

Experimental survey of FPGA-based monolithic switches and a novel queue balancer

Philippos Papaphilippou, Kentaro Sano, Boma A. Adhi, Wayne Luk

Abstract—This paper studies small to medium-sized monolithic switches for FPGA implementation and presents a novel switch design that achieves high algorithmic performance and FPGA implementation efficiency. Crossbar switches based on virtual output queues (VOQs) and variations have been rather popular for implementing switches on FPGAs, with applications in network switches, memory interconnects, network-on-chip (NoC) routers etc. The implementation efficiency of crossbar-based switches is well-documented on ASICs, though we show that their disadvantages can outweigh their advantages on FPGAs. One of the most important challenges in such input-queued switches is the requirement for iterative scheduling algorithms. In contrast to ASICs, this is more harmful on FPGAs, as the reduced operating frequency and narrower packets cannot “hide” multiple iterations of scheduling that are required to achieve a modest scheduling performance. Our proposed design uses an output-queued switch internally for simplifying scheduling, and a queue balancing technique to avoid queue fragmentation and reduce the need for memory-sharing VOQs. Its implementation approaches the scheduling performance of a state-of-the-art FPGA-based switch, while requiring considerably fewer resources.

Index Terms—FPGA, switch, virtual output queues, output-queued, crossbar, scheduling algorithms, queue balancing, NoCs

1 INTRODUCTION

Full-interconnection could be considered as one of the fundamental and most challenging problems in computer science, appearing from low-level circuits to higher-level applications including neural networks. This is mainly because the possible paths in a fully-interconnected system are $O(P^2)$, where P is the number of inter-connected entities. The scaling of this number directly impacts both software and systems engineering, and different techniques exist for attempting to make the best use of silicon and cycles.

Switch designs can be an integral part in FPGA designs. They are used inside the programmable logic for accessing different memories and peripherals from multiple workers [1], as well as for interfacing with input/output devices, such as for implementing network switches or stacks [2], or communicating with other FPGAs [3]. Even with hierarchical switch designs, efficient switch architectures of a lower radix (number of ports) are important as building blocks [4].

FPGAs are increasingly being used as accelerators in high-performance (HPC) [5] and edge computing, and they require implementations of high-end switches. In particular, recent advanced FPGAs have multiple high-bandwidth memories to provide high throughput, though the coordination of their numerous ports is left to the fabric [6]. Computing modules including fast Fourier transform [7] usually access those memory ports through multiple initiator ports. In our case, there are ongoing projects that motivated this paper. These include Riken’s CGRA evaluation framework

that is currently based on an FPGA cluster connected to the supercomputer Fugaku [8]. While well-performing configurable switches exist in the application-specific integrated circuit (ASIC) world [9], they are rarely combined with high-end FPGAs, let alone with HPC-related features.

The main principle behind the research in high-performance switching in hardware is to try to temporarily rearrange the packets in time, to optimise the efficiency of more primitive interconnects such as crossbars. In order to achieve scalability for a higher number of entities, hierarchical approaches try to eliminate hardware complexity from centralised complex switch architectures. Such approaches also include network-on-chip (NoC) designs [10], which still rely on switches of a lower radix (routers), with an overhead in performance when approaching full connectivity [11].

In this paper we present an FPGA-based switch architecture that exhibits high scheduling performance, while having similar FPGA design attributes as simpler designs. Our target specification is low to medium radix (number-of-port) switches with high-scheduling performance, and an efficient full-throughput (output-per-cycle) FPGA implementation.

The list of contributions of the paper is as follows:

- 1) A novel high-performance switch architecture that achieves algorithmic performance similar to the optimal for the studied traffic, while using a fraction of the resources of a competing approach on FPGAs [12], [13].
- 2) A queue balancing technique to avoid queue fragmentation for a more efficient use of the available memory resources, with a low hardware overhead.
- 3) A systematic FPGA-focused comparison (experimental survey) with existing switch architectures using simulations and implementation on real hardware (sections 5 and 6). All source code is open-sourced.

This paper is an extension of the paper “Efficient Queue-Balancing Switch for FPGAs” [14]. Additional material in-

- The authors Philippos Papaphilippou and Wayne Luk were with the Department of Computing, Imperial College London, UK (E-mail: p.papaphilippou17@alumni.imperial.ac.uk, w.luk@imperial.ac.uk). Philippos is now with Huawei Technologies R&D (UK) Limited.
- The authors Kentaro Sano and Boma A. Adhi are with RIKEN Center for Computational Science, Kobe, Japan (E-mail: {kentaro.sano, boma.adhi}@riken.jp)

cludes a more thorough simulation-based evaluation (section 5), such as with the inclusion of four additional traffic models, experiments on NoC adaptation, a more-detailed design space, and a complexity analysis (section 6) elaborating on the behaviour of the FPGA-based implementations.

2 BACKGROUND

2.1 Crossbar

The crossbar is a simple and popular interconnect for both FPGAs and ASICs. It consists of wires having crosspoints everywhere there is an input-output port combination, resulting in a crosspoint complexity of $P_I \times P_O$, where P_I is the number of input ports and P_O the number of output ports. On the crosspoints there are smaller “switches” which are logically equivalent to 2-to-1 multiplexers. On FPGAs, crossbars can logically be implemented with a $P_I - to - 1$ multiplexer per output port [15].

Crossbars and other primitive interconnects are considered “non-blocking”, as long as any configuration is supported without backpressure. However, this is only with regard to all possible configurations, such as with destinations being a passable permutation of the output ports. “Passable” means the inter-connect is capable of performing the permutation. However, when two ports send a packet to the same port, either one input is blocked or a packet is dropped according to the switch requirements.

The latest research on FPGA-based crossbar implementation focused on multi-stage alternatives. These are interconnects with similar functionality, including implementing permutation networks, such as with butterfly networks [6], sorting networks [16]. Another category of related research is about the crossbar efficiency when building routers for network-on-chip (NoCs) on FPGAs [11], [17]. Though, the latter category often overlaps with the design of scheduling algorithms according to the system requirements [18], as when using the following crossbar-based switch design.

2.2 Input-Queued Crossbar

The input-queued switch uses virtual output queues (VOQs) before the crossbar. There are $P_I \times P_O$ queues, where P_I is the number of inputs and P_O the number of outputs, to allow temporarily holding any incoming packets without collisions. Each VOQ corresponds to every input-output combination. VOQs are a popular workaround for the head-of-line blocking, which avoids blocking from such collisions to a certain degree, according to the traffic and queue length.

Figure 1 shows how VOQs are used in high-level for a switch of 4 input ports. A scheduling algorithm is responsible for the dequeuing decisions. One advantage of the input-queued crossbar is that each group of VOQs can be represented by a single memory, because there is up to one write and up to one read from each group per cycle.

Alternatively, input-queued switches are also found in NoCs, where instead of VOQs there can be a specified number of queues per input port. These queues represent virtual channels (VC), which are used as a more flexible abstraction than VOQs to support additional functionalities. These include avoiding deadlocks in wormhole switching

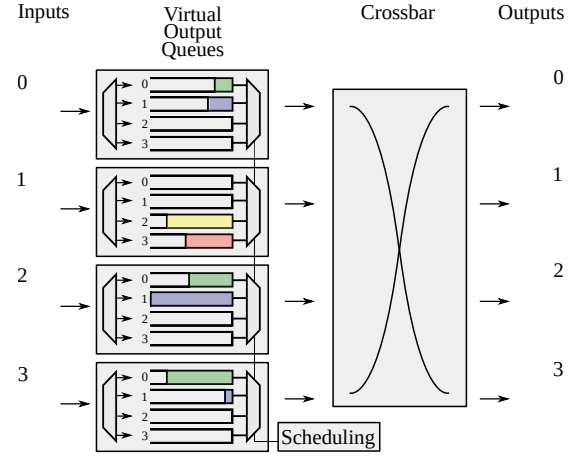


Fig. 1. Input-queued switch with VOQs

[10] (according to the behaviour of the routing algorithm), and enforcing quality of service (QoS) by traffic classification [19]. These could also be considered advantages of the approach, although they relate mostly to the application requirements.

The input-queued switch is one of the most popular switch architectures on FPGA designs. There are works that focus on the implementation efficiency of its scheduling algorithm [20], [21], [22], as well as its memory-sharing potential [4], [23]. The research on scheduling algorithms for VOQs has been rather thorough outside the FPGA domain, due to the importance of scheduling performance and complexity, as well as the performance requirements under different traffic types. Popular scheduling algorithms include PIM [24], iSLIP [25], DRRM [26] and EDRRM [27].

2.3 Output-Queued Crossbar

The output-queued crossbar is considered the best performing switch algorithmically [28]. As illustrated in the example of figure 2, there is a crossbar and only one queue per input port, however, these operate at P_I times the base operating frequency. This speedup (or equivalent workarounds) is necessary to be able to serve all requests one-by-one, and essentially remove HOL blocking from the crossbar. This is widely accepted as expensive and non-scalable [13], [29].

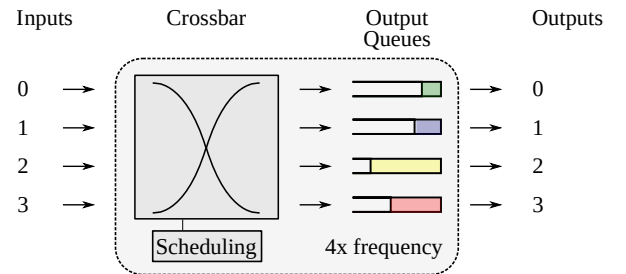


Fig. 2. Output-queued crossbar switch

There are optimisations that use a lower speedup [28]. Still, requiring a logic speedup is less desirable on FPGAs [12], due to the restricted operating frequency when compared to ASICs overall, as well as the more homogeneous

timing behaviour across the different FPGA resources [1], [30]. Hipernetch [12] is a relatively recent FPGA-based solution that uses a fully-pipelined structure that emulates an output-queued crossbar, with its high resource utilisation as its main drawback.

2.4 Output-Queued Switch (Without Memory Sharing)

A simpler variation of the latter design is the output-queued switch, which achieves the same performance as the output-queued crossbar in terms of average packet latency, without any speedup. There are output queues in the same organisation as virtual output queues, i.e. each port de-multiplexes the packet to P_O queues according to its destination. The difference to VOQs is that the output arbiters multiplex between queues destined directly for the output port, rather than between queues of the same input port. This is visualised in figure 3.

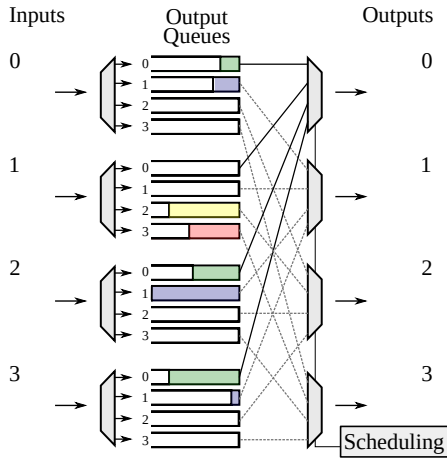


Fig. 3. Output-queued switch

One consideration has to do with the memory organisation of the switch, which becomes a concern for scalability, as some configurations require $P_I \times P_O$ queues. Note that in literature, an output-queued switch usually refers to also having a memory sharing technique.

When compared to the input-queued switch, one notable advantage of this approach is that it simplifies the scheduling complexity, as the arbiter decisions are independent of one another. This is because no crossbar is involved for limiting to passable permutations once the packets are stored in the queues.

On FPGAs, this is currently not a common switch architecture [4]. However, an equivalent generalised approach is used in router designs on FPGA-based NoCs [31], also known as the split-merge switch [32].

3 CHALLENGES

The challenges when selecting a switch architecture for FPGA implementation often appear as trade-offs.

C1. Full-Interconnection

Ideally, a distributed/parallel system would enable all entities to communicate with one another directly. However, the number of links required, as well as the total logic

complexity grows exponentially with the number of entities. Hence, network topologies enforce a multi-hop approach to emulate full-interconnection at the expense of increased latency and unfairness. In practice, this happens in all scales of computing, from the individual gates that only have a binary nature, to today's internet that consists of distributed nodes organised in a hierarchical structure [33].

In our context, the notion of full interconnection in FPGA switches is limited to those that can forward any input to any output simultaneously in a fixed latency of one or a few pipelined FPGA cycles under ideal conditions such as empty queues. As described, full interconnection is challenging to support for a high number of ports, hence the hierarchical approaches aiming at scalability [4]. While the main logic of the presented monolithic switches can be seen as combinational circuits, what happens inside the FPGA under the specified cycle can vary from implementation to implementation and from FPGA to FPGA due to the differences in internal architectures and toolchain heuristics. Thus, the effects of logic mapping are explored indirectly through implementation.

An analogy to a challenging aspect in FPGA-based and ASIC switches would be the hardened routers inside the FPGA fabric and the number of metal layers in ASIC dies. They are both used to implement wires, but exhibit restrictions in connectivity including the wire directions. FPGA routers are ultimately switches [34], and are based on multi-layer dies [35]. Still, a direct comparison of silicon-based switches and FPGA-based switches is non-trivial, as there are different technologies available to silicon but not on FPGAs. Some examples include the availability of pass gates or equivalents in silicon that simplify crossbar implementations [36], the higher operating frequencies and the fact that multiplexers are considered expensive on FPGAs [37], even though their fabric is based on hardened ones.

Full interconnectivity can sometimes be discounted according to the task requirements. For instance, systolic arrays only require a 2D mesh-structure for communication between the compute units [38]. There are also techniques to shrink network topologies and increase fairness [39], as well as additional techniques to localise and/or approximate [40] computation to reduce the packets and their flight time [41], [42]. These or similar approaches can appear at different scales, though they are outside the scope of this paper.

C2. Scheduling Performance

Different switch architectures try to temporarily rearrange the packets, so that on each cycle the inputs to the crossbar are in the form of passable permutations with respect to the port destinations.

The effectiveness of switches is measured in terms of packet latency, such as the average and worst-case latency, and depends on the scheduling decisions and other design choices of the switch. As the temporary reordering of the packets is usually done using memories, another important metric is packet loss rate, which depends on the queue size and organisation, as well as the scheduling decisions.

For example, one fundamental limitation of input-queued switches with virtual output queues (VOQs) is that only one packet per VOQ group can be extracted per cycle.

This limitation is also reflected in our simulation results when comparing input-queued switches with other architectures in section 5.

C3. Resource Utilisation

Especially on FPGAs, one important challenge is the implementation efficiency with regard to the resource utilisation (including look-up tables (LUTs) and flip-flop registers (FF)). One related consideration has to do with the memory organisation of the switch, which becomes a concern for scalability, as some configurations require $P_I \times P_O$ queues (quadratic for ($P_I = P_O$)), where P_I is the number of input ports and P_O the number of output ports. Section 6 elaborates on logic and space complexity.

C4. Memory Fragmentation

One challenge with switch designs is how efficiently the memory space is used. If the memory is used inefficiently, then this can impact the resource utilisation, as more memory will need to be available for certain performance.

This is present in both the output-queued switch and the input-queued switch (VOQs), as there is one FIFO per input-output combination. Thus, having only one possible destination per packet makes the memory utilisation to directly depend on the characteristics of the traffic. See sections 5.2 and 5.3 for examples on the impact of memory fragmentation using simulations.

C5. Critical Path

A long critical path can make an FPGA implementation fail the timing requirements, or otherwise run at a low operating frequency. For switch design this can translate into a mandatory reduction in throughput, or a degradation in other aspects of the design based on related trade-offs.

Certain switch designs require complex scheduling algorithms for achieving a modest scheduling performance. Such scheduling algorithms can easily contribute to the critical path of the design, especially for a growing number of ports. For instance, input-queued switches with virtual output queues (or virtual channels) are left with a maximal matching problem for a bipartite graph to be solved on-the-fly, and the available scheduling algorithms are generally expensive (see section 6). Such matching algorithms are usually iterative, meaning that they require multiple iterations to perform well and approximate maximal matching. Many of them are shown to converge after the $\log_2(P_I)$ th iteration [43], where P_I is the number of inputs.

On FPGAs, having a multi-cycle scheduling step [44] is undesirable. On hardened network switches this is not of a concern, such as with an ASIC implementation of iSLIP that reconfigures the crossbar once every 9 cycles [45]. This is because of the much higher operating frequency and the wider packets, which can be sent progressively in smaller flits. An equivalent FPGA implementation [23], although achieves a high operating frequency from splitting scheduling across multiple cycles, it reduces the throughput to only $1/3 + \log_2(P_I)$ [12].

4 A NOVEL QUEUE BALANCING SWITCH

Our solution uses an output-queued switch to simplify scheduling decisions (challenge C5) and improve scheduling performance (C1, C2), in combination with a queue balancing mechanism for overcoming memory fragmentation (C4) with an insignificant hardware overhead (C3).

One shortcoming of the output-queued switch is the reduced efficiency of the queues, which can cause fragmentation for certain traffic. For uniform Bernoulli arrivals this is not a problem, because the probability of each queue receiving a packet is equal among all queues, resulting in uniform queue occupancy. However, under uneven traffic such as with bursts, it is helpful to have a mechanism to balance the queues.

The idea is to add a rotator near the input ports, to create a round-robin effect for queue-balancing, such as from bursts. Note that there is no balancing guarantee, as it rotates all inputs based on a cycle counter, and alternative designs for randomising the input packets would also be appropriate. Figure 4 presents this approach in high-level.

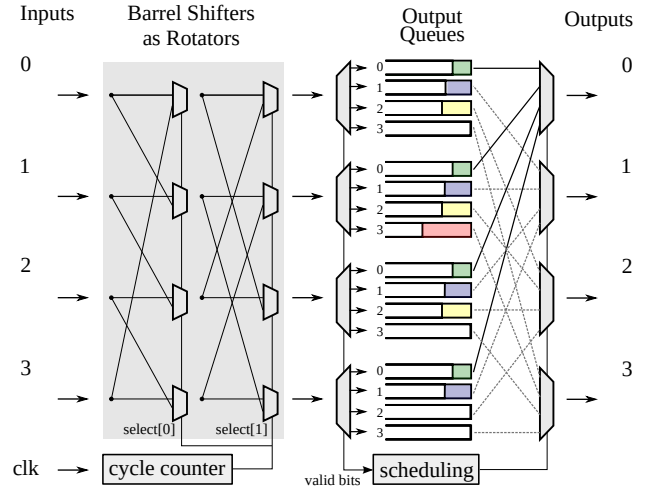


Fig. 4. Input rotation for load balancing

If plain round-robin arbiters are controlling the queue multiplexers, one issue would be that the order between packets arriving from a source to a destination is partially lost. This is because the rotation effect can land a packet in multiple queues (as the number of inputs (P_I)). Thus, a workaround is required that allows the arbiters to prioritise based on the arrival time.

The proposed workaround is introduced in figure 5. The required order information can be obtained by an additional set of P_O queues, each holding packets of P_I bits. Each packet is holding bits corresponding to the enqueue signals of each output queues, when grouped per output port rather than per input port (as per output-queuing) that have arrived on the same cycle. Each output arbiter holds the head packet of its corresponding queue until it extracts all packets that arrived together on that cycle, represented by this head packet. On each cycle, when no packet goes to a port, then no packet is stored in the valid bit queue of that port. In this way, the arbiters can expect at least one valid bit per valid bit packet, and no idle cycles or valid bit queue space is wasted for low input rates. The queue depth

requirement for each valid bit queue is $P_I * depth_{FIFO}$, to be able to handle the worst case of full utilisation with packets arriving on different cycles.

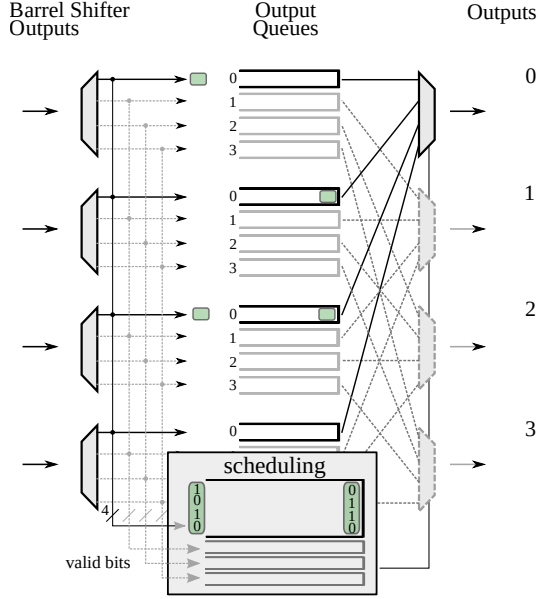


Fig. 5. Valid bit queues for reproducing the correct order after rotation

The idea of traffic randomisation near the inputs existed in the past [46], but it was for a more specialised use case. The main novelty here is the consulting of the input order by the scheduling module to reconstruct the correct order between the arriving packets that follow the same path, which widens its applicability on FPGAs to not only for asynchronous transfers, such as for supporting system interconnects. Network protocols such as TCP [47] are asynchronous by nature, however packet/flit reordering is still undesirable as the reordered packets can be more easily considered as dropped packets. In such cases, a high latency can yield a limited bandwidth overall as well as a hardware or software overhead to reconstruct the packet order [48]. If the packet order is not deemed useful, the scheduler part of the proposal can also be replaced with a simpler round-robin scheme, for example.

5 EVALUATION

In order to evaluate our solution and the queue balancing approach, we compare it against a selection of alternative monolithic/ non-hierarchical switch architectures¹. First, the scheduling performance is evaluated for different traffic patterns using simulations (sections 5.1 and 5.2). Then, a more system-oriented evaluation includes simulation for use in network-on-chips (sections 5.3). Finally, we proceed with FPGA implementation of a subset of the studied approaches (section 5.4), to investigate their resource requirement and timing characteristics.

This section also serves as the experimental survey, alongside section 6, which provides a high-level comparison on the practicality of the studied switches after also considering their theoretical complexities. This evaluation

also highlights the obtained improvements by using the proposed rotator solution on the output-queued switch. Hence, all parts of evaluation feature the output-queued switch, and the output-queued switch with a rotator.

For the simulation experiments (sections 5.1 to 5.3), low-level implementation details are abstracted, meaning that there is 0 processing latency, and the packet-size, operating frequency and other details are not used. With respect to the memory organisation we assume $P_I \times P_O$ independent queues for all switches.

For both simulation and implementation results, one of the compared switch architectures is Hipernetch [12], [13], which is a network switch implementation optimised for FPGA use, and almost functionally equivalent to the “output-queued crossbar”, generally featuring the best scheduling performance amongst the alternatives. This also requires $P_I \times P_O$ independent queues.

The switch simulations use the following traffic models. Based on the input rate $r \in [0, 1]$ number of input ports P_I , number of output ports P_O and $p_{i,j}$ the probability of input port $i \in \{0, 1, \dots, P_I\}$ to send to output port $j \in \{0, 1, \dots, P_O\}$:

- (a) *Uniform Bernoulli arrivals*: Each input port sends a packet to an output port with equal probability, i.e. $p_{i,j} = r/P_O$

This is the most common traffic model for comparing switches. It has a relatively desirable behaviour, as the packets are evenly distributed across the queues. Essentially, the proposal tries to emulate this model for the output-queued switch, so it is not expected to improve switching performance under this traffic.

- (b) *Uniform bursty traffic*: Each source port follows a state machine [12] that sends continuous bursts of a fixed number of consecutive packets (32 by default). Each $p_{i,j}$ is the same as above in the long run, as the overall distribution remains uniform, but it is not consulted in the decisions directly.

This model is also one of the most common traffic models in both the evaluation and real systems, and represents uneven traffic. It is similar to today’s internet, where it is said to work mostly in bursts (and because frames are usually broken down into multiple packets), as well as in system interconnects, where the data block size is usually greater than that of the switch. An example for the latter is when a 64-byte block is loaded into the cache through a 128-bit main memory interface in an x86 machine.

- (c) *Nonuniform – hotspot*: The hotspot model [27], [49] sets a steady probability for a packet to have the same destination port as the source (defaulting to 0.5, i.e. $p_{i,j} = r/2$ for $j = i$), and equal for the remaining output ports (i.e. $p_{i,j} = r/(2(P_O - 1))$ for $j \neq i$, $P_I = P_O$).

The hotspot model is also representative of a variety of today’s systems including SoCs where a memory node can be a communication hotspot in certain configurations [10]. It can also be seen as a hybrid traffic model, as it combines uniform traffic with a (longer term) bursty traffic, which is also reminiscent of today’s multi-tasking and multi-tenant systems.

1. Source code available: <https://philippos.info/switches>

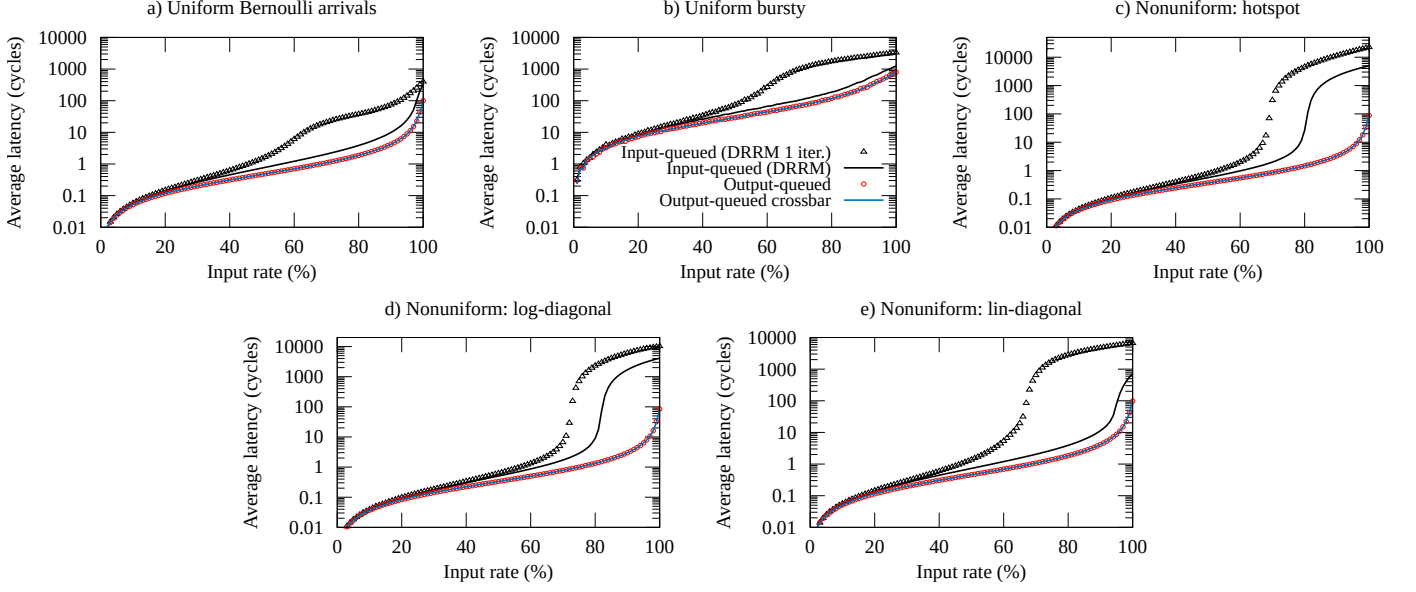


Fig. 6. Output-queued switches yield the lowest packet latencies

The final two nonuniform traffic patterns were introduced in the literature to add additional complexity to the hotspot model. These could be seen as less pronounced hotspot variations. Their intensity variation among each output port directly impacts the queue utilisation pattern, which is our prime target for normalisation through the proposed rotation approach.

- (d) *Nonuniform – log-diagonal*: This nonuniform traffic model [50], [51] scales the probabilities of consecutive output ports j by the same amount (0.5) when receiving from the same input port i . The resulting formula is $p_{i,j} = r \times 2^{P_O - 1 - ((i+j) \bmod P_O)} / (2^{P_O} - 1)$, where the i value is only used to rotate the resulting logarithmic probability matrix with respect to the sending input port [51].
- (e) *Nonuniform – lin-diagonal*: This is another nonuniform traffic model [50], [51], where the probability of sending from the same port to any two consecutive output ports differs by a fixed amount ($1/P_O$), i.e. $p_{i,j} = 2r \times (1 + ((i+j) \bmod P_O)) / (P_O(P_O + 1))$. Similarly with the log-diagonal model, the i here is only used as an offset to the probability matrix.

The selection of the competing approaches as well as traffic models is not completely consistent throughout this section, as the experiments have different motives. For example, section 5.3 uses a subset of the aforementioned traffic models that are also common for NoC evaluation.

For simulations, two variations of the input-queued switch (with virtual output queues (VOQs)) are also included. First, the input-queued “DRRM (1-iter)” switch represents one with a single iteration of the dual round-robin matching scheduling algorithm (DRRM [44]). As with other related algorithms [25], a single iteration yields generally suboptimal scheduling performance, but it is included for the interests of FPGA implementation efficiency. In other words, the 1-iteration version is included to study an input-queued switch with minimal scheduling complexity benefiting implementation (best case for the critical path) rather than scheduling behaviour.

Then, a $\log_2(P_I)$ -iteration version of “DRRM” is included, representing a VOQ-based switch with typical scheduling performance [13]. Multiple iterations are commonly used to approach maximal matching [43], hence the lower performance variation between other scheduling algorithms when $\log_2(P_I)$ iterations are used [12]).

For FPGA implementation (section 5.4), only the 1-iteration version of DRRM is used out of the two, since we target switches that can produce output-per-cycle (yielding full-throughput). Input-queued switches with multiple iterations are not considered for single-cycle implementation, due to the critical path for scheduling (see sections 3 and 6).

5.1 Scheduling Performance (Simulation): Latency

First, we would like to measure the algorithmic performance of the approaches independently of the implementation details, packet size and other system effects. Thus, any processing latency is normalised to 0, and only the time a packet stays in the queues is considered.

The evaluation framework is partly based on the open-source repository of Hipernetch [12]. Additional switches and traffic patterns are developed in high-level using python, including the output-queued with the rotator (proposal) to facilitate this study. Each data point represents an average of multiple runs, each with 25,000 cycles of injection time (and 5,000 cycles warmup). The number of ports is set to 16, but without loss of generality in the conclusions.

The first experiment studies the average packet latency, i.e. the average time a packet stays in the queues in cycles. For this experiment, the queues are conventionally considered of infinite size to focus on the scheduling performance irrespective of the queuing effects.

As observed from figure 6, the output-queued switch and the output-queued crossbar perform almost identically, and achieve the lowest average latency (see challenge C2). For uniform traffic, the input-queued DRRM switch achieves to be a relatively close second, but not for nonuniform traffic. DRRM’s single-iteration version is significantly

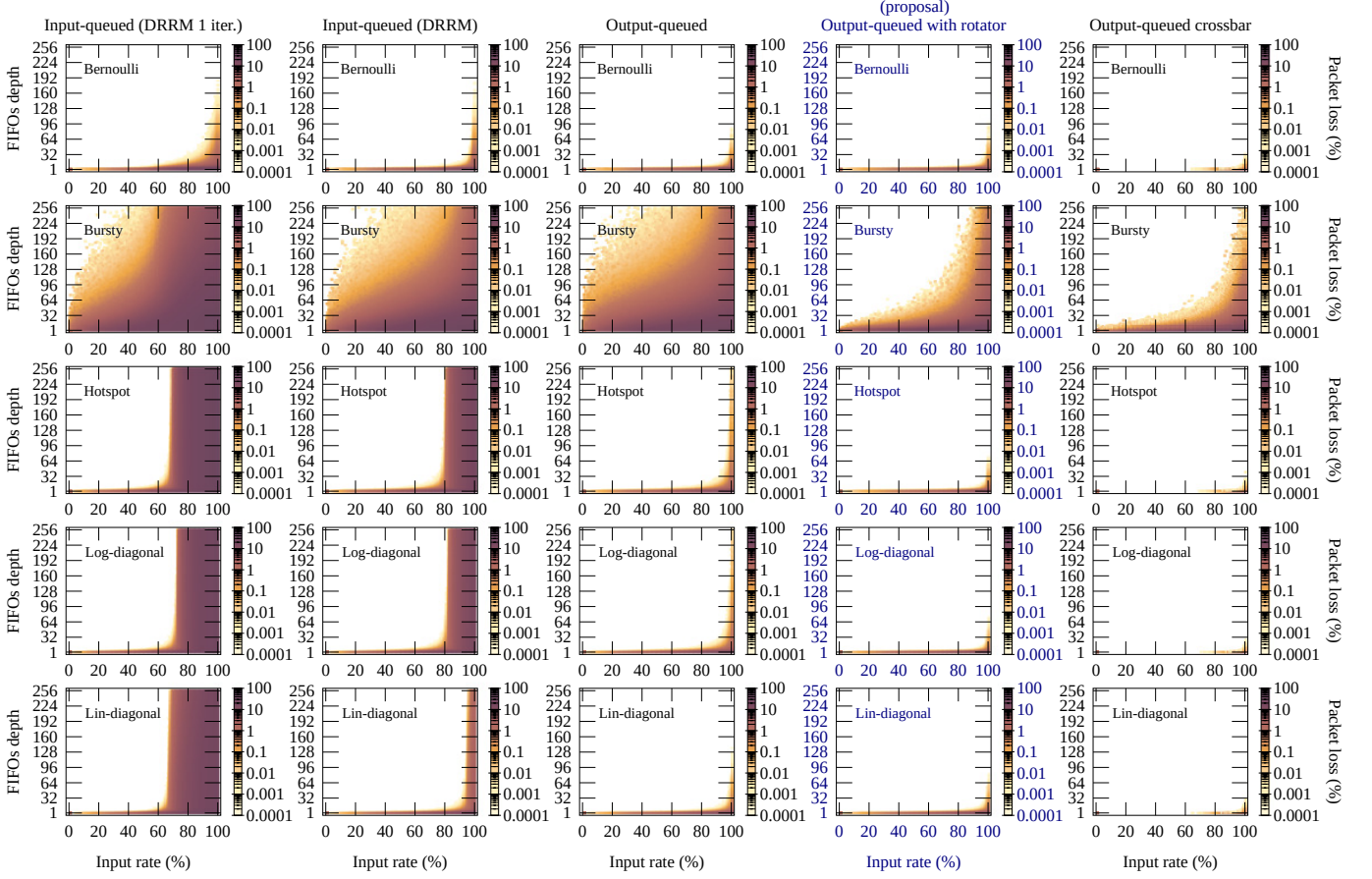


Fig. 7. Packet loss while varying the input rate and FIFO depth, for different traffic patterns. The rotator is close to the optimal (rightmost).

worse, starting from an input rate of about 40% and beyond for all traffic patterns (see challenge C5). As a numerical example for 100% input rate under uniform bursty traffic, the output-queued switches yield an average latency of 772 cycles, which is 1.6 times lower than the 1251 value of DRRM, which is in turn 2.66 times lower than 3325 from DRRM with 1 iteration. The respective numbers for the hotspot nonuniform traffic are 89, 4989 (56.1x higher) and 22759 (4.6x higher) cycles. The lin-diagonal nonuniform traffic differs more to the other models with respect to the discrepancy of the 1-iteration DRRM, updating the numerical comparison to 98, 723 (7.4x higher) and 6711 (9.3x higher) cycles for output-queued, DRRM and 1-iteration DRRM respectively.

Since this assumes infinite queues, the proposed queue-balancing switch is omitted here, as its latency performance is identical to the other output-queued switches. Studying the latency with infinite queues can also be indicative for the task completion time when the switch is blocking instead of dropping packets, such as for memory requests. Though, detailed simulations can be more appropriate for measuring performance in such systems (see section 5.3).

5.2 Scheduling Performance (Simulation): Packet Loss

There are applications where packet loss is allowed, such as in network switches. For instance, the Transmission Control Protocol (TCP [47]), which is the prominent protocol in

today's internet, assumes and mitigates packet losses, such as through acknowledgements and retransmissions. In such cases, minimising the packet loss rate is crucial for service and application performance.

On FPGAs it is important to study the impact of packet loss, as it relates directly to the queue size and organisation. For example, the main argument for input-queued switches (with VOQs) is the ability to share memories, reducing the queue complexity from $P_I \times P_O$ (quadratic for $P_I = P_O$) to P_I [4]. However, according to the theoretical switching performance, we show that it can still be worth using $P_I \times P_O$ queues, but of a fraction of the size, rather than P_I longer queues through memory sharing.

5.2.1 Input rate

The common parameter in the studied traffic models is the input rate. The impact of the queue size (depth) on the packet loss rate is presented in figure 7 for five different switching approaches under the studied traffic patterns, while varying the input rate. From left to right, the switches are sorted according to their ascending overall performance, starting from a DRRM-based switch with 1 iteration. The first four use the same queue arrangement as virtual output queues (VOQs), which is one queue per input-output combination (ignoring memory-sharing complications). One exception is with the last (output-queued crossbar), which is emulated by Hipernetch [13] that features the same queue organisation for consistency.

The main observation is the rotator effect on the output-queued switch, which considerably improves the queue utilisation efficiency. The exception to this is the uniform Bernoulli arrivals (first row of figure 7), as the packets are already evenly distributed across the input ports, which is what the rotator tries to emulate to improve the queue utilisation efficiency. The output-queued switch with the rotator yields a similar packet loss profile to the output-queued crossbar (equivalent [13]), which is considered optimal.

Under uniform bursty traffic, the packet loss profile of the output-queued switch (without rotator) is very similar to the DRRM switch, being in line with the expectations set in section 5.1. By including the rotator, under bursty traffic with 80% input rate and a FIFO depth of 32, the output-queued switch improves its packet loss from 11.7% to 1.5% (7.9x reduction in packet loss). An 11.7% packet loss can also be achieved with an output-queued switch with the rotator with a FIFO depth of only 7 (4.6x reduction in queue space).

For nonuniform traffic, the input-queued approaches have a more significant hit on performance than for uniform traffic, and the rotation approach further improves on the output-queued switch. This is because nonuniform traffic directly impacts the queue usage unevenness in the long term. For hotspot traffic at 100% input rate and FIFO depth equal to 4, the packet loss reduces from 4.3 to 1.6% (2.6x reduction), and the 4.2% is also improved when using half the FIFO size for the rotator at 3.6%. The respective comparison for the other nonuniform traffic models is from 4.6 to 1.5% (3.1x reduction) for log-diagonal and from 2.7 to 2.1% (1.2x reduction) for lin-diagonal traffic.

If we combine the inefficiencies of the DRRM switch with just 1 iteration for a comparison with the proposed switch (meaningful when it comes to implementation, as in section 5.4), the difference is more dramatic. Under bursty traffic with input rate 80%, a FIFO depth of 32 yields 26.2% packet loss for DRRM with 1 iteration (17.6x more than the rotator with the same FIFOs), and it is close to an output-queued switch with rotator with only a single register per FIFO (32x space reduction) at 26.3% packet loss. For nonuniform traffic, for an input traffic of 100% and a FIFO depth of 4, DRRM with one iteration drops 36.1% of the packets, which is more than the 8.5% of the rotator approach with a register per queue. Lowering the memory requirements can be a key to avoiding BRAM-based implementation using VOQs, which can be a limiting factor for scalability [23].

As a conclusion, the proposal performs near-optimally with respect to latency and queue utilisation for the studied traffic models, and an efficient FPGA implementation would further justify its appropriateness as an FPGA solution.

5.2.2 Traffic model-specific attributes

It is also relevant to understand the memory needs of a switch with respect to the traffic features other than the input rate. While most studied traffic models are more strictly-defined, some can be further customised.

Figure 8 presents a similar experiment only for bursty traffic and the output-queued switch (left), and with a rotator (right). The x-axis now represents the burst size instead of the input rate, which is now fixed to 80%. The burst size is demonstrated to be an important factor in deciding the memory organisation of the switch.

The rotator approach is also shown to be consistent in improving the packet loss with varying the burst size. In other words, the main difference between the two plots of figure 8 is that the y-axis (FIFO depth) is mostly stretched by a constant. This is because a single burst that would originally go into a single FIFO is now able to be distributed across multiple queues. For a numerical example, under bursts of size 128, a packet loss of around 14.5% can be achieved with either an output-queued switch with a FIFO depth of 100 or the proposed switch with a FIFO depth of 20. This is a 5x reduction, which is similar to the reduction reported by section 5.2.1 for a burst size of 32.

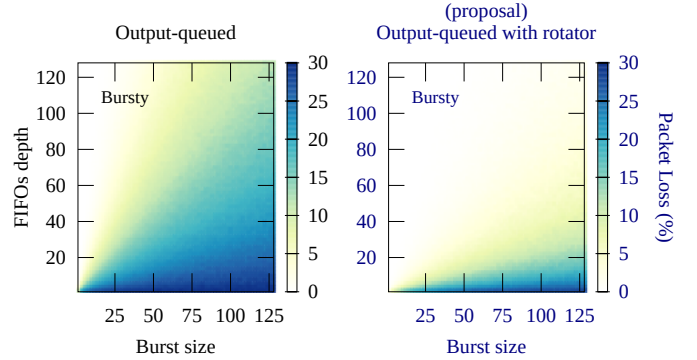


Fig. 8. Bursty traffic: impact of burst size on memory requirements

A similar experiment is also done for the hotspot nonuniform traffic, but instead of varying the burst size, the packet loss is measured while varying the probability to send to the same port. In this experiment the input rate is kept at 100%. This is illustrated in figure 9, where the proposal yields lower packet loss rates overall, though the relationship between them is less trivial than before (non-linear). For example, under hotspot traffic with 20% probability, the output-queued switch requires a queue length of about 150 to achieve near zero packet loss, and with 80% this increases to 254, while the proposal requires a FIFO depth of 66 and 38 respectively for similar packet loss behaviour.

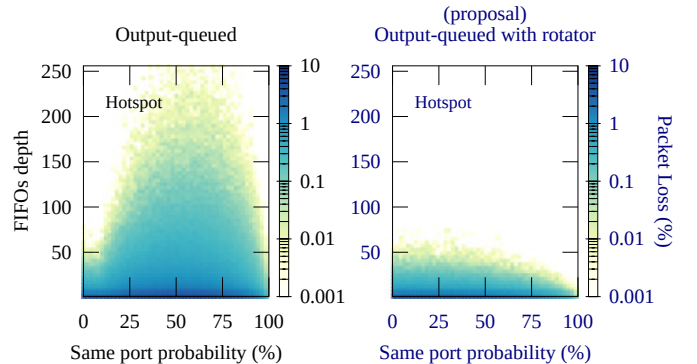


Fig. 9. Hotspot traffic: impact of the rate of sending to the same port

As can be observed by comparing the packet loss under bursty traffic and hotspot, as well as from the rest of the experiments in this section, bursty traffic can be the most demanding. This insight ignores the input-queued switch results (DRRM variations) where the crossbar limitations have a greater hit on performance for nonuniform traffic. For this comparison, the points of interest from the bursty

traffic are one to two orders of magnitude higher than those of the hotspot (also note the colour axis scale difference). This highlights the importance of optimising for bursty traffic, which is also representative for various systems with more advanced communication protocols (but moving from packet loss to additional latency).

5.3 NoC Simulation

The proposed solution can be adopted inside network-on-chips (NoCs) to implement the router of each node. This experiment evaluates its impact on NoCs and is based on an in-house NoC simulator modelling an 8×8 NoC totalling 64 nodes. Each node contains a placeholder for a 5×5 switch. The 5 ports per router switch represent each direction (north, east, south and west) and the node itself, as illustrated in figure 10. The selected traffic models are (a) uniform Bernoulli arrivals and (b) uniform bursty traffic.

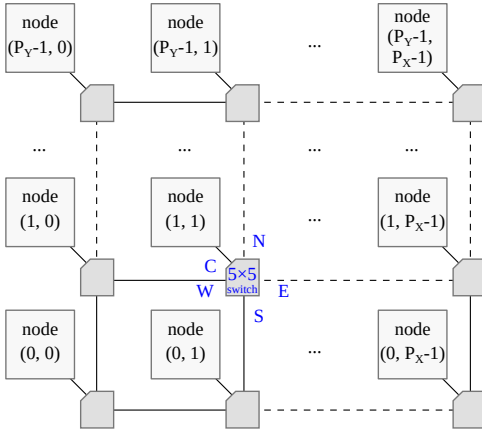


Fig. 10. 5-port (5×5) switches in a NoC

In contrast to the isolated switch simulations in the other parts of the evaluation, no packet loss is allowed, and at the same time the queues have an indicative fixed depth. This means that backpressure is enabled, which also introduces the risk for deadlock according to the routing algorithm.

The proposal conserves the deadlock-free property of the input-queued (wormhole) and output-queued switches as NoC routers that use existing routing algorithms for traversal through the nodes. Deadlock-free routing algorithms verified to work with all studied switches include dimension-ordered routing (XY for travelling first through the X axis, and YX for travelling first through the Y-axis) and other turn restriction-based routing [52]. This is because, in the worst-case the rotation serialises the packet options with respect to each output port of the router. This case is equivalent to having at least one buffer per output (or input) port, which can be satisfied by the turn model [52] (supporting more advanced models is part of future work).

Figure 11 introduces the results of this experiment. The general observation is that the expected performance ranking of the studied switches is inherited in NoCs built around them. However, there are some non-trivial cases. First, in contrast to the isolated latency experiments of section 5.1, here each of the three output-queued switches performs differently, including under Bernoulli arrivals. While the traffic models are equivalent to those in the isolated evaluation,

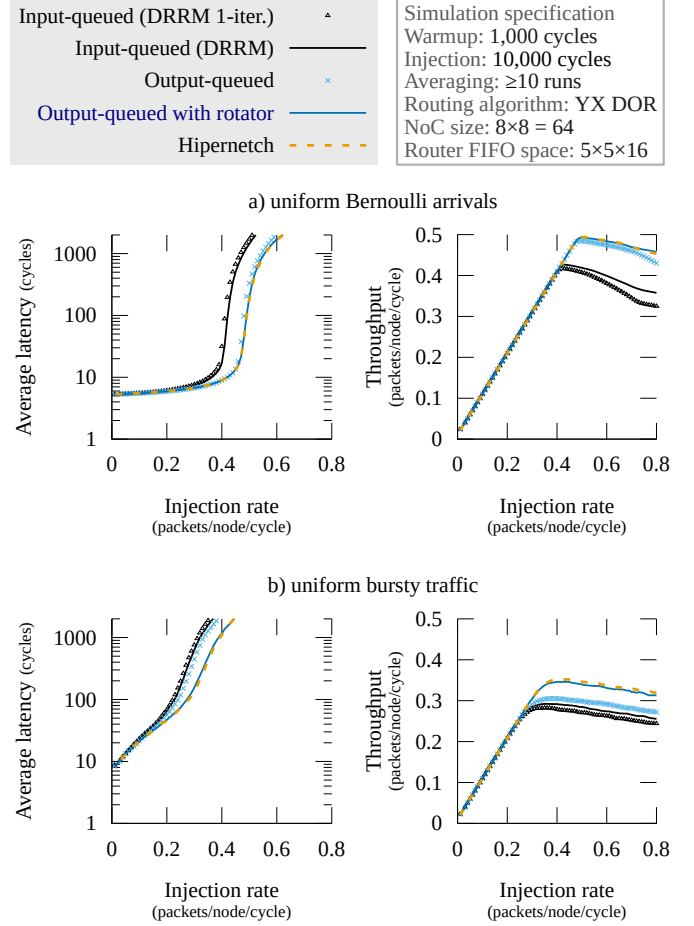


Fig. 11. Comparing switches as NoC routers

they concern the entire system, and the routers themselves do not necessarily experience the same distributions. For instance, the leftmost nodes never send packets to the west.

Similarly, the average latency metric is not representative of performance when stand-alone (hence also the use of infinite queues in section 5.1). Another performance metric that is used to study NoC performance is the throughput. The idea is that the system can “keep up” with the injection rate as long as it is equal to the throughput (for this setup), both of which are measured as the average number of packets being injected (or extracted for throughput) to each node per cycle. An example for why both metrics are useful is for an input rate of 40% of traffic a), where all switches yield a throughput of 100% the injection rate, even though the average latency already differs (8.9, 16.9 and 31.5 cycles for output-queued, DRRM and DRRM 1-iter. respectively). Additionally, the throughput can be used to easily identify inefficiencies at higher injection rates. For example, for 0.8 injection rate of a), the removal of the rotator from the output-queued lowers the throughput by 6.1%. This can be explained by the inefficient use of the buffer space in the VOQ-like arrangement of the output-queued switch, which can enforce backpressure more frequently.

Finally, the highest jump in performance for Bernoulli arrivals comes from the move to output-queued switches. Under bursty traffic however, the rotation effect of both the proposal and Hipernetch is more dramatic. The rotator-

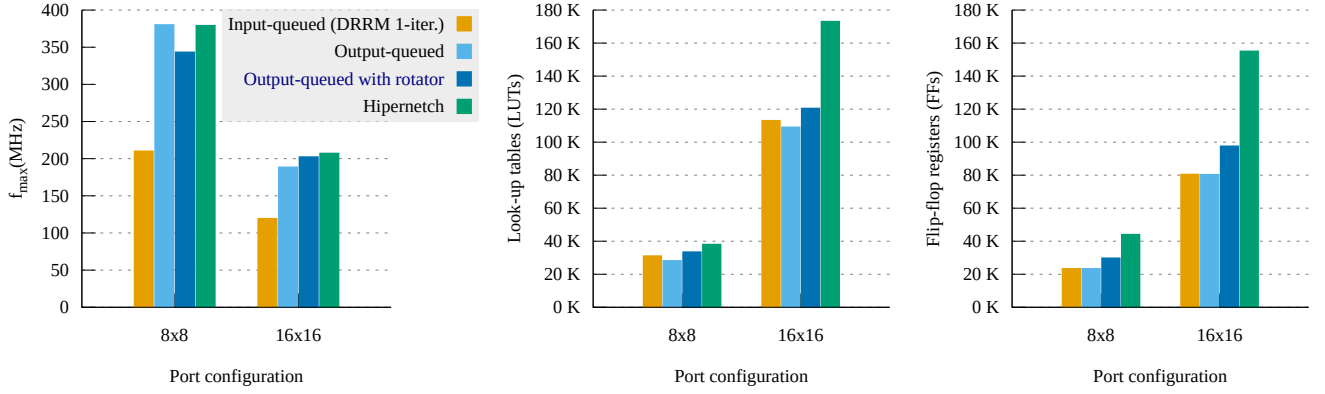


Fig. 12. Switch resource utilisation and operating frequency as reported by Vivado 2020.1 for Xilinx Alveo U280

based switch is shown to improve the throughput by 41, 28 and 6.5% over the 1 and 3-iteration versions of input-queued, and the output-queued for an input rate of 80% for Bernoulli arrivals. The numerical example above is updated to 28, 23 and 15% under bursty traffic with a 80% load. For Hipernetch the throughput values for this example were similar to the proposal ($\pm 1\%$).

5.4 FPGA-based Implementation

In order to assess the implementation efficiency of the compared switches as FPGA designs, three switches are implemented from scratch: (1) an input-queued switch with DRRM (1 iteration), (2) an output-queued and (3) an output-queued switch with the rotator optimisation. These designs are also compared with an existing switch with a similar setup (4) Hipernetch [12], with the highest theoretical performance. The switches are implemented for the Xilinx Alveo U280 board using Vivado 2020.1. These implementations work as peripherals on a real FPGA for validation purposes, but are out-of-context with respect to any I/O devices, system memories etc.

Switches (1), (2) and (3) feature FIFOs with depth equal to 8. The switch (1) even though is based on virtual output queues, it does not exploit memory sharing, as we test LUTRAM-based memory. Otherwise, the added logic complexity would shift our design space towards further optimising the critical path of memory sharing logic. The open source Hipernetch (4) implementation [13] only has a register per FIFO by default, being more a proof-of-concept.

5.4.1 Basic Implementation Comparison

Initially, Vivado-reported implementation metrics for utilisation are overviewed for the competing switches. As can be observed from the figure 12 left, the maximal operating frequency (f_{max}) is similar between switch implementations of the same port configuration, except with the input-queued switch which has a significant overhead in both 8x8 and 16x16 configurations. With respect to the FPGA resource utilisation, the look-up table (LUT, figure 12 middle) and flip-flop register (FF, figure 12 right) utilisation varies less between different switch sizes, and the difference is more consistent across different approaches. See section 6 for additional reasoning behind those observations.

The overhead of adding the rotator to the output-queued switch seems rather negligible, as the f_{max} drops from 380 to 343 MHz for 8x8, but increases from 188.5 to 202 MHz for 16x16. Such small variations are expected from heuristic-based place-and-route tools, but the increase in frequency can also be explained by the addition of pipeline stages from the rotator. In terms of LUTs and FFs, there is generally a 10 to 19% increase when adding the (pipelined) rotator. Though, it is still considerably more resource efficient than Hipernetch. It uses 59% more FFs and 44% more LUTs than the rotator approach for 16 ports, also noting it only has a register per FIFO in its original configuration.

The comparison in figure 12 used one data point per switch implementation for brevity. Specifically, switches (3) and (4) use a pipelined implementation, and there is a design space concerning where the pipeline registers are placed. (3) uses a fully-pipelined barrel shifter implementation (a register per pipeline stage), resulting in additional $\log_2(P_1)$ cycles. For (4), Hipernetch has an optimisation parameter (S) that relates to the number of registers in the pipeline [12]. We selected a well-performing [12] configuration of $S = 3$ and $S = 4$ for the 8x8 and 16x16 configurations respectively that provide a latency of 4 FPGA cycles in both cases. Following is a design space exploration of the register placement inside their corresponding pipelines, and is analogous to retiming, which can be done automatically by the toolchain [53].

5.4.2 Latency and Throughput Design Space

The placement of registers impacts more than one performance attribute of the resulting implementations. Thus, a more detailed design space exploration is presented for focusing on related performance metrics. These metrics are the operating frequency, port bandwidth (proportional to the frequency), and the port-to-port latency. Note that these metrics do not cover the algorithmic performance that was explored separately in simulation (in sections 5.1 to 5.3).

Figure 13 summarises how the obtained operating frequency translates into port-to-port latency in nanoseconds (ns) and port bandwidth in billion bits per second (Gbps), for both 8 and 16-port switches. Since all studied approaches can fully saturate the line rate, the frequency is multiplied by the studied packet size (256-bit) to provide the port throughput. At around 800 Gbps for both 8 and 16 ports,

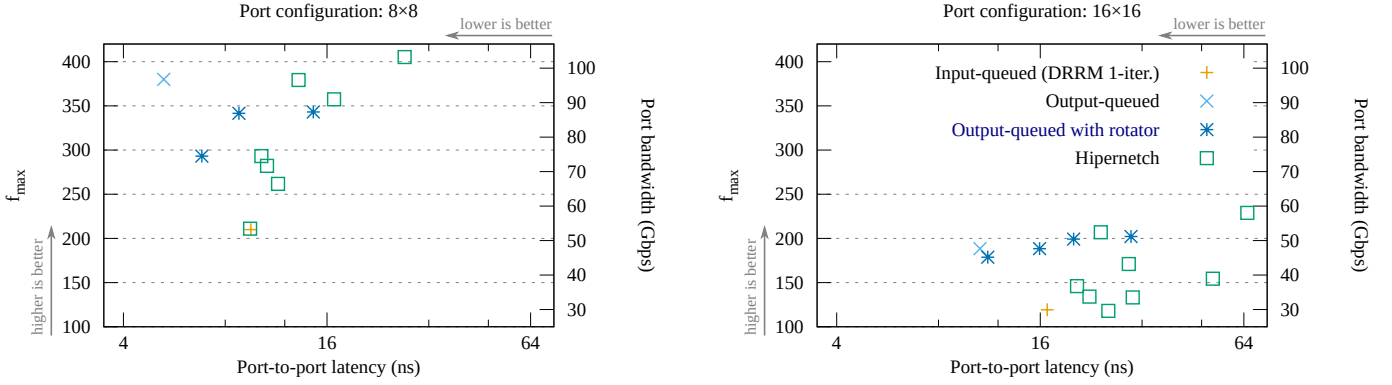


Fig. 13. Port bandwidth and port-to-port latency

the achievable throughput is similar between all switches excluding the input-queued. The jump to 16 ports approximately halves the operating frequency, hence the smaller variation in the aggregate throughput. This means that hierarchical approaches can still be relevant if a higher throughput is required [4], and our solution could also be applied as a building block.

Regarding the port-to-port latency (x-axes of figure 13), this is obtained by multiplying the clock period to the latency of each switch in FPGA cycles. The pipeline latency of (1) and (2) is 1 cycle (plus one more for enqueueing a single packet). Out of the two, (2) has the lowest resulting port-to-port latency due to its high operating frequency, which may be also useful in certain applications where the single-cycle latency might be desirable, such as for easier backpressure support (without excessive buffering needs), or in systems where exhaustive scheduling is a requirement, including for bursts in some systems interconnects.

The leftmost data point of (3) has a similar port-to-port latency to the output queued switch (in both the 8×8 and 16×16 configurations), because it represents an output-queued switch with a rotator of no additional pipeline registers, resulting in the same pipeline latency. A related observation is also that the design space of (3) mostly varies the port-to-port latency (x-axis) rather than the operating frequency (y-axis), when compared to (4). This highlights that the rotator has a relatively low overhead and that the most demanding logic relates to the multiplexers of the output-queued switch as a building block of (3).

When comparing (3) to (4), the proposal (3) achieves lowest port-to-port latency with a similar algorithmic performance (as shown in section 5.1). (4) can achieve the highest throughput, but for sometimes prohibitively high resource utilisation (as demonstrated in section 5.4.1 for one of the well-performing points, and with a fraction of the queue space).

All switches here provide relatively low port-to-port latencies, including our proposed queue-balancing switch (3), especially when compared with hierarchical [4] and iterative [23] alternatives which have already been shown to be an order of magnitude higher than (4) Hipernetch [13].

6 COMPLEXITIES

This section elaborates on the theoretical complexities of the studied architectures, in order to understand how our

approach compares to the alternatives, as well as to better interpret the experimental results from the FPGA-based evaluation (section 5.4).

With respect to the critical path, the scheduling complexity is shown to affect the operating frequency and subsequently the throughput. This is especially apparent when moving from the input-queued switch with virtual output queues (VOQs) to the output-queued switch. If we abstract the scheduling algorithm required by the input-queued switch as a combinational circuit, it has a high number of inputs (fanin). This is because the input-queued switch scheduler requires the valid bit of every queue (totalling $P_I \times P_O$), the current rotation of the priority encoders to enforce fairness such as with round-robin priority [44] ($P_I \times \log_2(P_O)$) and a bit per output port (P_O bits, in case port availability is required). This totals to $P_I \times (P_O + \log_2(P_O)) + P_O$, which sums to 336 inputs for $P_I = P_O = 16$.

On the other hand, output-queued switches have simpler scheduling, because each output arbiter is independent. A simple arbitration scheme that was used in the evaluation is to have a round-robin arbiter with a priority encoder per output port, which totals to a maximum fanin of $P_I + \log_2(P_I) + 1$ bits (the +1 is for when port availability is concerned). In the case of the rotator, the valid bits that need to be read by the port arbiter are stored in the form of small packets. Note that the priority is still round-robin, but on every arbiter offset wrap-around, the packet is dequeued so that each valid bit is read once for every packet. Thus, the output-queued switch with a rotator needs one more bit to check for valid packet availability, totalling $P_I + \log_2(P_I) + 2$ bits. This sums to 22 bits for $P_I = P_O = 16$.

Using an input switch with a multi-iteration and multi-cycle scheduling algorithm would normally isolate the workload in circuits with smaller fanin. Scheduling algorithms for input-queued switches approximate maximal matching without needing access to large truth tables through long LUT chains. Sometimes, each iteration consists of multiple phases (such as request, grant and accept in iSLIP [25]), and each can also occupy a different cycle. In such cases the maximum fanin reduces to that of the arbiters [45] typically totalling $\max(P_O + \log_2(P_O), P_I + \log_2(P_I)) + 2$ for round-robin in the arbiters near VOQs and near the outputs. Though, the suboptimal scheduling capabilities and FPGA-related complications of VOQs would still apply.

TABLE 1
Comparing switch architectures

Switch	Total queue space (bits)	Queue fragmentation	Fastest growing component	Highest scheduling fanin	Scheduling performance
Input-queued, memory-sharing [4], [23]	$P_I \times \text{depth}_{FIFO} \times \text{width}_{packet}$	Yes	$P_I \times P_O$ memory logic units	$P_I \times (P_O + \log_2(P_O)) + P_O$	Low
Input-queued, no memory-sharing	$P_I \times P_O \times \text{depth}_{FIFO} \times \text{width}_{packet}$	Yes	$P_I \times P_O$ packet queues	$P_I \times (P_O + \log_2(P_O)) + P_O$	Low
Output-queued	$P_I \times P_O \times \text{depth}_{FIFO} \times \text{width}_{packet}$	Yes	$P_I \times P_O$ packet queues	$P_I + \log_2(P_I) + 1$	High
Output-queued, with rotator	$P_I \times P_O \times \text{depth}_{FIFO} \times (\text{width}_{packet} + P_I)$	No	$P_I \times P_O$ packet queues	$P_I + \log_2(P_I) + 2$	High
Hipernetch [12], [13]	$P_I \times P_O \times \text{depth}_{FIFO} \times \text{width}_{packet}$	No	$P_O \times P_I \times \log_2(P_I)/2$ rotator switches	$P_I + \log_2(P_I) + 1$	High

The evaluation studied single-cycle implementations, including the single-iteration version of an input-queued switch. On FPGAs, the most significant overhead of iterative (and multi-phase) approaches could be considered the impact on throughput, due to the operating frequency limitations [13]. As the crossbar is reconfigured less frequently than the main design, there will be diminishing returns relating to the packet size to achieve the bandwidth potentially required by other entities on the system. For special use cases, iterative and multi-phase approaches could still be meaningful, if performance is not a priority, or for future FPGA architectures enabling faster operation.

Table 1 summarises the findings of the paper on the studied switch architectures, as well as the complexities mentioned in this discussion and related work. The “scheduling performance” reflects the results of section 5.1, and denotes whether the switch approaches the latency of the output-queued crossbar. The “queue fragmentation” attribute is based on the queue balancing ability for nonuniform traffic and is demonstrated experimentally in the results of sections 5.2 and 5.3.

The first entry of table 1 is a memory-sharing version of the input-queued switch and is provided here only for completeness. The memory-sharing aspect can have different attributes, including the need for $P_I \times P_O$ output pointer queues [4], [23], and differs according to the implementation. The added virtual queue-handling logic is abstracted inside the notion of “memory logic units” that is its fastest growing hardware component (ignoring the potential scheduler-related logic complexity, which is seen as variable and more purely combinatorial).

7 DISCUSSION AND FUTURE WORK

This section discusses the generality of the conclusions, as well as potential future work to widen their applicability. The evaluation in both simulation and implementation is consistent with the more-theoretical complexity analysis and the intuitions behind the proposal. The proposed switch combines the low latency of the output-queued switch with the queue balancing approach for a near optimal queue utilisation. The rotator benefits all uneven or bursty traffic. Implementation-wise, the input-queued is severely bottlenecked by the scheduling algorithm, even though a

simplistic one is used as a baseline. The resource utilisation remains similar for all other switches other than Hipernetch, which follows its “fastest growing component” complexity.

While the observations in simulation sections 5.1 to 5.3 hold without loss of generality, the exact numerical results are case specific, especially for demanding traffic. This is because when a switch is not able to cope with the incoming traffic then more and more packets are being accumulated in its queues, hence the latency or queue requirements being in the order of the magnitude of the simulated packet arrivals for high input rates.

This survey does not cover additional functionality and optimisation that might be desirable in certain scenarios. For example, for this reason as well as due to the reasoning in the discussion on FPGA use of section 6, only single-flit packets are considered. In the case of multi-flit packets, the proposed approach can be modified to update the cycle counter (see figure 4) only when all tails of in-transfer packets are received. Such arrangement can ease the reconstruction of packets and routing algorithms in more hierarchical approaches, which can be explored further.

Although sometimes it is trivial to generalise the studied techniques more, there are some aspects that are less orthogonal. For instance, quality of service (QoS) support and memory sharing, although less common on FPGA switches, they both involve more complex memory usage schemes, including virtual channels. Thus, future work could include a comparison of the rotator approach on such switches, and their interaction with the different memory technologies available on FPGAs. Additionally, the implementation efficiency for specific targets can be studied extensively, such as with regard to backpressure support, or alternative circuits for evenly distributing the packets into the queues. The latter could include random permutations from a butterfly network, or more deterministic approaches [12], [54].

For NoC use, the rotator approach can enforce limitations on the routing algorithm of the NoC according to the use case. Many NoCs require the NoC routing algorithm to have a consistent route for packets between each two nodes [10]. This is to ensure that flits of the same packet follow the same path, otherwise the rotation of the inputs inside the routers could rearrange their order. This requirement is already satisfied for simpler common NoC routing algorithms such as XY and YX DOR [55], though the wider applicability

of the rotator can be studied further.

8 CONCLUSIONS

In this paper, a novel switch architecture is proposed that approaches the algorithmic and FPGA-based performance of the state-of-the-art, but with a considerable reduction in resource utilisation. It is also demonstrated that the input-queued switches are inappropriate in high-throughput FPGA-based applications, due to the costly scheduling algorithms and the low theoretical switching performance. One challenge in common queueing schemes, such as with virtual output queues, is the queue fragmentation that can reduce the utilisation by an order of magnitude for demanding traffic. Our proposed rotator-based switch solves the fragmentation issue for the output-queued switch, while having a small hardware overhead. Our study targets monolithic high-performance designs with a low to medium radix in mind, and for FPGA use. They can be applied in existing hierarchical designs such as those for implementing larger interconnects, routers for network-on-chips (NoCs) and other switches aiming at scalability.

ACKNOWLEDGEMENT

This research was sponsored by dunnhumby. The partial support of EPSRC (grant numbers EP/L016796/1, EP/L00058X/1 and EP/N031768/1) and the Japan Society for the Promotion of Science (JSPS) KAKENHI (grant numbers 20H00593 and 21H04869) is gratefully acknowledged.

REFERENCES

- [1] A. Interconnect, “v2. 1 logicore ip product guide,” PG059, Xilinx, December, vol. 20, 2017.
- [2] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. López-Buedo, “Limago: An fpga-based open-source 100 gbe tcp/ip stack,” in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 286–292.
- [3] M. Naylor, S. W. Moore, D. Thomas, J. R. Beaumont, S. Fleming, M. Vousden, A. T. Markettos, T. Bytheway, and A. Brown, “General hardware multicasting for fine-grained message-passing architectures,” in *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 126–133.
- [4] Z. Dai and J. Zhu, “Saturating the transceiver bandwidth: Switch fabric design on fpgas,” in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 67–76.
- [5] F. Abel, J. Weerasinghe, C. Hagleitner, B. Weiss, and S. Paredes, “An fpga platform for hyperscalers,” in *IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, 2017, pp. 29–32.
- [6] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, “Hbm connect: High-performance hls interconnect for fpga hbm,” in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 116–126.
- [7] T. Miyajima and K. Sano, “A memory bandwidth improvement with memory space partitioning for single-precision floating-point fft on stratix 10 fpga,” in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 787–790.
- [8] B. Adhi, C. Cortes, Y. Tan, T. Kojima, A. Podobas, and K. Sano, “The cost of flexibility: Embedded versus discrete routers in cgras for hpc,” in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 2022, pp. 347–356.
- [9] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, “P4xos: Consensus as a network service,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1726–1738, 2020.
- [10] T. Van Chu and K. Kise, “Lef: An effective routing algorithm for two-dimensional meshes,” *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 10, pp. 1925–1941, 2019.
- [11] N. Kapre and J. Gray, “Hoplite: Building austere overlay nocs for fpgas,” in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–8.
- [12] P. Papaphilippou, J. Meng, and W. Luk, “High-Performance FPGA Network Switch Architecture,” in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 76–85.
- [13] P. Papaphilippou, J. Meng, N. Gebara, and W. Luk, “Hipernetch: High-performance fpga network switch,” *ACM Transactions on Reconfigurable Technology and Systems*, 2021.
- [14] P. Papaphilippou, K. Sano, B. A. Adhi, and W. Luk, “Efficient queue-balancing switch for fpgas,” in *2021 International Conference on Field-Programmable Technology (ICFPT)*, Dec 2021, pp. 1–5.
- [15] S. Shreejith, P. Mundhenk, A. Ettner, S. A. Fahmy, S. Steinhorst, M. Lukasiewicz, and S. Chakraborty, “Vega: A high performance vehicular ethernet gateway on hybrid fpga,” *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1790–1803, 2017.
- [16] R. Mueller, J. Teubner, and G. Alonso, “Sorting networks on fpgas,” *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 1–23, 2012.
- [17] B. P. Prasad, K. Parane, and B. Talawar, “An efficient fpga-based network-on-chip simulation framework utilizing the hard blocks,” *Circuits, Systems, and Signal Processing*, vol. 39, no. 10, pp. 5247–5271, 2020.
- [18] A. Bitar, J. Cassidy, N. Enright Jerger, and V. Betz, “Efficient and programmable ethernet switching with a noc-enhanced fpga,” in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 89–100.
- [19] N. Rameshan, A. Biyani, M. Gaur, V. Laxmi, and M. Ahmed, “Qos aware minimally adaptive xy routing for noc,” in *17th International Conference on Advanced Computing and Communication (ADCOM)*, Bangalore, India, 2009.
- [20] J. C. Borromeo, I. Cerutti, P. Castoldi, R. Reyes, and N. Andriolli, “Fpga-based implementation of two-step schedulers for modular optical interconnection networks,” *Journal of Optical Communications and Networking*, vol. 13, no. 5, pp. 116–125, 2021.
- [21] Z. Guang, Y. Lin, Z. Ming, and M. Yilan, “The improvement and implementation of islip algorithm based on fpga,” in *International Conference on Trustworthy Computing and Services*. Springer, 2014, pp. 260–266.
- [22] I. Cerutti, J. A. Corvera, S. M. Dumlaio, R. Reyes, P. Castoldi, and N. Andriolli, “Simulation and fpga-based implementation of iterative parallel schedulers for optical interconnection networks,” *Journal of Optical Communications and Networking*, vol. 9, no. 4, pp. C76–C87, 2017.
- [23] J. Meng, N. Gebara, H.-C. Ng, P. Costa, and W. Luk, “Investigating the feasibility of fpga-based network switches,” in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2019.
- [24] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, “High-speed switch scheduling for local-area networks,” *ACM Transactions on Computer Systems (TOCS)*, vol. 11, no. 4, pp. 319–352, 1993.
- [25] N. McKeown, “The islip scheduling algorithm for input-queued switches,” *IEEE/ACM transactions on networking*, no. 2, pp. 188–201, 1999.
- [26] J. Chao, “Saturn: a terabit packet switch using dual round robin,” *IEEE Communications Magazine*, vol. 38, no. 12, pp. 78–84, 2000.
- [27] Y. Li, S. Panwar, and H. J. Chao, “The dual round robin matching switch with exhaustive service,” in *Workshop on High Performance Switching and Routing, Merging Optical and IP Technologies*. IEEE, 2002, pp. 58–63.
- [28] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, “Matching output queueing with a combined input/output-queued switch,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.
- [29] R. B. Magill, C. E. Rohrs, and R. L. Stevenson, “Output-queued switch emulation by fabrics with limited memory,” *IEEE J. on Selected Areas in Communications*, vol. 21, no. 4, pp. 606–615, 2003.
- [30] Xilinx, “Block memory generator, v8. 4. logicore ip product guide,” 2017.

- [31] Y. Huan and A. DeHon, "Fpga optimized packet-switched noc using split and merge primitives," in *2012 International Conference on Field-Programmable Technology*. IEEE, 2012, pp. 47–52.
- [32] N. Kapre, N. Mehta, M. Delorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, "Packet switched vs. time multiplexed fpga overlay networks," in *14th Sym. on Field-Programmable Custom Computing Machines*. IEEE, 2006, pp. 205–216.
- [33] Z. Ge, D. R. Figueiredo, S. Jaiswal, and L. Gao, "Hierarchical structure of the logical internet graph," in *Scalability and Traffic Control in IP Networks*, vol. 4526. SPIE, 2001, pp. 208–222.
- [34] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, 1991.
- [35] M. B. Petersen, S. Nikolić, and M. Stojilović, "Netcracker: A peek into the routing architecture of xilinx 7-series fpgas," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 11–22.
- [36] C. Chiasson and V. Betz, "Should fpgas abandon the pass-gate?" in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–8.
- [37] H. Wong, V. Betz, and J. Rose, "Comparing fpga vs. custom cmos and the impact on processor microarchitecture," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 5–14.
- [38] W. Luk, V. Lok, and I. Page, "Hardware acceleration of divide-and-conquer paradigms: a case study," in *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 192–201.
- [39] K. T. Pham, T. T. Nguyen, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Scalable low-latency inter-fpga networks," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 234–245.
- [40] J. Barnes and P. Hut, "A hierarchical $O(n \log n)$ force-calculation algorithm," *nature*, vol. 324, no. 6096, pp. 446–449, 1986.
- [41] Y. Xue and P. Bogdan, "User cooperation network coding approach for noc performance improvement," in *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM, 2015.
- [42] —, "Improving noc performance under spatio-temporal variability by runtime reconfiguration: a general mathematical framework," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016, pp. 1–8.
- [43] F. J. Gonzalez-Castano, C. Lopez-Bravo, R. Asorey-Cacheda, P. S. Rodriguez-Hernandez, and J. Pousada-Carballo, "Analytical evaluation of phm convergence," *IEEE transactions on communications*, vol. 54, no. 9, pp. 1547–1553, 2006.
- [44] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a dual round-robin switch," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 3. IEEE, 2001, pp. 1688–1697.
- [45] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE micro*, vol. 19, no. 1, pp. 20–28, 1999.
- [46] M. Letheren, J. Christiansen, I. Mandjavidze, H. Verhille, M. De Prycker, B. Pauwels, G. Petit, S. Wright, and J. Lumley, "An asynchronous data-driven event-building scheme based on atm switching fabrics," *IEEE transactions on nuclear science*, vol. 41, no. 1, pp. 257–266, 1994.
- [47] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on communications*, vol. 22, no. 5, pp. 637–648, 1974.
- [48] K.-C. Leung, V. O. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (tcp): problems, solutions, and challenges," *IEEE transactions on parallel and distributed systems*, vol. 18, no. 4, pp. 522–535, 2007.
- [49] C. He and K. L. Yeung, "D-lqf: An efficient distributed scheduling algorithm for input-queued switches," in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.
- [50] Y. Shen, S. S. Panwar, and H. J. Chao, "Providing 100% throughput in a buffered crossbar switch," in *2007 Workshop on High Performance Switching and Routing*. IEEE, 2007, pp. 1–8.
- [51] A. Baranowska, G. Danilewicz, W. Kabacinski, J. Kleban, D. Parniewicz, and P. Dabrowski, "Performance evaluation of the multiple output queueing switch under different traffic patterns," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, vol. 1. IEEE, 2005, pp. 5–pp.
- [52] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *25 years of the international symposia on Computer architecture (selected papers)*, 1998, pp. 441–450.
- [53] X. U. G. UG612, "Timing closure user guide," 2012.
- [54] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by birkhoff and von neumann," in *7th Int. Workshop on Quality of Service. IWQoS'99 (Cat. 98EX354)*. IEEE, 1999, pp. 79–86.
- [55] J. W. Brown, "Adaptive network on chip routing using the turn model," Ph.D. dissertation, University of New Hampshire, 2013.



Philippos Papaphilippou received his PhD from Imperial College London in 2021. His PhD was funded by dunnhumby for researching novel accelerators to improve the performance of big data analytics. He is now a senior CPU architect at Huawei Technologies R&D (UK) Limited. His research topics include FPGAs, sorting algorithms, network switches, multi-processor architecture and data science.



Kentaro Sano is the team leader of the processor research team at RIKEN Center for Computational Science (R-CCS), responsible for research and development of future high-performance processors and systems. His research includes data-driven and spatial-parallel processors such as a coarse-grain reconfigurable array (CGRA), FPGA-based HPC, high-level synthesis for reconfiguration, and system architectures for next-generation supercomputing based on the data-flow model.



Boma A. Adhi graduated with Bachelor and Master degrees in Electrical Engineering from University of Indonesia in 2010 and 2013. He obtained his PhD in Computer Science and Engineering from Waseda University in 2020 and is currently a postdoc researcher in the Processor Research Team, RIKEN R-CCS, Japan. His interests include compilers, multicore SoC, and heterogeneous reconfigurable computing.



Wayne Luk is a professor of computer engineering at Imperial College London. He leads the Programming Languages and Systems Section, and the Custom Computing Research Group at the Department of Computing. He is a Fellow of the Royal Academy of Engineering, the IEEE, and the BCS.