Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 05, No. 02, August 2019.

A PARALLEL PIPELINED PACKET SWITCH ARCHITECTURE FOR MESH-CONNECTED MULTIPROCESSORS WITH INDEPENDENTLY ROUTED FLITS

Jamil Al-Azzeh[1], Mohammed Agmal[2] and Igor Zotov[2]

(Received: 27-Apr.-2019, Revised: 30-Jun.-2019, Accepted: 22-Jul.-2019)

ABSTRACT

In this paper, a packet switch architecture for mesh-connected multiprocessors based on the use of a set of input FIFO buffers and an output register matrix controlled by a novel distributed timing-based scheduling scheme is proposed. Simple static routing is assumed, with each packet split into a set of independently routed w-bit-wide flits. The device achieves at least 78% throughput for uniformly distributed traffic and an asymptotic higher bound of 100%. In contrast to the state-of-the-art VOQ-based switch architectures, the proposed switch is shown to reach its maximum throughput with no internal speedup required and has an order of magnitude lower hardware complexity. Compared to existing buffered crossbar non-VOQ switches with typical flit scheduling mechanisms, the proposed device demonstrates slightly higher throughput and substantially shorter delays in some practically important cases.

KEYWORDS

Multiprocessor, Mesh topology, Packet switching, Input-queued switch, FIFO-buffer, Flit, Pipelining, Throughput.

1. INTRODUCTION

Switching hardware is known to play a crucial role in the operation of a wide class of modern computer systems [1]. Mesh-connected multiprocessors are an example of systems whose performance is profoundly affected by the underlying built-in switching apparatus [2]. Inter-processor data exchange speed and remote memory access latency significantly depend on the throughput and performance of the switches distributed across the multiprocessor mesh [37]-[39].

The packet switching paradigm is the most widely used in contemporary multiprocessor designs, such as chip multiprocessors (e.g., see [3]-[5] and the references therein). Furthermore, hybrid (packet/circuit) switching approach has evolved as well (e.g., see [6]-[9]). Packet switches utilized in modern multiprocessors are in several respects similar to the asynchronous transfer mode switches employed in computer networks and supercomputers [1]. Various switch architectures mainly differ in the arrangement of internal packet buffers (queues). In input-queued switches, packets are first loaded into the corresponding input buffers and then switched to the required outputs *via* a crossbar. Such switches do not impose hard requirements for the internal speed of the switching hardware (crossbar) and, as a rule, have relatively low hardware complexity, which allows them to be utilized with a large number of inputs/outputs. However, it has been shown [10] that input-queued switches with simple FIFO buffers are limited in their throughput at approximately 0.64 in practically significant scenarios (or at $2-\sqrt{2}\approx 0.586$ asymptotically) due to the occurrence of blocked packets in the head cells of the buffers (known as HOL blocking). The highest possible throughput (up to 100%) is achieved in output-queued switches. In such devices, packets are immediately switched to the required outputs and then stored in the output buffers before being issued, which eliminates HOL blocking. However, output-buffered switches impose strict requirements on the internal speed of the switching apparatus, which must be several times higher than the external speed at which packets travel between processors. In this regard, their use is feasible with a relatively small number of inputs/outputs *n* and/or if the external switch speed is deliberately decreased.

To overcome the problems inherent to input- and output-queued switches with ordinary FIFO buffers,

1. J. Al-Azzeh is with Dep. of Computer Eng., Al-Balqa Applied University, Al-Salt, Jordan. Email: azzehjamil18@gamil.com
2. M. Agmal and I. Zotov are with Southwest State University, Russia. Emails: MohammedAgmaAbdo@gamil.com and zotovigor@yandex.ru

several solution methods have been proposed in the recent past (e.g., see [11]-[16] and the references therein). Most of these methods are based on the virtual output queue (VOQ) paradigm first proposed in [17]. In a VOQ switch with *n* input and *n* output trunks, *n* separate simple FIFO buffers are combined in parallel at each input port, each of which corresponds to a given output. Packets arriving at a particular input are immediately routed to an input queue corresponding to the required output and are then switched to the output line *via* a crossbar circuit; therefore, packets of the same input queue never require to be transferred to different outputs and thus HOL blocking never occurs.

In order to maximally improve the VOQ switch throughput, a central scheduler is required for contention resolution when two or more packets are destined to the same output at the same time slot (e.g., see [18]-[19]). Moreover, to achieve 100% throughput (or non-blocking performance), a stable scheduling policy is needed for any admissible traffic pattern. It is worth noting that adding extra crosspoint buffers to the internal crossbar enhances the switch throughput as well [20]. Severe VOQ switch scheduling algorithms, such as maximum size/weight matching [21], have been introduced and studied. Although these algorithms are shown to be stable, they are too complicated for efficient hardware implementation in multiprocessors. Consequently, suboptimal solution algorithms are usually adopted. The underlying idea is to find an input-to-output matching of the maximal size (known as MSM) with no input or output left unnecessarily idle. Finding an MSM is more efficient, because it does not require backtracking. Iterative scheduling algorithms for finding an MSM have been widely adopted (e.g., see [22]-[24] and the references therein). An iterative scheduling algorithm needs to execute up to *n* iterations to guarantee maximal size match. However, as far as we know, all existing stable iterative algorithms require a speedup of 2, approximately; i.e., the internal speed of the switch must be twice the external speed at which packets are transferred between processors. A lower speedup is always desirable, because it reduces the implementation cost and increases the external speed of the switch at the same time.

Furthermore, a load-balanced two-stage-based architecture has been studied in high-speed switch design and employed to maximize the switch throughput (e.g., see [25]-[26]). Load-balanced two-stage switches are excellent techniques of getting rid of the central scheduler and reducing hardware-level complexity. Nevertheless, their main drawback is that packets may be eventually mis-sequenced. Further, load-balanced switches suffer from high delay performance under low to medium load. Thus, the main challenge of contemporary switch technology is still how to improve the efficiency of the central scheduler significantly.

Although there are no major barriers in the use of wormhole-routed VOQ-based packet switches in mesh-connected multiprocessors, specific issues and limitations arise that make the usage less efficient compared to multi-computers and computer networks. Because multiprocessor nodes exchange short packets split into a small number of *w*-bit-wide flits transferred *via* trunks each in one clock cycle, the VOQ scheduling time might become a significant portion of the packet transfer duration. The other issue is that these packet switches require the internal speedup of about 2, which leads to significantly higher implementation cost and makes it impossible to set state-of-the-art external speed values. In this regard, it is worth mentioning the Epiphany IV VLSI-multiprocessor architecture as an example, which has 136-bit-wide packets with 64 bits of data, 64 bits of address and 8 bits of control [5].

The independently routed flit (IRF) paradigm is one of the alternatives to the traditional wormhole-routed VOQ-based networking for mesh-connected multiprocessors. With the IRF approach, a packet is divided into a set of flits of the same width *w*, each of which carries both data and address fields in addition to some control/identification bits (e.g., see [27]-[28]). In contrast to wormhole-routing, in the IRF approach, flits are not grouped to be processed and transferred as an atomic entity; instead, they are routed independently based on a simple static strategy. After arriving at a given destination, flits are lined up (reassembled) in a packet. If flits arrive in a wrong order, they can be fetched by their identifiers to restore the initial ordering, which is a deadlock and livelock free process. If one or more flits are missing, a timeout mechanism can help solving this reliability-related problem with no permanent blocking taking place. The main advantage of the IRF networking is that it provides more simple hardware solutions than wormhole virtual cut-through routing, because no specific wormhole-aware algorithms and apparatus are required to support virtual channels and to guarantee deadlock and livelock freedom.

In this paper, we consider mesh-connected multiprocessors similar to the Epiphany IV [5] or the Tile-Gx series [4]. Moreover, we employ the IRF-networking-based switch architecture with *w*-bit-wide flits and simple static routing [28]. Our main contribution is the proposition of a parallel pipelined switch architecture (which we further refer to as the PPIRF switch) based on the use of non-VOQ input FIFO buffers and an output register matrix (buffered crossbar) controlled by a built-in distributed flit scheduler implementing a novel row-wise oldest-flit-first discipline formalized based on the construction and manipulation of flit consistency graphs. The proposed architecture achieves the throughput of at least 78% for all practically significant scenarios and 100% asymptotic switch throughput (assuming Bernoulli uniform traffic). The major effect gained is that neither virtual queuing at input ports, nor internal speedup is required to achieve up to 100% asymptotic throughput, which leads to a quadratic asymptotic hardware complexity—an order of magnitude less than that of VOQ-based switches. Compared to similar buffered-crossbar-based switches, the PPIRF switch demonstrates slightly higher throughput and significantly shorter delays in some cases (e.g., for heavier flit traffic patterns), resulting from the novel scheduling policy employed. In what follows, we formally state the structural organization and operation of the proposed parallel pipelined switch; further, we present and briefly discuss some simulation results; finally, we make some comparison and present concluding remarks and future directions as well.

2. The Parallel Pipelined IRF Switch Architecture

Figure 1 shows a formalized structural model of the parallel pipelined IRF switch. The switch has several external connections (trunks)—a set of inputs $I_1, I_2, \ldots, I_n$ and a set of outputs $O_1, O_2, \ldots, O_n$—and is composed of input FIFO buffers (queues) $Q_1, Q_2, \ldots, Q_n$ of length (size) *L*, a register matrix $B = \|B_{ij}\|,\ i, j = \overline{1, n}$, demultiplexers $R_1, R_2, \ldots, R_n$, multiplexers $K_1, K_2, \ldots, K_n$, $M_1, M_2, \ldots, M_n$ and gates $G_1, G_2, \ldots, G_n$. In Figure 1, the solid lines represent flit transfer paths, while the dashed lines indicate the control signal paths.

Input I_1 and output O_1 have a predefined function to connect the switch to the corresponding processor core (current tile). The rest of the inputs and outputs link the current node to the neighboring nodes in the multiprocessor mesh. This means that for a two-dimensional (2D) mesh multiprocessor, a 5×5 switch is required ($n = 5$) at each node; in turn, a 3D mesh system would require the use of a 7×7 switch ($n = 7$).

The functions of the blocks in Figure 1 are as follows. The input FIFO buffers $Q_1, Q_2, \ldots, Q_n$ are used to temporarily store flit streams arriving at the corresponding input trunks before they are transferred to the register matrix *B*. It has to be mentioned that the size (*L*) of the buffers is specified at the switch implementation stage and is assumed to be arbitrary. Each input has the only associated FIFO buffer structured around a set of static registers, each of which is capable of holding a single *w*-bit-wide flit. The set of demultiplexers $R_1, R_2, \ldots, R_n$ implement a predetermined flit routing algorithm μ. The function of the register matrix *B* is to automatically distribute incoming flit streams between its rows following the output directions obtained from the routing algorithm. The multiplexers $M_1, M_2, \ldots, M_n$ implement a given flit scheduling scheme φ (the oldest-flits-go-out-first policy, in our case), which is presented in detail below. Furthermore, the multiplexers $K_1, K_2, \ldots, K_n$ together with the gates $G_1, G_2, \ldots, G_n$ are needed to temporarily block flits from being transferred to the register matrix from the corresponding queues if their respective target registers still contain unissued flits (HOL blocking).

3. The Operation of The PIRF Switch

The operation of the PPRF switch is generally organized in four cyclically repeating steps, which are as follows:

1. Determine the output directions for all the flits located in the head cells of the input queues (further referred to as the head flits) based on the routing algorithm μ.
2. Transfer the head flits from the respective input queues to the register matrix *B*, unless the corresponding target registers in the register matrix contain any other flits; then shift the queues in the case of a successful transfer.
3. Analyze the current flits-to-registers mapping and select a subset of flits, which must be issued from the register matrix based on the scheduling scheme φ taking into account the time elapsed from the

moments of their arrival.

4. Issue the selected subset of flits to the respective outputs and reset the corresponding registers of the register matrix *B*.

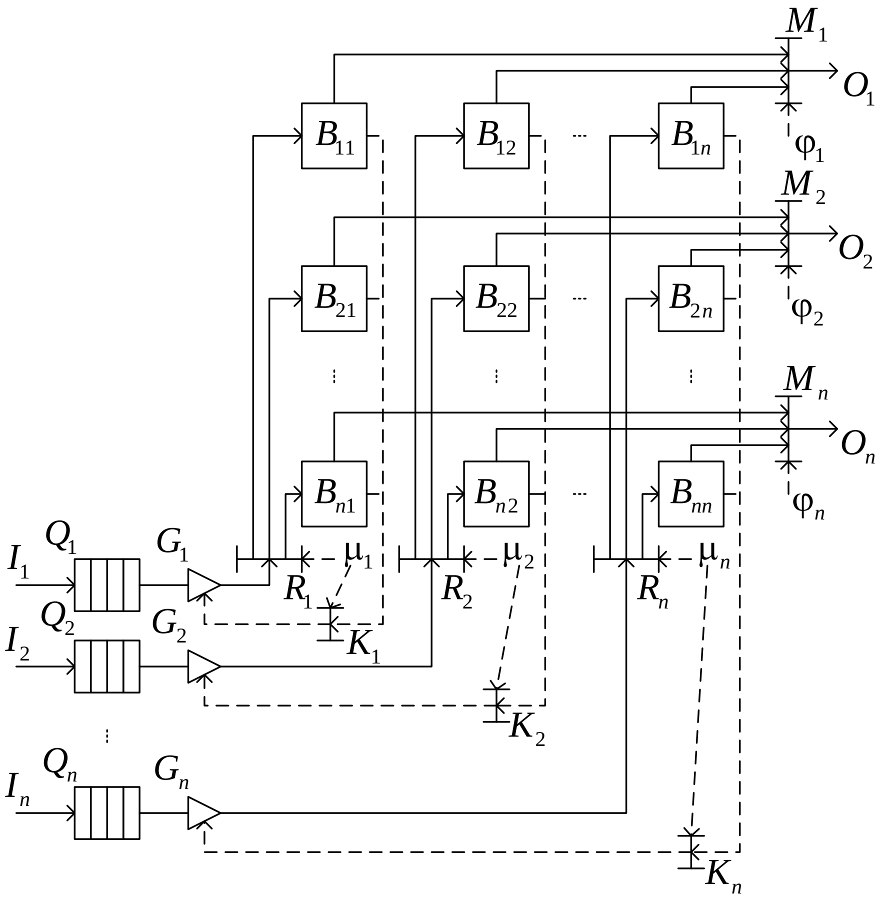

Figure 1. A formalized structural model of the parallel pipelined IRF switch.

Figure 2 shows a detailed parallel flow-chart representing the operation of the PPIRF switch. The flow-chart uses some formal constructs and symbols, whose meanings are explained below.

In Figure 2, F_k stands for the actual set of flits distributed between the registers of the register matrix *B*, where *k* denotes the current cycle on the switch operation timeline. The flit consistency graph Γ_k with a set of vertices F_k and a set of edges $\alpha_k \subseteq F_k \times F_k$ is introduced to indicate whether a pair of flits can be issued from the register matrix in parallel. The following rule formally defines the flit consistency relation α_k :

$$(f_q, f_r) \in \alpha_k \Leftrightarrow f_q \in \bigcup_{j=1}^{n} S_k(B_{i'j}),\ f_r \in \bigcup_{j=1}^{n} S_k(B_{i''j}) : i' \neq i'', \tag{1}$$

where S_k is an indicator function, such that $S_k(B_{ij}) = f_q$ if the register B_{ij} contains the flit f_q before the k^{th} cycle begins; and $S_k(B_{ij}) = \varnothing$ if the register B_{ij} is empty. Each vertex of the graph Γ_k is given a non-negative weight τ_q, which reflects the time elapsed since the flit f_q had arrived at the respective input of the switch (measured in cycles). A clique $\Gamma'_k = \langle F'_k, \alpha'_k \rangle$, $F'_k \subseteq F_k$, $\alpha'_k \subseteq \alpha_k$ in the graph Γ_k is selected, such that the total weight of its vertices becomes the maximum across the set of candidate cliques:

$$\sum_{f_q \in F'_k} \tau_q = \max. \tag{2}$$

If the graph Γ_k contains a set of cliques, for each of which condition (2) holds, then any clique Γ'_k of this set is picked out assuming uniformly distributed random selection process. It is evident from the above formal statements that the vertices F_k' of the clique Γ_k' are consistent based on (1) and the cardinality of F_k' is maximal by inclusion.

In Figure 2, we also employed a new indicator function *s*, such that $s(Q_l)=1$ if the queue Q_l currently contains at least one flit and $s(Q_l)=0$ otherwise. The symbol $\leftarrow$ denotes register/trunk read/write operations. For example, $f_i \leftarrow \text{head}(Q_i)$ means the extraction of the head flit f_i from the queue Q_i and $O_i \leftarrow f_{j_i}$ implies that flit f_{j_i} should be issued to output O_i. Additionally, $\mu(f_i)$ stands for the direction (i.e., the output trunk) that flit f_i is supposed to be routed based on the algorithm μ.

The operation of the PPIRF switch according to the algorithm of Figure 2 is organized in a series of cycles (loops). Each cycle (*k*th cycle) consists of vertices 3 to 14 and is performed within a single time slot.

First, condition 3 is checked. If at least one input queue contains flits, the condition holds and the loop begins. Otherwise, the algorithm terminates and the switch keeps staying idle.

Each cycle starts with a parallel section composed of *n* threads. The *i*th thread ($i=\overline{1,n}$) consists of vertices 4.*i*-8.*i* and describes how the flits are fetched and transferred from queue Q_i to the *i*th column of the register matrix. First, a flit (referred to as f_i) is read from the head register of queue Q_i (vertex 4.*i*). Second, the output trunk (and the corresponding target register of the register matrix) is determined to which f_i has to be relayed according to the supported routing algorithm (vertex 5.*i*). Third, the HOL blocking condition (vertex 6.*i*) is checked. If the target register still contains another flit, the *i*th thread terminates and the state of queue Q_i remains unchanged. Otherwise, flit f_i is extracted from the head cell of queue Q_i and immediately uploaded to the target register. Then, Q_i is shifted by 1 position.

As soon as all the threads have terminated, the loop proceeds with vertices 9 and 10. According to vertex 9, graph Γ_k is formed based on the current flit-to-register distribution. In fact, it is a dummy operation, because the graph is automatically formed immediately after all the arrived flits have been mapped onto the register matrix. Thus, it takes no extra time. Then, according to vertex 10 of the algorithm, a clique of the graph is selected based on formula (2). In the selection process, each row of the register matrix is examined separately and in parallel to its peers. In each row, a flit is picked out whose vertex in graph Γ_k has the maximum weight amongst the flits mapped onto the same row. This process has $O(n)$ runtime complexity if implemented sequentially. In the case of parallel calculation of the maximum, the runtime complexity is $O(1)$ at the cost of parallel hardware. If a pyramidal maximum computation circuit is employed, the process will take $O(\log n)$ time units. Because the rows of the register matrix are examined in parallel, the entire process of picking out a clique requires $O(1)$ time in the best case (parallel scheme) and $O(n)$ in the worst case (sequential scheme). Immediately after a clique has been picked out, another parallel section begins (vertices 11.*j*-12.*j*). In accordance with the *j*th thread of this section, the selected flit of the *j*th row (the one whose vertex belongs to the clique) is transferred to the *j*th output trunk and the corresponding register is immediately reset (vertex 12.*j*).

Hence, the final part of the loop commences, which consists of vertices 13 and 14. In line with vertex 13, the current graph Γ_k is reconstructed by eliminating all the vertices that correspond to the relayed flits and by incrementing the weights of all the remaining vertices corresponding to the flits that could not be selected and transferred within the *k*th loop. And finally, *k* is incremented (vertex 14) and the next loop starts.

Figure 3 gives an example of how flit consistency graphs are constructed and treated in the operation of a PPIRF switch according to the algorithm of Figure 2 (vertices 9 and 10). In Figure 3, a 5×5 switch is considered and the state of the register matrix for 9 consecutive cycles (marked with letters a, b, …, i) of the algorithm is presented. Hereinafter in the example, the squares and the circles denote the registers of the register matrix and the flits (vertices of graphs Γ_k) being processed, respectively;

the designation f_i^k stands for a flit extracted from queue Q_i in cycle *k*; the weights τ_q of the flits are placed inside the corresponding circles.

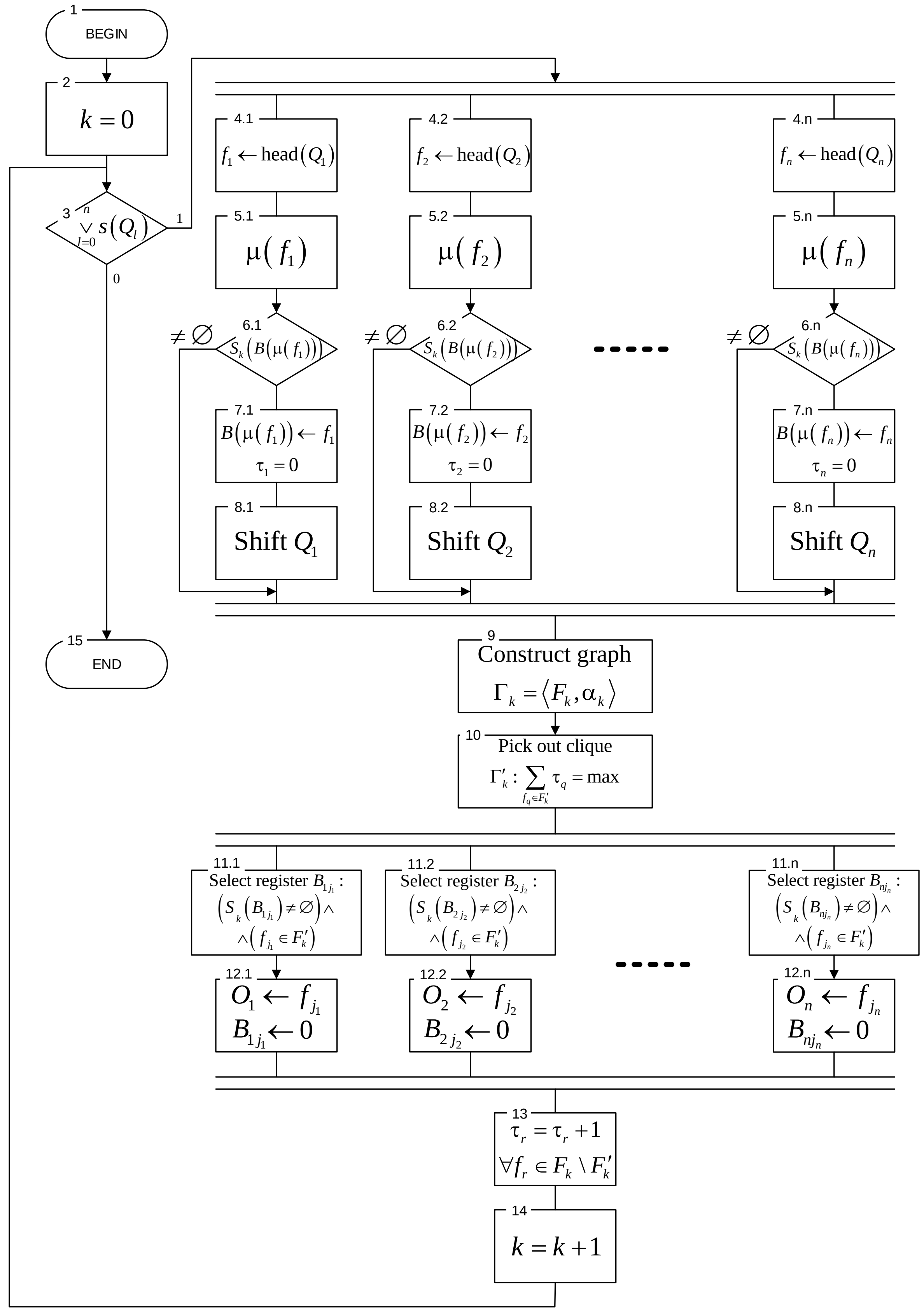
1 BEGIN
2 $k=0$
3 $\underset{l=0}{\overset{n}{\vee}} s(Q_l)$
1
0
4.1 $f_1 \leftarrow \text{head}(Q_1)$
4.2 $f_2 \leftarrow \text{head}(Q_2)$
4.n $f_n \leftarrow \text{head}(Q_n)$
5.1 $\mu(f_1)$
5.2 $\mu(f_2)$
5.n $\mu(f_n)$
6.1 $S_k(B(\mu(f_1)))$ $\neq\varnothing$
6.2 $S_k(B(\mu(f_2)))$ $\neq\varnothing$
6.n $S_k(B(\mu(f_n)))$ $\neq\varnothing$
7.1 $B(\mu(f_1)) \leftarrow f_1$ $\tau_1 = 0$
7.2 $B(\mu(f_2)) \leftarrow f_2$ $\tau_2 = 0$
7.n $B(\mu(f_n)) \leftarrow f_n$ $\tau_n = 0$
8.1 Shift Q_1
8.2 Shift Q_2
8.n Shift Q_n
9 Construct graph $\Gamma_k = \langle F_k, \alpha_k \rangle$
10 Pick out clique $\Gamma'_k : \sum_{f_q \in F'_k} \tau_q = \max$
11.1 Select register B_{1j_1} : $(S_k(B_{1j_1}) \neq \varnothing) \wedge$ $\wedge (f_{j_1} \in F'_k)$
11.2 Select register B_{2j_2} : $(S_k(B_{2j_2}) \neq \varnothing) \wedge$ $\wedge (f_{j_2} \in F'_k)$
11.n Select register B_{nj_n} : $(S_k(B_{nj_n}) \neq \varnothing) \wedge$ $\wedge (f_{j_n} \in F'_k)$
12.1 $O_1 \leftarrow f_{j_1}$ $B_{1j_1} \leftarrow 0$
12.2 $O_2 \leftarrow f_{j_2}$ $B_{2j_2} \leftarrow 0$
12.n $O_n \leftarrow f_{j_n}$ $B_{nj_n} \leftarrow 0$
13 $\tau_r = \tau_r + 1$ $\forall f_r \in F_k \setminus F'_k$
14 $k = k + 1$
15 END

Figure 2. A parallel flow-chart representing the operation of the PPIRF switch.

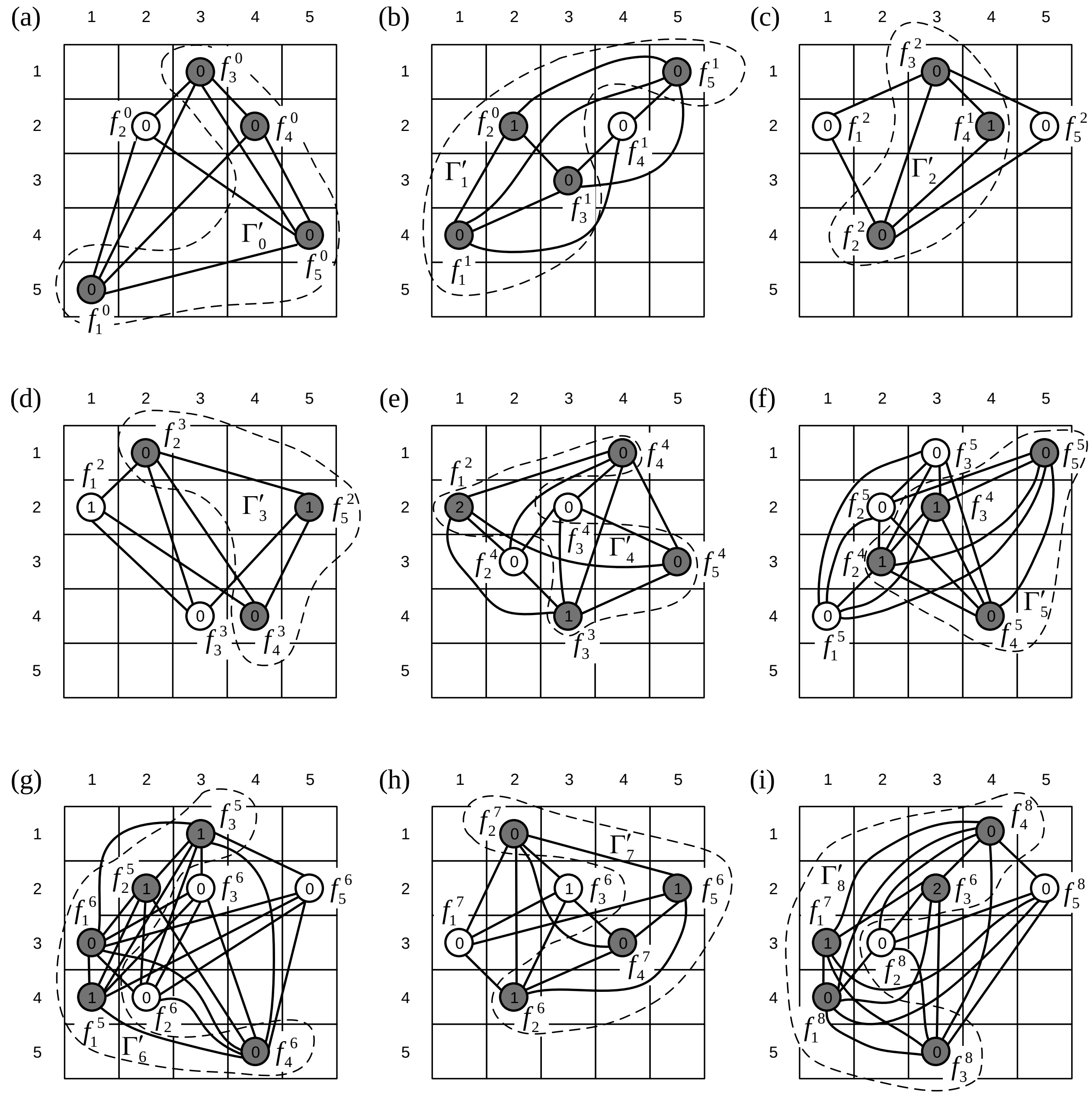

Figure 3. Example of flit consistency graph construction and manipulation in the operation of a 5×5 PPIRF switch according to the algorithm of Figure 2.

Figure 3(a)-(i) demonstrates how the cliques $\{\Gamma'_k\}$ (encircled in dashed curves) of the corresponding flit consistency graphs $\{\Gamma_k\}$ are picked out according to (2) and how the graphs are reconstructed as the process evolves step by step. Note that new flits are added to the register matrix at random as if they were directed using the routing algorithm μ.

4. Simulation Studies and Results

We conducted a series of comprehensive simulation studies to investigate the throughput, delay and some other characteristics of the proposed PPIRF switch and compare our solution with similar buffered crossbar switches supporting typical packet scheduling policies (uniform and round robin) [29]-[30]. Based on the extended Q-chart modeling language—in a similar way as utilized in [31]—we constructed a queuing model that represents the behavior of the PPIRF switch and the buffered peers under consideration. The choice of the language we used is determined by the presence of simulation entities such as group controls, which greatly simplify the implementation of various switching functions; for example, multiplexing and demultiplexing, which allows creating quite compact models for complex devices and systems. To perform the simulation experiments, we employed a dedicated Q-chart-based simulation testbed (Visual QChart Simulator) developed by Zotov et al. [32]. The Visual QChart Simulator integrated environment enables the user to input Q-chart models as graphics, automatically check and compile them to C++ code. Further, it generates an executable using a C++ command line compiler/linker chain to simulate the behavior of a given unit/system with preset to simulation parameters such as simulation duration, number of flits issued and flit traffic characteristics, to mention but a few.

All the switches examined were assumed to be queuing networks, in which flits arriving at the inputs were considered as random service request streams. Typically, Bernoulli uniform, Bernoulli hot spot and bursty uniform traffic patterns are assumed when studying the throughput and performance of the switch architectures [10], [19], [24] and [29]. However, bursty traffic is specific to the ATM-based networks and uncommon for the multiprocessors considered in the paper; therefore, we studied the uniform and hot spot traffic patterns only. Short-term traffic asymmetry was admitted. In our experiments, we set up all simulation conditions as done in [10]. Considering Bernoulli uniform traffic, the probability that a flit arrives at the i^{th} input of the switch $\left(i=\overline{1,n}\right)$ in the next cycle was assumed to be *p*; thus, the probability of no packet arrival is $q=1-p$. The supported routing algorithm μ was considered a *priori* unknown and therefore we supposed uniformly distributed selection of the register matrix target rows to transfer flits from the input buffers.

In the simulation experiments, the number of inputs/outputs *n* was selected from the range of 5 to 25 with one step increment, which corresponds to practically significant scenarios for modern multiprocessors such as mesh [4]-[5], torus [33], cube [34], twisted torus [35] and crossed cube [36]. We also studied next-generation switches separately, with $n >> 25$ (up to 1000) and set the Bernoulli distribution parameter *p* equal to 1 for all *n* inputs to evaluate the throughput of the switch. The duration of the simulation study was assumed to be 10,000 switch flit relay cycles. Besides, the required number of repetitions with fixed parameters was determined by the Student's criterion for the significance level of $\alpha = 0.02$.

Figure 4 shows the 5×5 switch Q-chart utilized in our experimental studies (note that the same Q-chart structure is used to model both the PPIRF switch and existing buffered crossbar peers [29]-[30], but different control logic is employed reflecting the corresponding scheduling policies). Five types of entities (elements) are utilized in the Q-chart: flit generators (denoted as Gx), flit processors (Dx), queues (Qx), group controllers (RCx and MCx) and gates (Kx). Table 1 presents a list of the functions of the simulation entities. In the chart of Figure 4, the solid lines denote the information links which fix flit transmission paths and the dashed lines show control signals (links) that specify the conditions affecting the state of the controlled simulation elements that are letting flits pass through or blocking them temporarily. Arrows specify in which directions flits or control signals are transmitted. Furthermore, both information and control links are unidirectional.

The gates are a cornerstone of the Q-chart logic. For the chart shown in Figure 4, the following gate enable/disable rules apply:

- one gate is enabled (open) only among gates K1*i*1, K2*i*1, K3*i*1, K4*i*1 and K5*i*1 ($i=\overline{1,5}$), with the probability of 0.2 (or the probability of $1/n$ for arbitrary *n*);
- gate K*ji*2 ($i, j=\overline{1,5}$) is enabled (open) if queue Q*ji*2 is empty;
- gate K*ji*3 ($i, j=\overline{1,5}$) is enabled (open) subject to Q*ji*2 containing a flit waiting to be issued (depending on the time the flit has spent in the switch);
- gate K_LOAD_*i* ($i=\overline{1,5}$) is enabled (open) if all queues Q1*i*1, Q2*i*1, Q3*i*1, Q4*i*1 and Q5*i*1 are empty;
- gate K_LOST_*i* ($i=\overline{1,5}$) is enabled (open) only if queue Q_*i* has a limited capacity and is full of flits (this is useful to identify the conditions when the switch starts losing incoming flits).

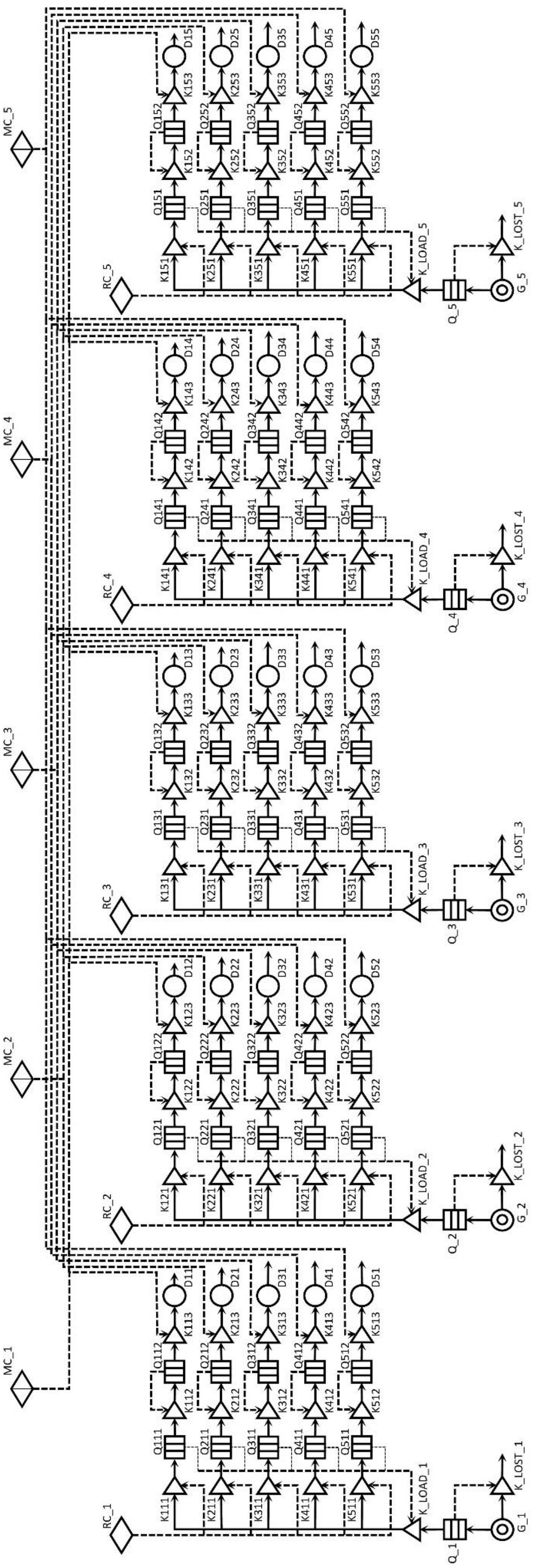

Figure 4. Q-chart representing a switch with five input and five output trunks.

Table 1. The functions of the simulation entities of the Q-chart given in Figure 4.

No.	Element ID	Element function
		Flit generators
1	G_i $(i=\overline{1,5})$	They simulate a flit stream that arrives at input I_i (input traffic).
		Flit processors
2	Dji $(i,j=\overline{1,5})$	They simulate a flit stream initially arrived at input I_i and now leaving the jth row of the register matrix *via* output O_j (output traffic).
		Queues
3	Q_i $(i=\overline{1,5})$	They represent input queues Q_i.
4	Q$ji$1 $(i,j=\overline{1,5})$	They simulate the transferring of a flit from the queue Q_i to the jith register of the register matrix.
5	Q$ji$2 $(i,j=\overline{1,5})$	They simulate the storing of a flit in the jith register of the register matrix for flits initially uploaded from the queue Q_i
		Group controllers
6	RC_i $(i=\overline{1,5})$	They select a row of the register matrix based on the routing algorithm μ (with equal probabilities in this study).
7	MC_j $(j=\overline{1,5})$	They select a register in the jth row of the register matrix to issue the flit in accordance with the algorithm φ (oldest flits are issued first in this study).
		Gates
8	K$ji$1 $(i,j=\overline{1,5})$	They enable/disable the selection of the queue Q_i before uploading its head flit to the jith register of the register matrix.
9	K$ji$2 $(i,j=\overline{1,5})$	They enable/disable the upload of the head flit of the queue Q_i to the jith register of the register matrix.
10	K$ji$3 $(i,j=\overline{1,5})$	They enable/disable the issuance of the flit stored in the jith register of the register matrix to the output O_j.
11	K_LOAD_i $(i=\overline{1,5})$	They enable/disable the transfer of flits from the queue Q_i to the register matrix (HOL blocking).
12	K_LOST_i $(i=\overline{1,5})$	They enable/disable the reception of incoming flits for the queue Q_i (buffer overflow).

The group controllers RCx and MCx are key elements of the Q-chart as well. While the RCx controls model the supported routing algorithm μ (operating like column-wise random selectors), the MCx elements reflect the implemented row-wise flit scheduling policy. To switch to a different scheduling scheme, the MCx controls need to be reconfigured (internally, this means a different C++ subroutine is selected to manage the element).

Figure 5 shows graphs representing the throughput *versus* the number of inputs/outputs dependencies for the PPIRF switch and the buffered crossbar peers implementing the uniform and the round robin row-wise scheduling policies resulting from our simulation studies based on the Q-chart of Figure 4. Here only low-size switches, with $5 \le n \le 25$, are considered (we assumed that the cases $n < 5$ are practically unfeasible for the multiprocessors of the class under consideration).

By analyzing the obtained graphs, it was found that the throughput of the PPIRF switch has the lower bound of approximately 0.78, which was observed to grow smoothly as the number of inputs/outputs of the switch increases. Furthermore, it was found that short-term traffic boosts have no significant effect on switch throughput. Yet, the PPIRF switch was shown to have about 1.2–2.5% higher throughput compared to the buffered crossbar switches controlled by the uniform and the round robin scheduling mechanisms.

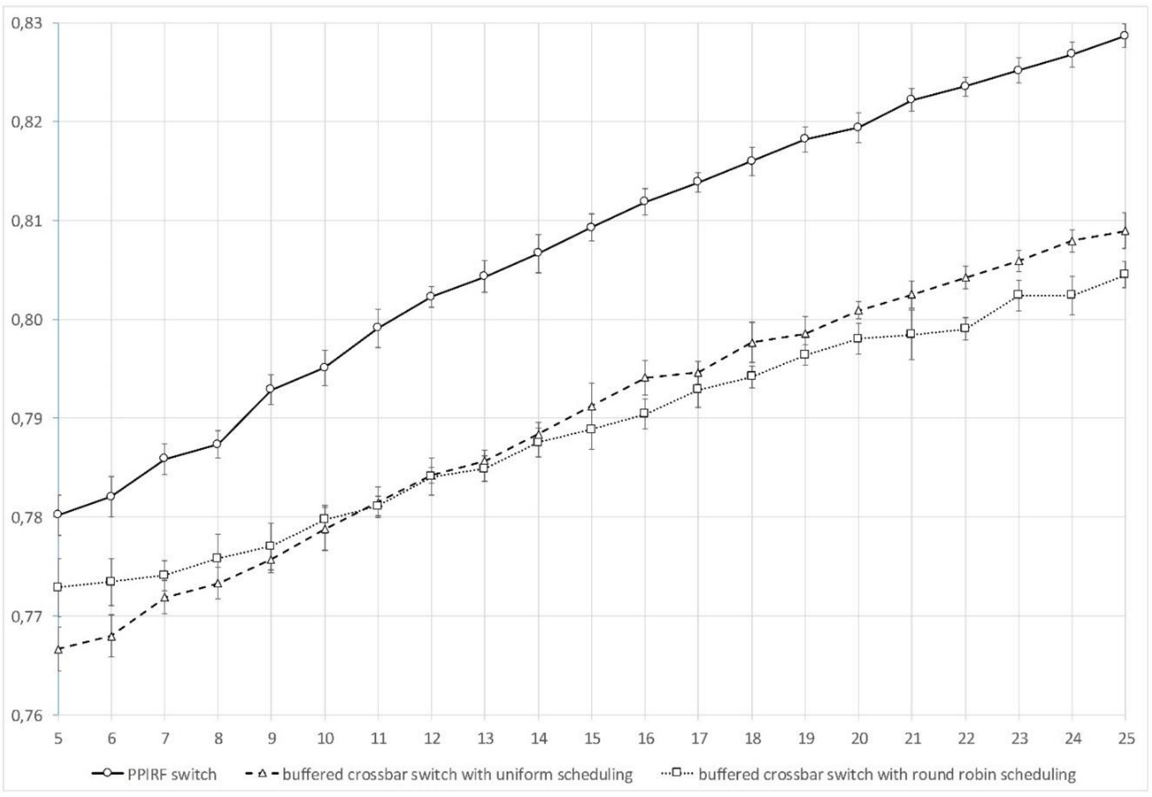

Figure 5. Throughput *versus* the number of inputs/outputs graphs for the low-size PPIRF switch and the buffered crossbar switches with the uniform and round robin flit scheduling (the confidence interval is shown for $\alpha = 0.02$).

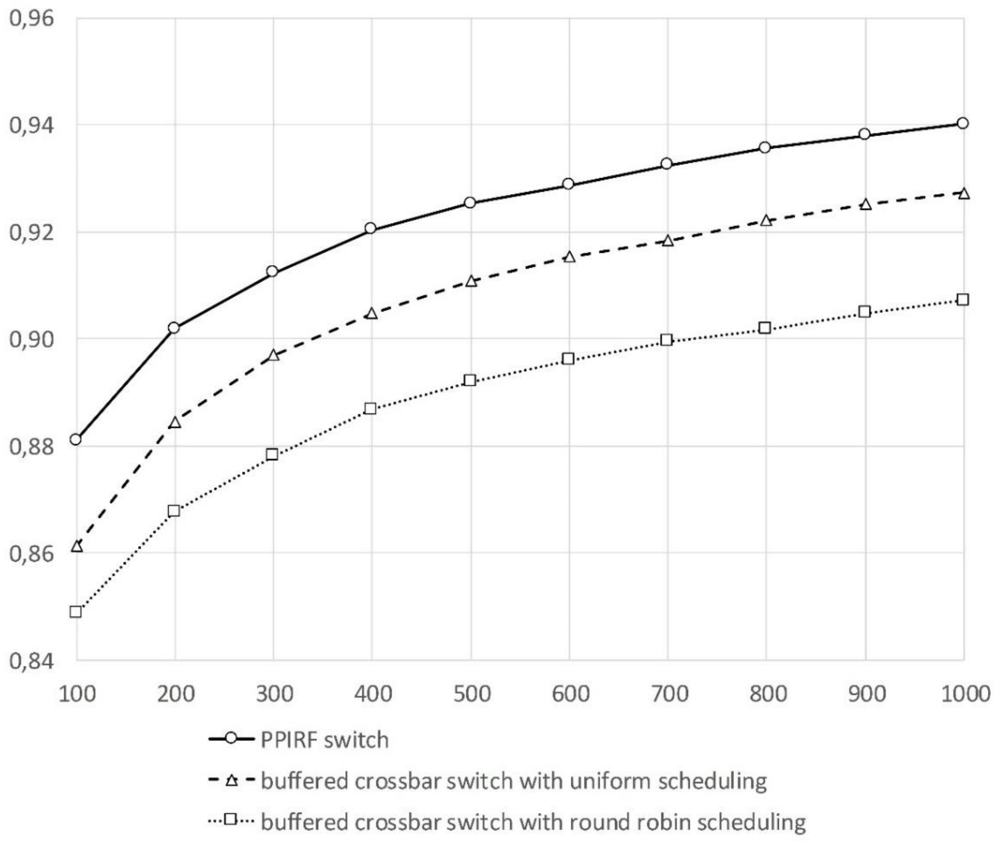

Figure 6. Throughput *vs.* number of inputs/outputs graphs for the high-size PPIRF switch and the buffered crossbar switches with the uniform and round robin flit scheduling.

We additionally conducted a series of simulation studies considering high-size switches with $100 \le n \le 1000$. Figure 6 shows graphs representing the throughput versus the number of inputs/outputs dependency for high-size switches obtained from the simulation studies based on the Q-chart of Figure 4. Since the confidence interval is less than 1% of the throughput average with $\alpha = 0.02$, error bars are not shown in Figure 6. Notwithstanding, it was observed that the throughput of the PPIRF switch exceeds 90% starting at $n \approx 190$; asymptotically, it eventually becomes 100%. This is because for higher values of *n*, the HOL blocking probability P_i^{BL} smoothly decreases to zero. According to Figure 6, the PPIRF switch has approximately 2.5–3.2% and 3.5–3.8% higher throughput compared to the buffered crossbar switches with uniform and round robin flit scheduling, respectively.

Figure 7 presents graphs reflecting the HOL blocking probability *versus* the Bernoulli distribution parameter *p* dependency for PPIRF switches with *n* input/output trunks (the error bars in this figure are not shown, since they are less than 1% of the average). The graphs validate that the probability P_i^{BL} decreases with an increase in the number of inputs/outputs *n*, with the most significant decrease occurring at the maximum intensity of incoming flit streams. For instance, in the case of a 10×10 PPIRF switch, the probability is about 1.62 times lower than for a 5×5 switch. However, for a switch having 15 inputs/outputs, the specified probability is about 1.32 times less. For a 20×20 switch, it decreases by about 1.22 times. Further, for a switch with 25 inputs/outputs, a decline of about 1.17 times was observed. Further simulation study conducted for large-sized switches revealed a further decrease in the probability P_i^{BL}. With $n = 100$, the maximum probability is at approximately 0.04, while with $n = 500$ it is at approximately 0.013 and with $n = 1000$, it is at about 0.0081.

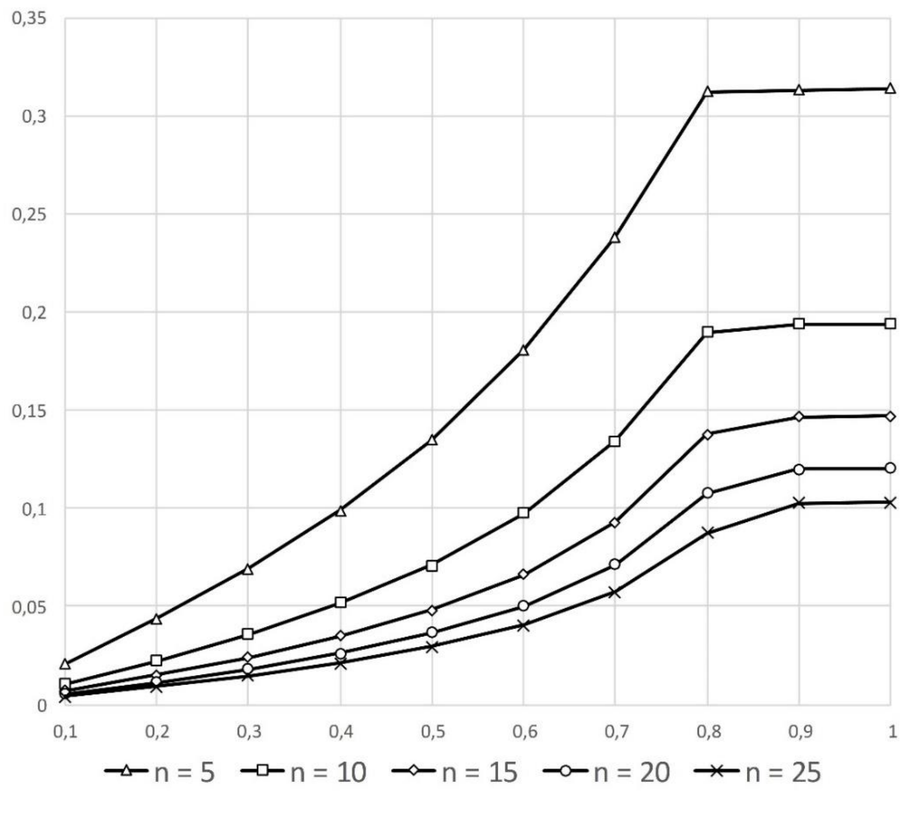

Figure 7. The HOL blocking probability *versus* the Bernoulli distribution parameter graphs for a PPIRF switch having $5 \le n \le 25$.

The reason why P_i^{BL} behaves like it is stated above is that the probability (referred to as p_1) for a flit to be transferred to a particular target register of the register matrix decreases as *n* grows; in addition, the decrease is non-linear. For example, while $p_1 = 0.2$ for a 5×5 switch, it goes down to as low as 0.04 for a 25×25 one (if uniform distribution is assumed and $p_1 = 1/n$ therefore). Assuming p_2 to be the probability that a register of an arbitrary column is not empty (still contains another flit), we obtain $P_i^{\mathrm{BL}} = p_1 p_2$; i.e., $P_i^{\mathrm{BL}} = p_2/n$ in the case of uniform distribution. Hence, P_i^{BL} will go down with *n* whatever value of p_2 is given. Our simulation has confirmed that p_2 is significantly less than 1 for any admissive traffic patterns.

Based on extra simulation experiments, we studied the delay of the PPIRF switch and compared it to that of the buffered crossbar switches with uniform and round robin flit scheduling (measured in time $n = 25$ slots, with a time slot corresponding to the switch clock pulse period duration). Figure 8 presents the corresponding delay *versus* the Bernoulli distribution parameter *p* graphs for $n = 5$, $n = 15$ and (the error bars in this figure are not shown for the sake of simplicity). According to the graphs, the proposed architecture has no speed advantage over the peers for $p \leq 0.6$; however, as the incoming traffic grows heavier, the PPIRF switch demonstrates better performance in terms of delay. Moreover, the greater the number of input/output trunks, the higher the difference between the compared architectures. Assume $p = 0.8$, which is rather heavy traffic. In this case, a 5×5 buffered crossbar switch with the uniform scheduling policy shows an average delay of as high as 232 time slots, a 5×5 buffered crossbar switch with round robin scheduling is capable to relay a flit in 177 time slots on the average, while a 5×5 PPIRF switch has a delay of as low as approximately 141 time slots. For 15×15 switches, the delay mean values are significantly lower owing to decreased HOL blocking probability and are equal to 84, 89 and 18 time slots, respectively. And finally, for 25×25 devices, the average delays are as low as 29, 34 and 7 time slots, respectively. In general (for any admissive *n*), the PPIRF architecture is a faster solution than the other switches considered in this study in the case of heavy traffic ($p \geq 0.7$).

5. A Summary of Comparison of The Proposed Architecture to VOQ-based Switches

A comparative study of the PPIRF approach *versus* the latest VOQ-based (including crosspoint-buff ered) approaches is of significant interest as well. Some VOQ-based switches are known to be stable; thus, they can achieve up to 100% throughput for any admissible traffic. At the same time, they have a hardware complexity of at least $O\left(n^2 L\right)$ (*n* FIFO buffers of length *L* at each of *n* input trunks) and, therefore, the implementation cost may not be acceptable for multiprocessors of the considered class.

Additionally, VOQ-based approaches with no crosspoint buffers require a speedup of about two, which increases the implementation cost and decreases the potentially reachable external speed of the switch. Table 2 presents a comprehensive summary of the comparison results (uniformly distributed traffic is assumed).

6. Conclusions and Future Directions

In this paper, we have proposed a parallel pipelined flit switch architecture for mesh-connected multiprocessors with independent flit routing (the PPIRF switch) based on non-VOQ input FIFO buffers and an output register matrix supporting a novel row-wise oldest-flit-first flit scheduling policy. *Via* a comprehensive simulation study, it was found that the proposed approach allows achieving a throughput of at least 78% for practically significant scenarios and up to 100% throughput asymptotically (as $n \to \infty$) with no internal speedup and square hardware complexity in contrast to existing VOQ-based switch architectures. Due to its lower implementation complexity, the PPIRF switch is suitable for large-scale designs and complex network topologies, such as twisted torus, crossed cube, to mention but a few (e.g., see [34]-[35]), requiring many input/output trunks at each node. Compared to known non-VOQ switches with crosspoint buffers, the proposed device demonstrates slightly (1.2–3.8%) higher throughput and substantially (up to several times) shorter delays under heavy traffic patterns.

In the future, it is important to study how to increase the throughput of low-size PPIRF switches by adding some modifications to the scheduling policy to alleviate the influence of the HOL blocking probability and how to efficiently implement our methodology in practical switches taking into account state-of-the-art limitations. In addition, it would be interesting to study the multiprocessor behavior when applying our algorithm to the real communication networks which deliver real-world traffic.

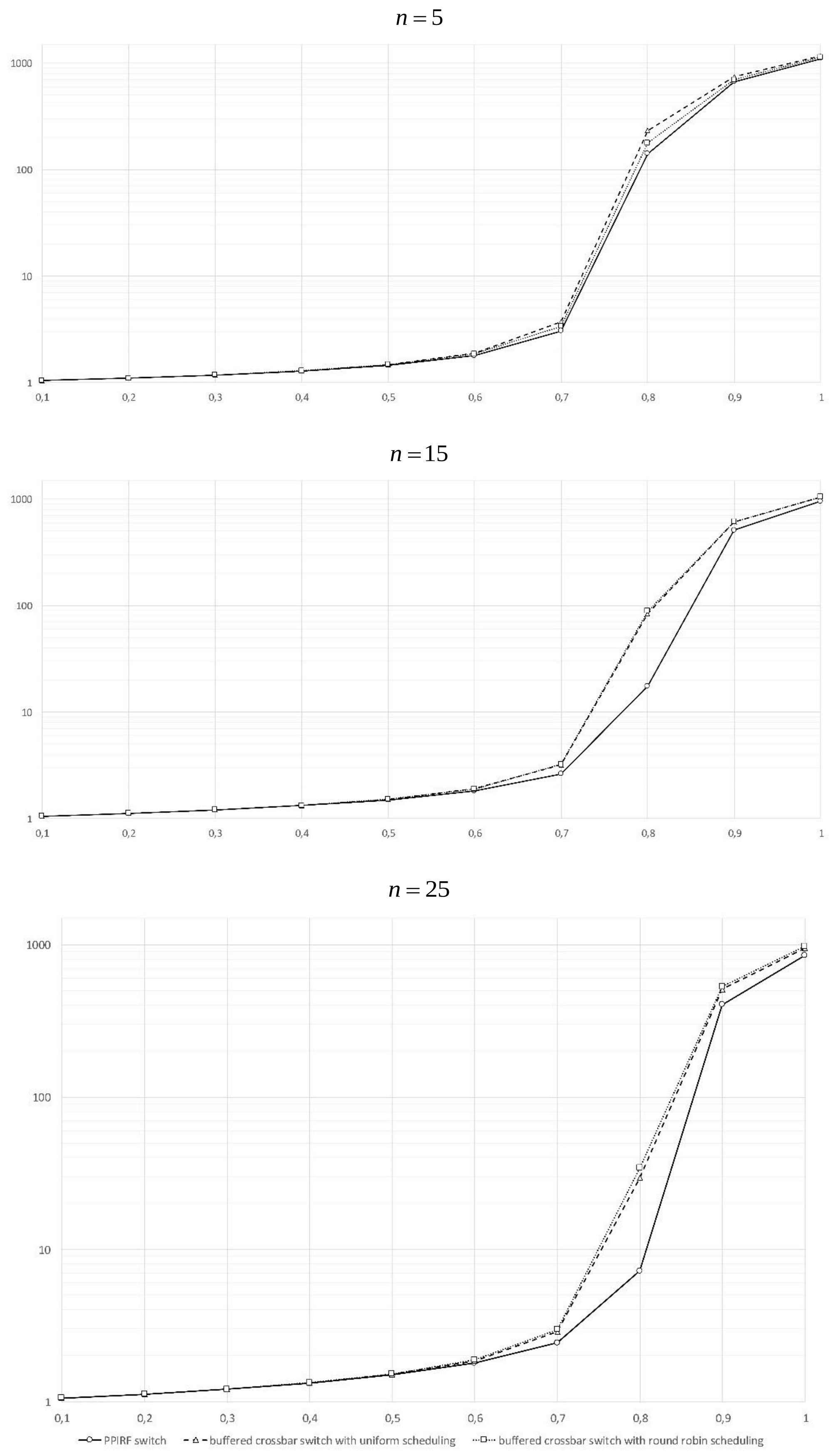

Figure 8. Delay *versus* the Bernoulli distribution parameter graphs for the low-size PPIRF switch and the buffered crossbar switches with the uniform and round robin flit scheduling.

Table 2. Summary of the comparison of the proposed switch architecture with several state-of-the-art architectures.

Switch architecture	Throughput (uniformly distributed traffic)	Hardware complexity	Speedup
iSLIP scheduled VOQ switch [19]	Up to 100% asymptotically	$O\left(n^2L\right)$	2
SQUID VOQ crosspoint-buffered switch [20]	Up to 100%	$O\left(n^2L\right)$	1 (not required)
RR/LQF scheduled VOQ switch [24]	Up to 100%	$O\left(n^2L\right)$	$2-\frac{1}{n}$
Proposed IRF switch architecture	At least 78% for practical scenarios and up to 100% asymptotically	$O\left(\max\left\{n^2,nL\right\}\right)$	1 (not required)

ACKNOWLEDGEMENTS

The authors would like to thank the Editor-in-Chief Prof. Ahmad Hiasat and four anonymous reviewers for their detailed and insightful comments, which helped to significantly improve the quality of the paper.

REFERENCES

[1] S. Misra and S. Goswami, Network Routing: Fundamentals, Applications and Emerging Technologies, Wiley Telecom, 2014.

[2] A. A. Jerraya and W. Wolf, Multiprocessor Systems-on-Chips, San Francisco, Elsevier, Inc., 2005.

[3] Z. Yu, R. Xiao et al., "A 16-Core Processor with Shared-Memory and Message-Passing Communications," IEEE Trans. Circ. Syst. I: Regular Papers, vol. 61, no. 4, pp. 1081-1094, 2014.

[4] Tilera Corp., "Tile Processor Architecture Overview for The TILE-Gx Series," [Online], Available:

http://www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf, (access date: 30.06.2019).

[5] A. Olofsson, "Epiphany-V: A 1024 Processor 64-bit RISC System-on-Chip," [Online]: Available:

https://www.parallella.org/docs/e5_1024core_soc.pdf, (access date: 30.06.2019).

[6] P. Lotfi-Kamran, M. Modarressi and H. Sarbazi-Azad, "An Efficient Hybrid-Switched Network-on-Chip for Chip Multiprocessors," IEEE Trans. Comput., vol. 65, no. 5, pp. 1656-1662, 2016.

[7] G. Chen, M. A. Anders et al., "A 340 mV-to-0.9 V 20.2 Tb/s Source-synchronous Hybrid Packet/Circuit-switched 16×16 Network-On-Chip in 22 nm Tri-Gate CMOS," IEEE J. Solid-St. Circ., vol. 50, no. 1, pp. 59-67, 2015.

[8] A. Mazloumi and M. Modarressi, "A Hybrid Packet/circuit-switched Router to Accelerate Memory Access in NoC-based Chip Multiprocessor," Proc. of Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 908-911, 2015.

[9] M. H. Foroozannejad, M. Hashemi et al., "Time-scalable Mapping for Circuit-switched GALS Chip Multiprocessor Platforms," IEEE Trans. Comput.-aided Design of Integr. Circ. and Syst., vol. 33, no. 5, pp. 752-762, 2014.

[10] M. Karol, M. Hluchyj and S. Morgan, "Input *Versus* Output Queueing on a Space-Division Packet Switch," IEEE Trans. Commun., vol. 35, no. 12, pp. 1347-1356, 1987.

[11] L. Deng, W. S. Wong et al., "Delay-constrained Input-queued Switch," IEEE J. Selected Areas Commun., vol. 36, no. 11, pp. 2464-2474, 2018.

[12] K. Kang, K.-J. Park, L. Sha and Q. Wang, "Design of a Crossbar VOQ Real-time Switch with Clock-

driven Scheduling for a Guaranteed Delay Bound," Real-time Systems, vol. 49, no. 1, pp. 117-135, 2013.

[13] M. J. Neely, E. Modiano and Y. -S. Cheng, "Logarithmic Delay for NxN Packet Switches under the Crossbar Constraint," IEEE/ACM Trans. Networking, vol. 15, no. 3, pp. 657-668, 2007.

[14] S. Durkovic and Z. Cica, "Birkhoff-von Neumann Switch Based on Greedy Scheduling," IEEE Comput. Archit. Letters, vol. 17, no. 1, pp. 13-16, 2018.

[15] C. -S. Chang, D. -S. Lee and C. -Y. Yue, "Providing Guaranteed Rate Services in the Load Balanced Birkhoff-von Neumann Switches," IEEE/ACM Trans. Networking, vol. 14, no. 3, pp. 644-656, 2006.

[16] H. -I. Lee and S. -W. Seo, "Matching Output Queueing with a Multiple Input/Output-queued Switch," IEEE/ACM Trans. Networking, vol. 14, no. 1, pp. 121-132, 2006.

[17] Y. Tamir and G. Frazier, "High Performance Multi-queue Buffers for VLSI Communication Switches," Proc. of the 15th Annu. Symp. Comput. Archit., pp. 343-354, June 1988.

[18] T. Anderson, S. S. Owicki, J. B. Saxe and C. P. Thacker, "High-speed Switch Scheduling for Local-area Networks," ACM Trans. Comput. Syst., vol. 11, no. 4, pp. 319-352, 1993.

[19] N. McKeown, "The iSLIP Scheduling Algorithm for Input-queued Switches," IEEE/ACM Trans. Networking, vol. 7, no. 2, pp. 188-201, April 1999.

[20] Y. Shen, S. S. Panwar and H. J. Chao, "SQUID: A Practical 100% Throughput Scheduler for Crosspoint Buffered Switches," IEEE/ACM Trans. Networking, vol. 18, no. 4, pp. 1119-1131, August 2010.

[21] N. McKeown, V. Anantharam and J. Walrand, "Achieving 100% Throughput in an Input-queued Switch," Proc. of the 15th IEEE INFOCOM, pp. 296-302, San Francisco, CA, USA, Mar. 1996.

[22] J. Chao, "Saturn: A Terabit Packet Switch Using Dual Round Robin," IEEE Commun. Mag., vol. 38, no. 12, pp. 78-84, Dec. 2000.

[23] S. Mneimneh, "Matching from the First Iteration: An Iterative Switching Algorithm for an Input-queued Switch," IEEE/ACM Trans. Networking, vol. 16, no. 1, pp. 206-217, Feb. 2008.

[24] B. Hu, K. L. Yeung, Q. Zhou and C. He, "On Iterative Scheduling for Input-queued Switches with a Speedup of 2-1/N," IEEE/ACM Trans. Networking, vol. 24, no. 6, pp. 3565-3577, 2016.

[25] B. Hu and K. L. Yeung, "Feedback-based Scheduling for Load Balanced Two-Stage Switches," IEEE/ACM Trans. Networking, vol. 18, no. 4, pp. 1077-1090, Aug. 2010.

[26] C. -S. Chang, D. -S. Lee and Y. -S. Jou, "Load Balanced Birkhoff-von Neumann Switches, Part I: One-stage Buffering," Comput. Commun., vol. 25, no. 6, pp. 611-622, 2002.

[27] Y. Chen, "Cell Switched Network-on-Chip Candidate for Billion-transistor System-on-Chips," Proc. of IEEE Int'l. Soc. Conf., pp. 57-60, 2006.

[28] A. Olofsson, "Mesh Network," US Patent, no. 8531943 B2, Sep. 10, 2013.

[29] M. Lin and N. McKeown, "The Throughput of a Buffered Crossbar Switch," IEEE Commun. Let., vol. 9, no. 5, pp. 465-467, 2005.

[30] M. Nabeshima, "Performance Evaluation of a Combined Input- and Crosspoint-queued Switch," IEICE Trans. Commun., vol. E83-B, pp. 737-741, 2000.

[31] I. V. Zotov, "Distributed Virtual Bit-slice Synchronizer: A Scalable Hardware Barrier Mechanism for N-dimensional Meshes," IEEE Trans. Comput., vol. 59, no. 9, pp. 1187-1199, Sep. 2010.

[32] I. V. Zotov et al., "The VisualQChart Simulation Environment," Computer Program Certificate RU 2007611310, appl. 13.02.2007, publ. 27.03.2007.

[33] D. Zydek, H. Selvaraj and L. Gewali, "Synthesis of Processor Allocator for Torus-based Chip Multiprocessors," Proc. of the 7th Int'l. Conf. on Information Technology: New Generations, pp. 13-18, 2010.

[34] A. Samad, M. Q. Rafiq and O. Farooq, "Performance Evaluation of Task Assignment Algorithms in Cube-based Multiprocessor Systems," Proc. of the 1st Int'l. Conf. on Emerging Trends and Applications in Computer Science, pp. 48-51, 2013.

[35] J. M. Camara, M. Moreto et al., "Twisted Torus Topologies for Enhanced Interconnection Networks," IEEE Trans. Parallel Distrib. Syst., vol. 21, no. 12, pp. 1765-1778, 2010.

[36] K. Li, Y. Mu, K. Li and G. Min, "Exchanged Crossed Cube: A Novel Interconnection Network for Parallel Computation," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 11, pp. 2211-2219, 2013.

[37] J. Al Azzeh, "Distributed Mutual Inter-unit Test Method for D-Dimensional Mesh-connected Multiprocessors with Round-Robin Collision Resolution," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 05, no. 01, pp. 1-16, April 2019.

[38] J. Al Azzeh, "Improved Testability Method for Mesh-connected VLSI Multiprocessors," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 04, no. 02, pp. 116-128, August 2018.

[39] J. Al Azzeh, "Fault-tolerant Routing in Mesh-connected Multicomputers Based on Majority-operator-produced Transfer Direction Identifiers," Jordan Journal of Electrical Engineering, vol. 03, no. 02, pp. 102-111, April 2017.

ملخص البحث:

في هذه الورقة، يتم اقتراح معمارية لمفتاح رُزَم لمعالجات متعددة متصلة بعضها ببعض في شكل شبكة، وذلك بناءً على استخدام مجموعة من مصدّات الإدخال ومصفوفة تسجيل في المخرج بحيث يجري التحكم في عمل المفتاح عن طريق مخطط جدولة يستند الى التوزيع الزمني. يتم استخدام نهج ثابت وبسيط من أجل عُبور المفتاح، بحيث يجري تقسيم كل رُزمة الى مجموعة من الأجزاء التي تُوجَّه في مساراتٍ مستقلة. الجهاز المقترح يحقق نسبة عبور لا تقل عن 78% للعبور الموزّع بانتظام، ويمكن أن تقارب في حدّها الأعلى نسبة 100%. ولدى مقارنة الجهاز المقترح في هذه الدراسة بمثيلاته من مفاتيح الرُزَم التقليدية، وجدنا أن المفتاح المقترح يصل الى نسبة العبور القصوى له دون الحاجة الى زيادة السرعة الداخلية، ناهيك عن أنه أقل تعقيداً من حيث المعدات المستخدمة. وبالمقارنة مع مفاتيح تقليدية استُخدمت فيها آليات عُبور نمطية، تبين أن المفتاح المقترح يمتلك نسبة عبور أعلى بقليل، وأنّ التأخير فيه هو أقل بشكل جوهري في بعض الحالات ذات الأهمية العلمية. فقد تراوحت نسبة تفوق النموذج المقترح بين 1.2% و 3.8%، في حين كان التأخير أقصر بحوالي 5 مرات في بعض الحالات.