication. When computational module connected to the switch cannot provide enough buffer to receive the data, the back pressure mechanism is required between end-to-end communicating computational
modules. The user can establish the necessary communication paths by setting the routing table appropriately before the job starts. The table is assumed to be static until the executing job is finished, although dynamic change is possible if all related boards are synchronized.

The current FiC system uses the FireFly cables as high-speed serial links. A FireFly cable has four bi-directional lanes, and the bandwidth of each lane is 9.9 Gbps. The STDM switch operates at 100 MHz, the same clock frequency as the PR region described later, and can transfer a 170-bit data every two clock cycles. Therefore, the maximum effective bandwidth per lane is 8.5 Gbps. About 60 clock cycle latency is required for transferring data from a board to another board. From the duration of the slot, the programmer can estimate the latency between a source board to the destination board if needed. The detail of the performance of STDM switch is described in [5], [9].

2.3 Partial Reconfiguration on FiC Board

Figure 3 shows a diagram of the different modules embedded in the FPGA of the FiC board. FPGAs on FiC boards can be roughly divided into two areas. One is a static region that contains the STDM switch and other IP modules (Xilinx AURORA) used for the communication between boards. The static region is always in operation without any changes to maintain the platform. The other is a partial reconfiguration (PR) region where replaceable application modules are implemented. By adopting this implementation, it is possible to dynamically reconfigure multiple applications while maintaining constant communication between boards. Since FiC uses a circuit-switching network, if one FPGA is reconfigured and modules, such as switches responsible for communication, are lost from the path, applications on other boards will also stop. Therefore, the static region needs to continue working during reconfiguration.

2.4 The Management System for FiC

Figure 4 depicts an overview of the fundamental management system for FiC. Currently, the management system for FiC consists of three main components. The first is ficwww [10], which runs on each FiC board as the basis of the management system. The FiC-RM developed in this research manages FiC boards using ficwww. Figure 5 gives an overview of ficwww. By using ficwww, users can perform operations such as FPGA configuration through API without logging in directly to the Raspberry Pi3 board mounted on each FiC board. The flow for using ficwww is briefly explained below. First, the user describes the information necessary to realize the desired operation in a JSON file. For example, if the desired operation is the configuration to the FPGA, the necessary information is the data that the user wants to send. Using this API, the user posts this JSON file to the web-server running on the Raspberry Pi3 board, and then it performs the operation desired by the user.

The second is a tool called ficmgr that makes it possible to perform operations such as configuration for multiple FPGAs at once using ficwww. Ficmgr runs on the control server or the user’s machine. The third is a debugger that enables remote verification tasks. This allows users to debug through the XVC (Xilinx Virtual Cable) server on the Raspberry Pi3 board without directly accessing the FiC board. Since none of these three holds the resource usage of the entire system, the role they play in FiC is different from that of FiC-RM, which manages resources based on usage.

3. Challenges for Realizing Multi-Tenancy in FiC

Currently, FiC cannot be used in multi-tenancy because there are four major challenges. In this section, we explain about these challenges that need to be solved to realize multi-tenancy in FiC.
boards. By using ficwww and ficmgr, users can perform various operations on the FPGA without directly accessing target FiC boards. However, conflicts in the use of FPGA boards are unavoidable at present because the board assignment is manually done, and there is no system-wide resource assignment mechanism. Therefore, when more than one user want to use FiC at the same time, it is necessary to manually check the board usage to avoid collision of using the same board. This problem can be solved by automating board assignment according to each user’s request and board usage.

### 3.2 Separation of Communications

The second challenge is the separation of communications between FPGA boards. This is achieved by setting the shape of the FPGAs assigned to the user to be rectangular. By doing so, each user can guarantee the necessary communication paths are inside the rectangle. This is because the FiC boards are connected with 2D torus, as shown in Fig. 6. In FiC-RM, the routing table of each board is set so that all communications can be performed without going through anything other than the board assignment to the user. Actually, FiC allows more flexible network settings. However, if communication is allowed via boards not assigned to the user, the number of slots in STDM switches will increase. This is because different slots must be assigned to each user’s communication path to achieve the separation of communications. As the number of slots increases, the efficiency of communication decreases. Therefore, in FiC-RM, the number of slots is suppressed and the decrease in communication efficiency is prevented by determining the board assignment so that the communications can be completed within the boards assigned to the user. It also keeps to guarantee network bandwidth and latency in application execution, that is the benefit of circuit switching network with the STDM switches.

Unlike the task assignment in parallel processors, migrating takes large overhead in FPGAs. Although some researches address the migration of FPGAs [11], [12], it requires to download the states of registers and memory contents, upload the configuration data and such memory contents, and restart the task. Even for a single FPGA, it takes long time, and the migration of a multi-FPGA group will take more. Thus, the de-fragmentation using task migration is not realistic in our system, and we should develop an algorithm to suppress the fragmentation as much as possible.

### 3.3 Constrained and Efficient Board Assignment

The third challenge is to achieve constrained and efficient board assignment. When FiC-RM automatically determines the board assignment, as mentioned above, the FPGAs assigned to each user must be arranged in a rectangular shape. FiC-RM uses the algorithm shown in [13] to find rectangular free spaces. And, it assigns the requested number of boards in one of the rectangles to the user. Most of the related research on rectangular task placement on FPGA such as [13] deals with task placement for a single chip. On the other hand, in this research, we deal with the arrangement of resource requests from each user of FiC, which is an interconnected multi-FPGA system. When considering task placement in FiC, what is different from the case of a single chip is the flexibility of the rectangle type. In FiC, if the number of boards required by the user is guaranteed and the communication path is set correctly, there is no restriction on the type of rectangle (for example, 1×4 rectangle or 2×2 rectangle). This is because in FiC, all boards, which are the minimum units of assignment, are uniform. If the sizes of the rectangles are the same, the amount of resources contained in them are equal. On the other hand, in the case of a single chip such as [13], the type of rectangle is defined at the time of the user’s request. This is because the amount of each resource contained in each CLB (Configurable Logic Block), which is the minimum unit of assignment, is not uniform. Different types of rectangles contain different amounts of resources. To maintain flexibility, we need other criteria to decide which type of rectangle to choose. FiC-RM selects the rectangle using the algorithm described in Sect. 4.2.1.

### 3.4 Management of Slots

The fourth and final challenge is the management of slots used by each user. In FiC-RM, to improve the communication performance, routing tables that minimize the number of slots are generated and set to the switches on each board. However, when deciding the assignment of the slots to each communication path on the FiC-RM side, the user must rewrite the application program every time the slot to be used changes. FiC-RM abstracts slot IDs using conversion tables so that users can program without being aware of the slots to be assigned. This abstraction is essential for using FiC with multiple tenants including IoT devices in MEC, since the required jobs must be executed on groups of boards as much as possible.
4. Multi-Tenant Resource Management System FiC-RM

4.1 Overview of FiC-RM

FiC-RM solves the issues shown in Sect. 3 by introducing a mechanism for assigning resources based on the availability of boards and managing slots used by each user. First, we define the terms for the following explanation.

- slice: It is an execution environment isolated for each user, and consists of nodes and flows that connect them. FiC-RM reserves, monitors, and releases slices, and loads and starts applications based on user requests. Each slice is assigned a unique ID (slice ID) in the system.

- node: It is a computing resource, that is a FiC board in FiC system. Each node has a unique ID in the system (physical ID), and FiC-RM assigns a unique ID in the slice to each node (logical ID).

- flow: It is a virtual communication path within the slice. Each flow is assigned a unique ID (flow ID) within the slice. Application modules use the flow ID to specify the packet destination.

Users describe the information required for the operation they want to perform, and send the JSON file format request to FiC-RM running on the control server. The target of the request is all the boards included in the slice or a specific board in the slice, depending on the type of API. Then, FiC-RM operates the user request on each target FiC board using ficwww shown in Sect. 2.4. Table 1 shows a list of FiC-RM APIs. With the operations shown in this table, slice creation is described in details in Sect. 4.4 as an example.

The benefits of the user are, firstly, to concentrate on higher degree of flow of the development without considering the manual configuration of a specific FPGA board. Secondly, it is possible to obtain an FPGA resource according to the required FPGA processing request. These fine-grain board assignments for satisfying their demands are achieved by the proposed FiC-RM.

4.2 Board Assignment

FiC-RM automates the board assignment to tackle the issue presented in Sect. 3.1. The method for board assignment used in FiC-RM is as follows.

4.2.1 Board Placement

FiC-RM processes the user requests in a FIFO manner. For the request at the beginning of the task queue, FiC-RM checks whether the requested number of boards can be reserved. If boards can be reserved, it performs the slice creation operation according to the procedure presented in Sect. 4.4. If boards can not be reserved, the request is stalled until another slice is freed. In this fashion, no request can be overtaken by a following one.

In FiC-RM, as described in Sects. 3.2 and 3.3, the board assignment is decided so that the shape of each slice is rectangular. Therefore, it is necessary to find a rectangular free space. FiC-RM finds a rectangular free space using the Scan Line Algorithm (SLA) proposed in [13]. By using SLA, Maximum Empty Rectangles (MER) can be found. MER is an empty rectangle that can not be entirely covered by any other rectangle. For example, two MERs are depicted in Fig. 7.

Algorithm 1 shows the method of the board assignment after finding MER. FiC-RM can find MERs larger than the user request. Then, in an order from the smallest MER, they are checked to see if the requested slice can be assigned in the form of a rectangle. If it can be assigned, the slice is assigned in the MER. If multiple MERs have the same size, or if the number of boards included in the MER is greater than the user request, multiple assignment patterns are possible. In this case, FiC-RM selects the assignment pattern that maximizes the size of MER remaining after the assignment.

4.2.2 Node ID Abstraction

FiC-RM uses two types of node IDs shown in Sect. 4.1 and maintains their correspondence. At the time the user creates the application modules, they do not know on which board

| Path                | Method | Operation                                      |
|---------------------|--------|-----------------------------------------------|
| /slice              | POST   | Creating a slice                              |
| /slice/<slice-id>/start | PUT    | Starting the HLS modules for all nodes in the slice |
| /slice/<slice-id>/status | GET    | Getting information about the slice            |
| /slice/<slice-id>/<node-id> | PUT   | Sending data to the HLS module on a specific node |
| /slice/<slice-id>/<node-id> | GET   | Receiving data from the HLS module on a specific node |
| /slice/<slice-id>   | DELETE | Releasing the slice                           |

Table 1 FiC-RM REST API list for users.
Algorithm 1 Algorithm for board assignment in FiC-RM

Input: MER list \( M \) and the number of boards to be assigned \( \text{boards} \)
Output: Board IDs to be assigned \( \text{Assigned} \)

1: Begin BoardAssignment
2: Sort \( M \) in ascending order of rectangle size;
3: \( \text{remaining size} \leftarrow 0; \)
4: for \( i = 0 \) to \( M \).Length – 1 do
5: if \( (M[i].size < \text{boards}) \) then
6: continue
7: else if \( (M[i - 1] \geq \text{boards} \) and \( M[i - 1].size < M[i].size \)) then
8: break
9: else
10: for All possible rectangle types for \( M[i] \) do
11: for All possible placement patterns for the rectangle in \( M[i] \) do
12: \( \text{tempsize} \leftarrow \) the largest remaining rectangle size after the assignment with the placement pattern;
13: if \( \text{tempsize} > \text{remaining size} \) then
14: Update \( \text{Assigned} \) and \( \text{remaining size} \);
15: end if
16: end for
17: end for
18: end if
19: end for
20: return \( \text{Assigned} \)
21: End BoardAssignment

An example of using these two types of IDs is represented in Fig. 8. In this example, four FiC boards are assigned to user-1 and user-2 two by two. Board-0 and 1 (physical node ID) are included in slice-1 of user-1. They are assigned logical node ID 0 and 1 in the slice. Board-2 and 3 (physical node ID) are included in slice-2 of user-2. They are also assigned logical node ID 0 and 1 in the slice. For example, suppose that user-2 requests FiC-RM to send data to the board with ID 0 in the assigned slice ((1) in Fig. 8). FiC-RM derives that logical node ID 0 in slice-2 means physical node ID 2 based on the correspondence list ((2) in Fig. 8). Then, the operation requested by the user is performed on the board with physical node ID 2 ((3) in Fig. 8). With such a mechanism, users can perform various operations using only logical node ID. That is, they do not need to be aware of which physical board they are using.

4.3 Slot ID Abstraction

The solution to this challenge, previously explained in Sect. 3.4, is shown hereafter. In FiC, users specify the packet destination by slot ID. The packet is forwarded according to the routing tables of STDM switches to reach the application module on the target node. Currently, the physical node ID is often used as a slot ID directly. This is because many applications perform all-to-all broadcasting, and in that case, each communication path can be easily separated by using the physical node ID. However, this method uses a large number of slots and is inefficient except for broadcasting. FiC-RM assigns slots to each flow in a slice, and thus, abstraction of both slot id and board id is essential. The tool developed in [8] is used to determine which slot to be assigned to each flow. This tool generates routing tables that minimize the number of slots used in each slice. However, when slot assignment is automated, the user needs to rewrite the application program every time the slot to be used changes. To eliminate such changes on the user side, in this research, we implement tables that convert the flow ID and slot ID to each other.

Two conversion tables are implemented on each FiC board, as shown in Fig. 9. One is a PV (Physical-Virtual) conversion table that converts the destination field of packets from slot ID to flow ID and passes it to the HLS module. The other is a VP (Virtual-Physical) conversion table that converts the destination field of packets from flow ID to slot ID. Figure 9 shows an example of two conversion tables when slot-2 is assigned to flow-0. 0 is set at address 2 of the PV conversion table, and 2 is set at address 0 of the VP conversion table. Since FiC-RM sets the conversion table appropriately in this way, users can create applications without being aware of the slot actually used.

4.4 Slice Creation

Figure 10 briefly shows the flow of creating a slice. First, the user describes the specifications of the slice in a JSON file ((1) in Fig. 10). Figure 11 shows an example of a JSON
file for the slice definition in FiC-RM. In “ficapp_list”, the configuration data to be deployed to each node is defined. Here, “node_id” means logical node ID of each node in the slice. “flow_graph” defines a set of flows for this slice. In this example, two flows are defined: node-0 to node-1, and node-1 to node-0. “flow_id” means the flow ID of each flow.

When the user posts the JSON file described to the specified URL, FiC-RM receives it and decides the board to be assigned in the way shown in Sect. 4.2.1. Then, as shown in Sect. 4.2.2, the correspondence between the physical node ID and the logical node ID is defined. Next, using the tool developed in [8], slot assignments for each flow are determined and routing tables are generated ((2) in Fig. 10). Then, FiC-RM configures the data and sets the routing table to each board ((3) in Fig. 10). It also sets the conversion table based on the correspondence between the slot ID and the flow ID. When these processes are completed and the slice is successfully reserved, FiC-RM returns the slice ID of the created slice to the user ((4) in Fig. 10). When the user releases the slice, the list showing board usage is updated. By doing so, the board contained in the released slice can be assigned to a new user thereafter.

5. Evaluation

Hereafter, the evaluation environment is briefly described. As the operating environment, a FiC system consisting of 12 MARK-1 boards and a control server was used. In reality, FiC consists of 24 boards, as shown in Fig. 6, and there are two types of boards: MARK-1 and MARK-2. FiC-RM does not create slices comprised of different types of FiC boards. Therefore, we use only MARK-1 boards in this evaluation. The FPGA on the MARK-1 board is Xilinx Kintex UltraScale XCKU095-FFVB2104. The PR area and STDMM switch on each FPGA, shown in Fig. 3, operate at the same clock of 100 MHz. FiC-RM and NFS (Network File System) server run on the control server. And, it shares storage with configuration files with the Raspberry Pi3 board on each FiC board.

5.1 Resource Overhead of Conversion Table

We firstly evaluate the basic overhead of the conversion tables processing on MARK-1 boards. Table 2 shows the resource overhead of the FPGA on MARK-1 board to implement the conversion table described in Sect. 4.3. Looking at this table, the percentage of overhead to total available resources is about 0.1% for LUT (lookup table) and about 0.3 to 0.4% for LUTRAM (distributed RAM), FF (flip-flop), and BRAM (block RAM). Since the overhead is negligible, the implementation of the conversion table has almost no effect on the implementation of applications.

5.2 Elapsed Time for a Slice Reservation

To compare the total time required to reserve the slice with different numbers of FGPGAs, we set several patterns to evaluate with slices consisting of a maximum of 12 boards. The results are shown in Fig. 12. The blue part shows the time required to configure the FPGA boards, and the orange part represents the time required to determine the board assignment explained in Sect. 4.2.1. From this graph, it can be seen that even when the number of boards included increases, the increase in the time required to reserve the slice is reduced. For example, when comparing the case where the number of boards is 2 and the case where the number of boards is

| Table 2 Resource overhead of conversion table. |
|-----------------------------------------------|
|      | Overhead | Available (mark 1 board) | Percentage |
|------|----------|--------------------------|------------|
| LUT  | 679      | 537600                   | 0.13%      |
| LUTRAM | 286     | 76800                    | 0.37%      |
| FF   | 3398     | 1075200                  | 0.32%      |
| BRAM | 5.5      | 1680                     | 0.33%      |
12, the time increase is 5 seconds or less even though the number of boards becomes 6 times. This is because the configuration, which takes most of the time, is done in parallel for each board. However, although the configurations are done in parallel, the time required for configuration increases slightly as the number of boards increases. We consider that the network between the control server and each board becomes the bottleneck. On the other hand, the time required to determine the board assignment decreases as the number of boards increases. This is because the processing time decreases in proportion to the decrease in the number of possible assignment patterns.

5.3 Board Assignment Algorithm

A simulation was performed to investigate the usefulness of the board assignment algorithm in FiC-RM described in Sect. 4.2.1. In this simulation, the algorithm used in FiC-RM was compared with the other two board assignment algorithms. However, no matter which algorithm was used to determine which board to assign, the algorithm used to find MERs was SLA. The outline of the three algorithms, including the proposed one, is as follows.

- **worst-fit**: The slice is placed in the largest of the existing MERs.
- **first-fit**: Each MER is checked in the order in which they were found, and the slice is placed as soon as the one in which the slice can be placed is found. In [13], simulations were performed using this algorithm.
- **proposed**: The smallest MER in which the slice can be placed is selected. If there are multiple MERs of the same size, the one with the larger MER remaining after placement is selected. Details were previously shown in algorithm 1.

Experiments in which 100 requests were posted were conducted five times for each algorithm. The board usage rate was recorded at the timing when the board usage status changed, 500 times each for creating slices and releasing slices, that is 1000 times in total. Then, the average board usage of 1000 times was calculated for each algorithm. We also measured the average response time of 500 requests (the time between posting a request and receiving a response). The maximum value of the arrival time was determined, and a uniformly distributed random number was generated between that maximum value and 0 as the arrival time of each request. The execution time of each request (the time from the completion of slice creation to the release) was defined by generating a random number with a normal distribution (mean = 5, standard deviation = 2). The number of boards is 24 boards (4x6). As previously mentioned, there are only 12 boards (3x4) for MARK-1 and the minimum unit of assignment is 1 board. However, we conducted the simulation with 24 boards for future extension of the system. The number of boards included in each requested slice was defined by generating a random integer with a maximum value of 12.

The results are shown in Figs. 13 and 14. Figure 13 shows the average board usage for each case where the request arrival frequency is different. The horizontal axis shows the minimum and maximum values of the arrival time of each request, and the vertical axis shows the average of the board usage. On the horizontal axis, the more to the left, the narrower the request arrival interval, and the more frequent requests occur. Looking at this graph, it can be said that the proposed algorithm has a higher board usage than the other two algorithms at any arrival time interval, and efficient board assignment was performed.

Figure 14 shows the average response time for each arrival time interval. Looking at this graph, it can be said that the proposed algorithm had a smaller response time than the other two algorithms at any arrival time interval. The difference from other algorithms is significant, especially when
requests arrived frequently in both Figs. 13 and 14. For example, when the arrival time was 0 to 300 seconds, the proposed algorithm improved the average resource usage by about 3% and the response time by about 20 seconds compared to the others. This is because the narrower the request arrival interval, the more often the boards run out and cannot be assigned, and the influence of the assignment algorithm increases.

6. Related Work

6.1 FPGA Virtualization

FPGA virtualization in cloud computing has been widely researched [7]. Although most of them are to flexibly and safely support FPGA resources in the cloud for users like other computational resources in the cloud [1], [14]. However, our research goal is supporting multi-tenant resource management for bare-metal style stand-alone multi-FPGA systems assuming to be provided for MEC. Our superiority is to support using a group of FPGA boards connected with the STD switch (slice) which can execute timing critical jobs. On the other hand, although a slice is a kind of virtualization, FiC-RM can only treat a limited shape of slices and resource sharing in a slice is not supported. The difference comes from the different targets and goals.

Commercial cloud services called FaaS (FPGA as a Service) is beginning to spread. In the existing FaaS cloud, the FPGA card is often connected to the host PC via the PCIe bus. In such a system, the user can exclusively use FPGAs through the host PC or virtual machine. A typical example of FaaS cloud is Amazon EC2 F1 Instances [2].

FiC-RM should be classified into the multi-node level virtualization in the survey paper [7]. LEAP [15] focuses on the virtualization of links between FPGAs and supports flexible virtualization, but it cannot be used for FiC providing the STD switch network. Although Melia [16] also supports node level virtualization for FPGAs in the cloud, the application is limited for MapReduce. Other resource management systems [17], [18] tends to focus on a single application rather than multi-tenant.

6.2 FPGA Systems with Multiple Users

Khawaja et al. [3] proposes AmorphOS, a system that encapsulates FPGA logic and provides it to each task. In this system, the same FPGA is shared by multiple tasks while maintaining independence by encapsulation. That is, it allows multiple tenants to be placed on a single FPGA chip. This research focuses on resource management inside a single FPGA chip, and unlike FiC-RM, it does not handle multiple FPGAs.

As a method for developing a multi-tenant FPGA system, Istvan et al. [4] proposes a system for sharing application modules on an FPGA between multiple users. By using this system, it is possible to dynamically change the allocation of communication and computational resources for each user. The system aims to efficiently assign FPGAs to tasks under conditions where the number of application modules is limited. Therefore, it is not assumed the case where sharing is difficult or frequent rewriting is required since there is a wide variety of application modules. On the other hand, in FiC-RM, each user configures the FPGA boards every time the required application changes, so there is no problem even if each user uses a completely different type of application.

6.3 Finding a Space on FPGAs

There are two types of algorithms for finding a rectangular free space on the surface of an FPGA. The first one is finding overlapping rectangles like MER. In [19], the staircase algorithm [20] is introduced as an algorithm to search for MERs on a single FPGA. SLA [13] used in FiC-RM is also an algorithm for finding MER. In FiC-RM, we adopted SLA instead of staircase algorithm because it was shown in [13] that SLA is faster than staircase algorithm.

The second algorithm is finding non-overlapping rectangles. In [21], the basic algorithm for finding non-overlapping rectangles and using them to place tasks is proposed. In [22], the algorithm shown in [21] has been improved to raise the efficiency of task assignment. Moreover, in [22], the time required to find the space where the task can be assigned is shortened by using a hash array as the data structure for managing empty rectangles. MER is the largest free rectangular space and allows partial overlap, whereas non-overlapping rectangles do not. Therefore, non-overlapping rectangles are not the largest free rectangular space, so tasks may be refused even if there are enough available resources. FiC-RM aims to efficiently assign resources to each user. Therefore, considering the disadvantage of non-overlapping rectangles that there is a possibility of making a mistake in determining whether or not to assign, we adopted an algorithm that searches for MER instead of non-overlapping rectangles.

7. Conclusion

FPGAs are now being used in a variety of applications in cloud computing. Multi-tenancy is necessary considering that multiple users can simultaneously use them without intervention. In this paper, we have developed FiC-RM, which is a resource management system that realizes multi-tenancy on FiC system in which multiple FPGA boards are directly connected through high-speed serial links. By using FiC-RM, users can use FPGA resources without interfering with other users and being aware of which boards and slots they are actually using. We also confirmed that FiC-RM efficiently allocates multiple slices from an FPGA resource pool, and the resource utilization outperforms the existing naive algorithms.

Our proposed algorithm is needed to be improved. The algorithm that finds MER (Maximum Empty Rectangles) by SLA (Scan Line Algorithm) allocates MER in response
to the request for the required number of FiCs, however it can only consider the allocation of consecutive (Rectangle) boards of connected FiCs. Therefore, if the amount required at one time is large, the efficiency of board placement may deteriorate and fragmentation may increase. Regarding this, we consider that these problems will be improved by introducing an algorithm that considers the connection and delay of the connected network and an algorithm that manages the free area itself. Also, we plan to extend FiC-RM to support not only spatial scheduling but also temporal scheduling in the future.

Now, we assume a gateway server as a host to receive the requests from IoT devices, but for treating requests from IoT devices in the FPGA cluster directly, we need to consider the security problem. It is our important future work. The proposed FiC-RM is FPGA independent, so we are now implementing it into a cluster consisted of Zynq-Ultrascale+ and Alveo U50.

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR19K1, Japan.

References

[1] S. Byrna, J.G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, “FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack,” 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, pp.109–116, May 2014.
[2] Amazon Web Services, Inc., “Amazon EC2 F1 Instance,” https://aws.amazon.com/jp/ec2/instance-types/f1/
[3] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schulkraza, and C.J. Rossbach, “Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS,” 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, pp.107–127, USENIX Association, Oct. 2018.
[4] Z. István, G. Alonso, and A. Singla, “Providing multi-tenant services with FPGAs: Case study on a key-value store,” 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp.119–124, 2018.
[5] K. Azegami, K. Musha, K. Hironaka, A.B. Ahmed, M. Koibuch, Y. Hu, and H. Amano, “A STDM (static time division multiplexing) switch on a multi-FPGA system,” 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc), pp.328–333, 2019.
[6] ETSI, “Mobile edge computing: A key technology towards 5G,” ETSI White Paper no.11.
[7] A. Vaishnav, K.D. Pham, and D. Koch, “A survey on FPGA virtualization,” Proc. FPL 2018, Sept. 2018.
[8] Y. Hu and M. Koibuchi, “Optimizing slot utilization and network topology for communication pattern on circuit-switched parallel computing systems,” IEICE Trans. Inf. & Syst., vol.E102-D, pp.247–260, Feb. 2019.
[9] K. Ito, K. Iizuka, K. Hironaka, Y. Hu, M. Koibuchi, and H. Amano, “Implementing a multi-ejection switch and making the use of multiple lanes in a circuit-switched multi-FPGA system,” Proc. CANDARW, pp.211–217, Nov. 2010.
[10] K. Hironaka, B.A. Akram, and H. Amano, “Multi-FPGA management on flow-in-cloud prototype system,” 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2019), pp.443–448, July 2019.
[11] A. Vaishnav, K. Pham, and D. Koch, “Live migration for OpenCL FPGA accelerators,” Proc. ICFPT 2018, pp.38–45, Sept. 2018.
[12] A. Fukuhara, S. Shohata, and H. Nishi, “FPGA context-based live migration maintaining network consistency,” Proc. CANDARWS, pp.81–86, 2020.
[13] J. Cui, Q. Deng, X. He, and Z. Gu, “An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs,” 2007 Design, Automation & Test in Europe Conference & Exhibition, pp.1–6, 2007.
[14] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, “Enabling FPGAs in hyperscale data centers,” Proc. UIC-ATC-ScaleCom, pp.1078–1086, 2015.
[15] K. Fleming and M. Adler, “The LEAP FPGA operating system,” Proc. FPGAs for Software Programmers 2016, pp.245–258, Sept. 2016.
[16] Z. Wang, S. Zhang, B. He, and W. Zhang, “Melia: A MapReduce framework on OpenCL-based FPGAs,” IEEE Trans. Parallel Distrib. Syst., vol.27, no.12, pp.3547–3560, 2016.
[17] O. Pell, O. Mencer, K.H. Tsoi, and W. Luk, “Maximum performance computing with dataflow engines,” High-Performance Computing Using FPGAs, pp.747–774, 2013.

Miho Yamakura received B.S. degree from Keio University, Yokohama, Japan, in 2019. She is a master student in Keio University in the presence.

Ryousei Takano is a senior research scientist of the Institute of Advanced Industrial Science and Technology (AIST), Japan. He received his Ph.D. from the Tokyo University of Agriculture and Technology in 2008. He joined AXE, Inc. in 2003 and then, in 2008, moved to AIST. His research interests include system software and high performance computing. He is currently exploring an operating system for heterogeneous accelerator clouds.
Akram Ben Ahmed received his M.S.E. and Ph.D. degrees in Computer Science and Engineering from the University of Aizu, Japan, in 2012 and 2015, respectively. He later joined Keio University, Japan, as a postdoctoral researcher. He is currently a Research Scientist at the National Institute of Advanced Industrial Science and Technology (AIST), Japan. His research interests include on-chip interconnection networks, reliable and fault-tolerant systems, and ultra-low-power embedded systems.

Midori Sugaya is a professor of the Shibaura Institute of Technology (SIT), Faculty of Science and Engineering, Japan. She received her Ph.D. from Waseda University in 2010. Previously, she was a researcher of CREST project, Research and Development Center of the project from 2008–2010. Her research interests include system software and dependable computing, and emotion aware robotics. She is a member of IEEE, ACM. She is also a member of IEICE, IPSJ of Japan.

Hideharu Amano received Ph.D. degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.