Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 8, Number 2, pages 133–138, July 2019

Design and Implementation of Circuit Switched Scheduling in Flow-in-Cloud Systems

Yao Hu, Michihiro Koibuchi
Information Systems Architecture Science Research Division
National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430
{huyao, koibuchi}@nii.ac.jp

***Abstract*—The FiC (Flow-in-Cloud) project aims at an efficient AI computing system composed of FPGA-based switching nodes called FiC-SWs, which are connected by a number of serial links. Unlike other multi-FPGA packet-switching systems, the switch fabric is realized by circuit switching implemented with time-division multiplexing (TDM) for predictable communication latency between any compute nodes. One of major challenges in FiC systems is to effectively schedule applications to achieve an efficient task execution system using limited available network resources. In this work, we present the design and implementation of circuit switched scheduling in FiC systems.**

***Index Terms*—Parallel computing, FPGA, circuit switching, job scheduling**

I. Introduction

In recent years, specific target applications such as AI accelerators and deep learning machines emerge on large-scale parallel computing systems [1]. The FiC (Flow-in-Cloud) project [2] aims at combining many heterogeneous computing nodes, such as energy efficient GPUs and FPGAs, which are directly connected by a wide-bandwidth network. FiC-SW is an FPGA-based switching node for a FiC system. Each FiC-SW is connected to its neighbors through bidirectional links. Such a parallel computing system consists of an interconnection of many FiC-SWs to enable a large number of compute nodes to communicate with each other by propagating packets in the network.

Figure 1 depicts a FiC system consisting of FPGA switching boards (i.e., FiC-SWs) with an FPGA and low-power GPUs each. The FPGA switching boards are used as network switches as well as data processing accelerators, which are connected by high-speed serial links such as optical interconnects. Unlike other multi-FPGA packet-switching systems, the switch fabric is realized by circuit switching implemented with time-division multiplexing (TDM) for predictable communication latency between any computing nodes. The network keeps the common benefits of the traditional circuit switching technology, e.g., it provides a dedicated communication channel (circuit) for each communication node pair. To support more than one path on a link, we introduce the technology of clock-by-clock time-division multiplexing. Each link can have multiple time slots, and each input slot is associated to an output slot according to its routing table. The number of time slots and the routing table are statically set before a job or a part of a job is executed, and thus they are basically not

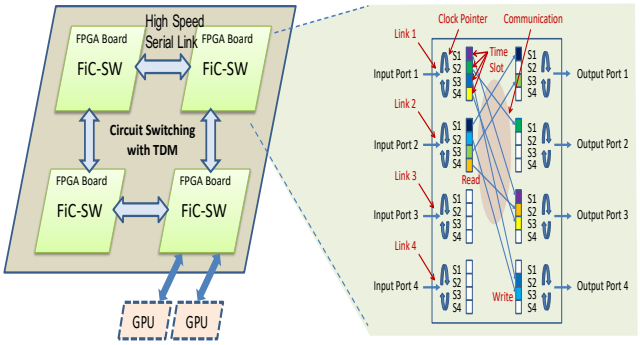

Figure 1. A FiC system using TDM-based FPGA circuit switching.

changed on the fly. Each FiC-SW in the network caches (reads) incoming data in an input slot (buffer) and later transfers (writes) it to an output slot (buffer). The connection between the input slot and the output slot is established beforehand. The number of allocated time slots is a direct factor to affect the communication latency. As the size of the network is scaled up, the transmission delay between distant FiC-SWs increases significantly, and it results in higher power consumption and lower communication performance. In our previous works [3] [4], we have proposed a case for fine-grained circuit-switched networks to make predictable the lowest bandwidth and the largest latency for every communication node pair.

One of major challenges in FiC systems is to achieve an efficient task execution system using limited available network resources. To share the limited network resources among many applications with ever-increasing demand in FiC systems, a high-performance communication among compute nodes is extremely essential. Efficient job scheduling can perform an important role in boosting the communication performance with these network resources and their interconnection networks. In this study, we explore the use of a circuit switched job scheduling method rather than traditional packet switching mechanisms to efficiently support diverse communication patterns in our target FiC interconnection networks.

The rest of this paper is organized as follows. Section II describes the mechanism of the FiC system and its interconnects. Section III presents the design and implementation of circuit switched scheduling in FiC systems. Section IV shows several preliminary evaluation results. Section V concludes

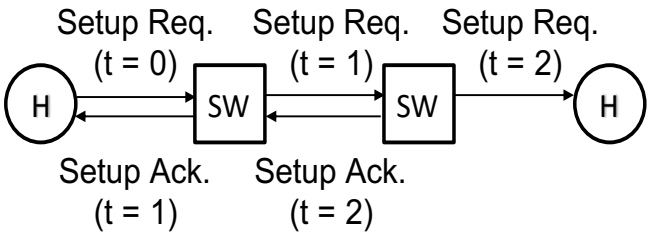

Figure 2. Procedures of circuit setup.

with a summary of our findings in this paper.

II. Circuit Switched FiC Interconnects

A. Circuit Setup and Data Transfer

In FiC systems, circuits are dynamically created for each flow. The setup request reserves a default (low) bandwidth for each flow. When an application requires a higher bandwidth, the bandwidth allocation can be combined to achieve an aggregated higher bandwidth. Figure 2 illustrates the setup process using a simple 2-host 2-switch topology, where the source host tries to establish a flow to the destination host. The source host initiates a setup request, which travels through all intermediate switches to reserve bandwidth, and finally reaches the destination host. The source host does not wait a full round trip delay before it receives an ACK. The switches respond with ACKs immediately after processing setup requests. The ACK travels to the previous switch or host, informing it that the current switch is ready to accept data. After the switch processes the setup request, it simultaneously forwards the setup request to the next hop switch and sends an ACK to the previous hop switch, which relays it to the source host. When the source host receives a setup ACK, it begins sending data on the connection. In this protocol, the source host does not wait for a full round trip delay before pumping data into the network.

If the connection is rejected by some switch along the route (e.g., a common available time slot cannot be found), a teardown message is sent along the reverse path to the source host, releasing resources allocated in the switches it passes. Similarly, when a switch or a link fails, the neighboring switch detects the error and tears down all existing connections. When the source host receives the teardown message, it attempts to use an alternative switch or to use another routing path to create a new connection. The data injected into the connection where a setup request subsequently failed is discarded by the network, and resent by the source host after another circuit is successfully set up.

B. Circuit Switching with Time Division Multiplexing

One port of the swiching node, FiC-SW, is connected to a compute node or a neighboring FiC-SW. Each port consists of one or several links and each link consists of several time slots (buffers). The volume of one time slot (buffer) is the size of the data received and accommodated within one cycle.

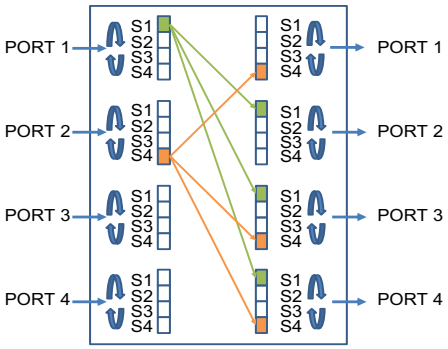

Figure 3. Multicast is supported in FiC systems.

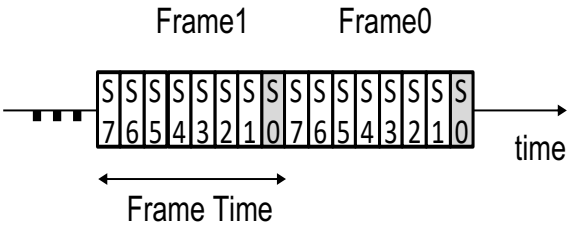

Figure 4. Example of TDM frames on the serial link.

The connection between an input slot and an output slot is established before the data transfer. Updating of the connection is supported after circuit reconfiguration. The read or write operation on the input or output side tours the time slots one by one in cycles. On the same time slot, the read and write operations are synchronized with the same frequency, thus no conflict like write-after-read or read-after-write occurs. Because each connection is allocated one or multiple time slots during the whole communication, for each communication node pair its link bandwidth can be guaranteed and its end-to-end latency can be predictable. Note that here the connection not only refers to the normal 1-to-1 unicast, but also supports the 1-to-m multicast (Fig. 3) so as to improve the whole network transfer efficiency.

Figure 4 depicts an example of data frame which is composed of 8 time slots each. If there are n time slots allocated to one link, and a communication occupies m $(m < n)$ slots, its bandwidth is m/n link bandwidth. Therefore, the end-to-end latency of any communication in the network can be guaranteed or predictable. The end-to-end reconfiguration time is defined as t_c, and thus the scheduling reconfiguration time can be calculated as $n \times t_c$. The most important thing for the FiC-SW design is to get the minimum necessary number of time slots (N). On one hand, a large number of time slots can cause large end-to-end latency because of iterative time slot access. On the other hand, a small number of time slots may result in long queuing or even conflict among multiple communications at the same port. When the FiC interconnection network is composed of identical FiC-SWs installing the same number of time slots, the minimum necessary number of slots (N) is a direct factor to affect the

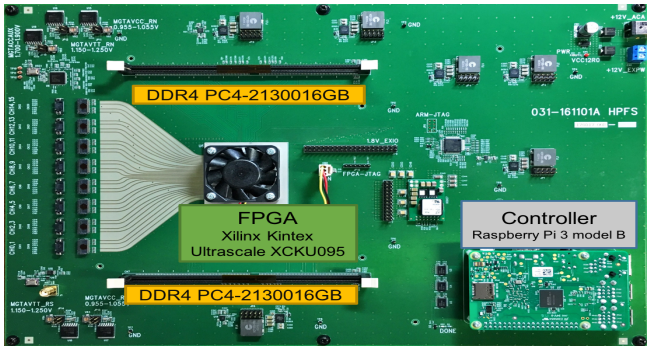

Figure 5. The prototype FPGA board of FiC-SW [2].

whole network latency performance.

C. Prototype FPGA Switching Board (FiC-SW)

The work [2] has developed a prototype FPGA board of FiC-SW, as shown in Fig. 5. It provides 32 high speed serial lanes with 9.9Gbps bandwidth at maximum. There are 8 full-duplex links by using four lanes for each direction. Here, the 9.9Gbps high speed serial connection is called a lane, and the group of four lanes which connects a pair of nodes is called a link. As for payload, each lane has 8.5Gbps data rate and each link has a total of 34Gbps data rate. After deserialization with 64b/66b coding, the input data are translated into 150MHz 64-bit signals. These 64-bit signals are handed off to the FPGA internal 100MHz clock domain. After ECC decoding, an 85-bit payload is obtained in every 100MHz clock cycle. The FiC system uses the same clock source for the communicating compute nodes in the entire system. Therefore, there is no clock frequency difference between a sender and a receiver, and the flow control to compensate possible slight frequency difference between the sender and the receiver is not required.

The FiC-SW FPGA board is equipped with two 16GB DRAM modules, and each can be accessed with 72-bit data width to store streaming data or weight data for Convolutional Neural Network (CNN). In order to provide a large number of links and DRAM modules, a high-end FPGA in the middle rank family, Xilinx Kintex Ultrascale XCKU095, is used. It is one of the most cost-efficient FPGAs available in the current market. For configuration and data management, a Raspberry Pi 3 card is mounted as a daughter board which can be connected with Ethernet.

III. Circuit Switched Scheduler

We developed a circuit switched scheduler in C++ [5] for job dispatching and scheduling in FiC systems. The program supports traditional typical traffic patterns [6] [7] and any custom traffic patterns injected in circuit switched networks, including *uniform*, *bit reversal*, *matrix transpose*, *neighbor*, *perfect shuffle*, *butterfly*, *bit complement* and *tornado*. The program supports to estimate the minimum necessary number of slots (N), and generate the routing table for each switch in a circuit switched network with any specified traffic pattern. The program is also implemented to schedule the jobs defined in a workload file with an existing or customized job scheduling strategy, which is elaborated in this section.

A. Key Entity Types

1) Pair:

- pair_id: ID of the node pair (source and destination)
- flow_id: ID of the data flow, which contains one or multiple node pairs
- job_id: ID of the job, which contains one or multiple data flows
- paths: links the node pair passes through
- source: source node of the pair
- destination: destination node of the pair
- alloc_slot: allocated slot number for the node pair
- hops: number of hops of the node pair

2) Flow:

- pairs_id: IDs of contained node pairs
- flow_id: ID of the data flow
- paths: links the data flow passes through
- alloc_slot: allocated slot number for the data flow

3) Job:

- time_submit: submission time of the job
- time_run: runtime or execution time of the job
- node_num: requested number of compute nodes
- pairs: logical communication node pairs within the job
- pairs_m: physical communication node pairs on the system
- job_id: ID of the job
- time_dispatch: dispatched time of the job
- nodes: list of occupied compute nodes
- flows: IDs of contained data flows

B. Workload File

The scheduler recognizes the job list enclosed in a workload file. Typically, a workload file includes at least the following items:

- submit_time: submission time of the job
- run_time: runtime or execution time of the job
- node_num: requested number of compute nodes
- source: ID of the source node
- destination: ID of the destination node
- flow_id: ID of the belonging flow
- job_id: ID of the belonging job

We use a common approach to model parallel "rigid" jobs [8], which refer to jobs that use a fixed number of compute nodes during runtime. As a job comes in, the scheduler evaluates the job requirements, places it into the job queue and prioritizes it according to the job's requested number of compute nodes. The priority to allocate unused compute nodes is determined by a specific scheduling policy, e.g., FCFS [9] and backfilling [10] [11]. Each compute node runs only one job at a time. Currently, the processing time used for job execution is the runtime as specified in a workload file. The simulation time starts from the first job submission and ends until all the jobs are finished.

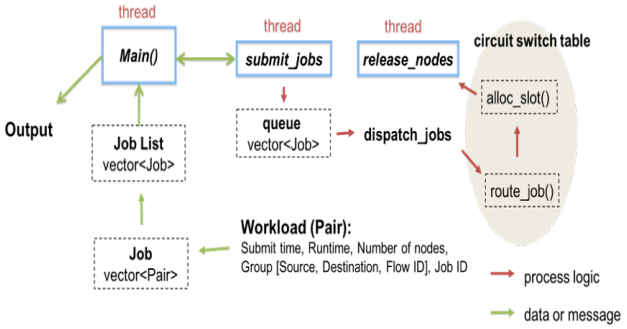

Figure 6. Processing logic of our circuit switched scheduler.

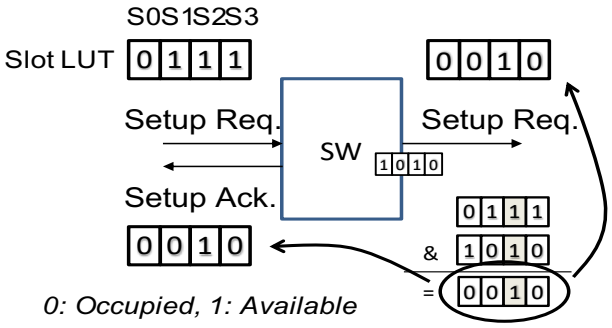

Figure 7. Allocation of slot number.

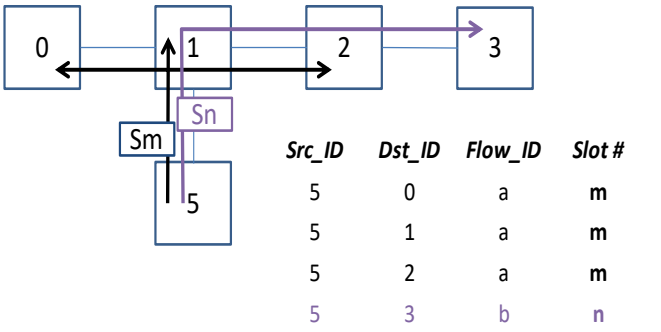

Figure 8. The node pairs with the same flow ID should be assigned the same slot number.

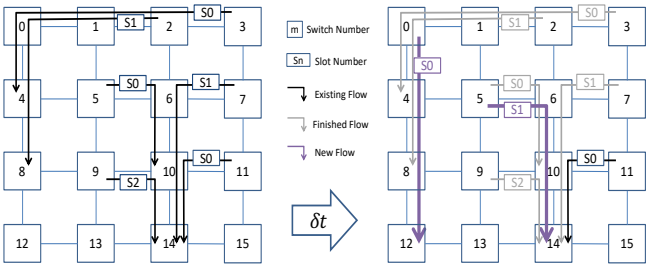

Figure 9. The occupied slot will be released for the use of a future data flow after the current application is finished.

C. Processing Logic

The program of our circuit switched scheduler is implemented with the multi-threaded programming model, which includes the following three basic threads:

- main: responsible for resolving user jobs and generating simulation results
- submit_jobs: responsible for submitting and dispatching user jobs according to specified requirements in workloads
- release_nodes: responsible for releasing occupied compute nodes after related jobs are finished

After a user job is dispatched from the job queue, the routes are calculated for the communicating node pairs. Usually, the communication travels via the shortest path between the commnicating nodes. In the meantime, a slot number is assigned for each data flow respectively within the job, and the intermediate switches along the path need to update their routing tables. An illustration of the processing logic of our circuit switched scheduler is shown in Fig. 6.

D. Slot Allocation

Figure 7 gives a simple example of how slot number is allocated. When a setup request reaches the switch, a common available time slot should be identified if it exists. This typically requires all switches on the path to coordinate this request. When the switch services an incoming setup request from the previous hop, it first ANDs the received slot LUT with its own slot LUT to identify a common available slot. It then forwards the setup request together with the new slot LUT to the next switch along the path. If a common available slot is found, the swith also sends an ACK to the previous hop with the new slot LUT. When the next switch responds with an ACK later, the slot LUT update in the current switch associates its own common available slot with the local slot of the next switch (not shown). Eventually, an available slot is assigned to the sending data flow. If a common available slot cannot be found, the system should wait for the release of an occupied slot.

Notice that, the node pairs with the same flow ID should be assigned the same slot number. For example, as shown in Fig. 8, the node pairs (5, 0), (5, 1) and (5, 2) have the same flow ID a, therefore they are assigned the same slot number m. In other words, the multicast can be also realized by specifying the same flow ID to multiple communicating node pairs. In this example, the node pair (5, 3) with the flow ID b is an usual unicast communication, which is assigned the slot number n.

The occupied slot will be released for the use of a future data flow after the current application is finished. As shown in Fig. 9, after the communications (3, 4) and (2, 5) are finished, their occuiped slots *S0* and *S1* are released. Thus the new communication (0, 12) can be allocated the slot *S0* again. Similarly, after the communications (5, 10), (7, 14) and (9, 14) are finished, their occupied slots *S0*, *S1* and *S2* are released. Because the communication (11, 14) occupying the slot *S0* still exists in the network, the slot *S0* cannot be repeatedly assigned to the new communication (5, 14). It is prohibited that two flows occuping the same slot number are sent towards

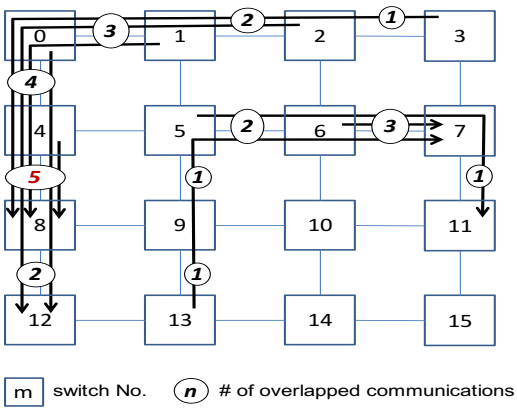

Figure 10. The minimum necessary number of slots (N) with an example communication pattern over a 2-D mesh network.

the same destination node, because this may hurt or damage the end host. Therefore, instead, the slot *S1* can be assigned to the new communication (5, 14).

E. Number of Slots

We analyze the minimum necessary number of slots (N) of all switches with various famous communication patterns over the FiC interconnection networks. The value of N is equal to the maximum number of overlapped communications on the same link, if one communication consumes one slot.

We calculate the maximum number of overlapped communications on the same link by using a modified version of an on-chip data transfer algorithm [12]. Firstly, we divide a parallel application into a set of small parallel tasks, which are mapped to respective communication node pairs as the communication pattern. Secondly, for all communication node pairs we calculate the communication path from a source node to its destination node. Finally, we get the maximum number of overlapped communications going through the same switch port, which is equal to the value of N for the switch and network design.

Figure 10 shows a 2-D (4×4) *mesh* network topology with an example communication pattern. Let $c(s, d)$ denote the communication from switch s to switch d, and let $l(u, v)$ denote the link from switch u to switch v. Note that we assume *full-duplex* links, that is, two communications going along opposite directions do not interfere with each other and they are treated separately. Therefore, the communication $c(s, d)$ is not equal to the communication $c(d, s)$ and the link $l(u, v)$ is not equal to the link $l(v, u)$. In this network, the maximum number of overlapped communications $c(0, 12)$, $c(1, 8)$, $c(2, 12)$, $c(3, 8)$ and $c(4, 8)$ on the same link $l(4, 8)$ is 5, thus $N = 5$.

F. Routing Table

The information of slot number is essentially important in our design of circuit switched network in FiC systems. Each switch in FiC systems maintains a routing table which records the relationship of input port, output port and slot number.

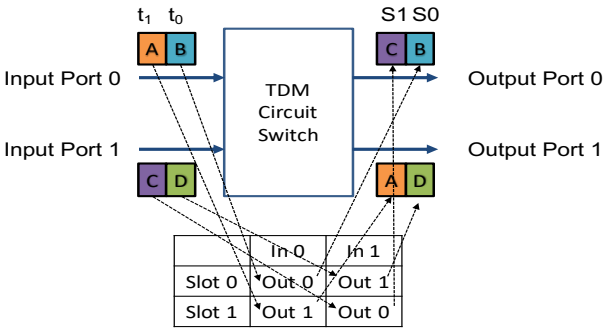

Figure 11. An illustration of routing table of a 2×2 switch.

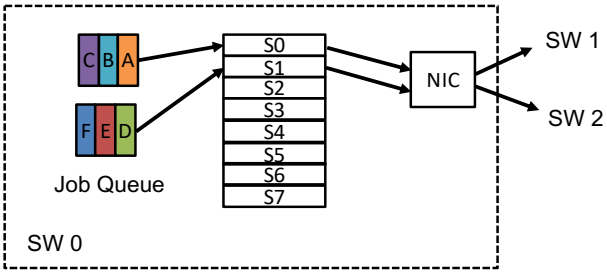

Figure 12. Each slot is equipped with a job queue in our circuit switched network.

Here, the slot number is assigned from 0, thus it should be less than the minimum nessary number of slots (N) in a circuit switched network.

Figure 11 depicts a simple example of a 2×2 switch. From the input port 0, the data cells B and A are sent via the output ports 0 (slot *S0*) and 1 (slot *S1*). From the input port 1, the data cells D and C are sent via the output ports 1 (slot *S0*) and 0 (slot *S1*).

G. Scheduling Strategy

Unlike in a traditional packet switched network, each slot is equipped with a job queue in our circuit switched network, as shown in Fig. 12. The user jobs in the queue are dispatched following a specific scheduling policy such as FCFS [9] and backfilling [10] [11]. Considering that the circuit switched network benefits from large data flows rather than small data flows due to smaller reconfiguration overhead, we can also apply the large-flow-first (LFF) policy as the scheduling strategy in FiC systems. In other words, compared to a small data flow, a larger data flow usually has a higher priority to be dispatched to minimize the number of reconfigurations especially at a busy period on the system.

IV. Preliminary Evaluation

A. Number of Slots

Table 1 shows the values of N in different networks with various communication patterns. We assume that one end node is connected to one switch. It can be seen that, as the network size increases, the value of N becomes large. For example, for the communication pattern of *uniform*, at least 4, 6, 11, 14 and

Table 1. The minimum necessary number of slots (N) for different communication patterns.

# of end nodes	uniform	bit reversal	matrix transpose	neighbor	perfect shuffle	butterfly	bit complement	tornado
16 (2-D mesh, 4×4)	4	3	3	3	2	2	2	2
64 (2-D mesh, 8×8)	6	7	7	4	4	4	4	4
256 (2-D mesh, 16×16)	11	15	15	3	8	8	8	8
1024 (2-D mesh, 32×32)	14	31	31	4	16	16	16	16
4096 (2-D mesh, 64×64)	28	63	63	7	32	32	32	32
4096 (3-D mesh, $16 \times 16 \times 16$)	13	15	48	8	8	8	8	8
4096 (4-D mesh, $8 \times 8 \times 8 \times 8$)	9	56	56	5	4	4	4	4

```
    ...
=== Routing path for each node pair ===
    ...
Pair ID 4 (Flow ID 0, Job 1):
  SW 0 (port 0->1) - [slot 0] -> SW 1 (port 2->0)
Pair ID 5 (Flow ID 0, Job 2):
  SW 6 (port 0->1) - [slot 0] -> SW 7 (port 2->0)
Pair ID 6 (Flow ID 1, Job 2):
  SW 5 (port 0->1) - [slot 0] -> SW 6 (port 2->4) - [slot 0] -> SW 2 (port 3->0)
    ...

=== Port information for each switch ===
SW 0 :
    Port 0 (Slot 0) --> Port 1 (Slot 0), from node 0 to node 1 (Pair ID 4, Flow ID 0, Job 1)
    Port 1 (Slot 1) --> Port 3 (Slot 1), from node 3 to node 8 (Pair ID 8, Flow ID 2, Job 2)
SW 1 :
    Port 2 (Slot 0) --> Port 0 (Slot 0), from node 0 to node 1 (Pair ID 4, Flow ID 0, Job 1)
    Port 1 (Slot 1) --> Port 2 (Slot 1), from node 3 to node 8 (Pair ID 8, Flow ID 2, Job 2)
SW 2 :
    Port 3 (Slot 0) --> Port 0 (Slot 0), from node 5 to node 2 (Pair ID 6, Flow ID 1, Job 2)
    Port 1 (Slot 1) --> Port 2 (Slot 1), from node 3 to node 8 (Pair ID 8, Flow ID 2, Job 2)
    ...
```

Figure 13. Example run of our circuit switched scheduler.

28 time slots are required if a 2-D mesh network is composed of 16, 64, 256, 1024 and 4096 switches, respectively. Besides, if the network size is the same, a higher-dimension network requires less number of slots. For instance, in 4096-node networks with the *uniform* communication pattern, 2-D mesh, 3-D mesh and 4-D mesh require at least 28, 13 and 9 slots, respectively.

B. Output of Circuit Switched Scheduler

The program of our circuit switched scheduler is run to simulate job dispatching and scheduling in FiC systems. Simulation results are sorted by scheduling clocks (e.g., t1, t5, t7, t9, etc.) when a new incoming job is dispatched or an existing running job is finished. For each scheduling clock, there are two types of simulation output files. One type is the system log file which records the current job status and resource utilization on the system. The other type is the current routing table for each switch in the network, which records the relationship of input port, output port and slot number.

We run the circuit switched scheduler using a small workload of nearly 100 jobs with random arrival timings for a Poisson process. Each job specifies the required number (1, 2, 4, 8 and 16) of compute nodes in a 2-D (4×4) mesh network. In each output log file at a certain scheduling cycle, as shown in Fig. 13, two kinds of key information are generated, routing path for each node pair and port information for each switch. For both the records, the information of slot number is used to adjust the job scheduling strategy and improve the link utilization over the whole network. In the future, we will customize more efficient job scheduling algorithms using a real supercomputer trace of user jobs taken from [13] for FiC systems.

V. Conclusion

In this work, we introduced FiC (Flow-in-Cloud), the first large-scale multi-FPGA system equipped with circuit switching network. In a FiC system, FiC-SW is an FPGA-based switching node which is implemented with time-division multiplexing (TDM). We presented the design and implemenation of circuit switched scheduling which depends on multiplexing of time slots in each FiC-SW switching node to increase link utilization and communication performance in FiC systems. Preliminary evaluation results showed the effectiveness and efficiency of our circuit switched scheduler for FiC systems.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 19K20263.

References

[1] "Nvidia dgx-1," https://www.nvidia.com/ja-jp/data-center/dgx-1/.
[2] K. Musha, T. Kudoh, and H. Amano, "Deep learning on high performance fpga switching boards: Flow-in-cloud," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Eds. Cham: Springer International Publishing, 2018, pp. 43–54.
[3] Y. Hu, T. Kudoh, and M. Koibuchi, "Reducing number of slots in circuit-switched network for parallel computers," *IEICE Technical Report*, vol. 117, no. 153, pp. 111–116, July 2017.
[4] ——, "A case of electrical circuit switched interconnection network for parallel computers," in *The 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT' 17)*, Taipei Taiwan, December 2017.
[5] [Online]. Available: https://github.com/KoibuchiLab/circuit-switch-scheduler
[6] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: an engineering approach*. Morgan Kaufmann, 2002.
[7] W. D. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.
[8] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, March 2015.
[9] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "On the design and evaluation of job scheduling algorithms," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1999, pp. 17–42.
[10] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The easy - loadleveler api project," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1996, pp. 41–47.
[11] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Computing*, vol. 12, pp. 529–543, 2001.
[12] M. Koibuchi, K. Anjo, Y. Yamada, A. Jouraku, and H. Amano, "A simple data transfer technique using local address for networks-on-chips," *IEEE Transaction on Parallel and Distributed Systems*, vol. 17, no. 12, pp. 1425–1437, 2006.
[13] "Parallel workloads archive," http://www.cs.huji.ac.il/labs/parallel/workload/.