ShortPath: A Network-on-Chip Router with Fine-Grained Pipeline Bypassing

Anastasios Psarras, Ioannis Seitanidis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos

Abstract—Scalable Network-on-Chip (NoC) architectures should achieve high-throughput and low-latency operation without exceeding the stringent area/energy constraints of modern Systems-on-Chip (SoC), even when operating under a high clock frequency. Such requirements directly impact the NoC routers and interfaces comprising the NoC architecture. This paper focuses on the micro-architecture of NoC routers and presents *ShortPath*, a pipelined router architecture that can achieve high-speed implementations by parallelizing as much as possible – and without resorting to speculation – the allocation steps involved in the operation of a VC-based router. Most importantly, ShortPath is augmented with a fine-grained pipeline bypassing mechanism, which skips all stages without contention and "fast-forwards" the flits to the first point of contention. Pipeline bypassing in ShortPath is always productive, and even if a flit loses in arbitration, it does not repeat any of the stages already bypassed. Extensive network simulations and hardware analysis – using standard-cell-based synthesis and placed-and-routed layout – corroborate the efficiency of ShortPath, in terms of both network performance and hardware complexity, as compared to the most relevant current state-of-the-art architecture.

Index Terms—Networks-on-Chip, Routers, Pipelined Organization, Pipeline Bypass, VLSI

1 Introduction

The last decade has witnessed a fundamental paradigm shift in digital system design: the transition to the multi-core realm [1]. Multi-core systems have elevated the criticality of the on-chip interconnection fabric, which is now tasked with satisfying amplified communication demands. Networks-on-Chip (NoC) have emerged as the dominant communication medium in multi-core setups, primarily due to (a) their innate physical scalability, which simplifies system integration, and (b) their high-performance characteristics, stemming from the parallel handling of multiple flows traversing the network [2], [3].

Virtual Channels (VCs) are an integral part of most NoC architectures, since they contribute to performance enhancement, and they facilitate network traffic separation. The latter attribute enables, in turn, network virtualization, traffic isolation [4], and quality-of-service provisioning [5]. Moreover, VCs can be used for the definition of Virtual Networks (VNs), which are used to avoid protocol and/or routing deadlocks. Packets belonging to a VN complete their entire trip in the network by traversing the same VN. In-flight transitions from one VN to another are either prohibited, or done with very restrictive rules, which are used to guarantee deadlock freedom in the cases of: (a) separating request/reply traffic [6] and/or other cache coherence message classes [7], and (b) supporting adaptive routing [8].

Being the main building block of the on-chip network, the VC-based NoC router has been the focus of significant research attention, and the target of several micro-architectural optimizations [9], [10]. The goal of this work is to design a scalable VC-based router architecture that strikes a cost-effective tradeoff between latency (in cycles) and clock period (operating frequency), while still offering a low-area/power implementation. Ideally, the NoC router should be able to (1) facilitate the shortest possible intra-router latency to each individual flit (i.e., single-cycle traversal), and (2) its clock period should be as short as possible to enable high-frequency implementations. These desired traits are innately contradictory: high operating frequencies are typically achieved by pipelining, which, inevitably, precipitates commensurate increases in latency (cycles). Our objective is to cost-effectively reconcile these conflicting properties through a re-organization of the NoC router's micro-architecture.

The proposed router micro-architecture, called *ShortPath*, relies on an array of novel micro-architectural innovations, which work synergistically to optimize the router's operation. Overall, the **key attributes and contributions** of ShortPath are the following:

- A collapsible pipelined router organization, which allows for dynamic pipeline-stage bypassing, in order to incur the minimum possible latency to each traversing flit. This is the first (to the best of our knowledge) architecture to guarantee that *each flit will spend the minimum possible number of cycles* in each router, based on prevailing traffic conditions under both low and high network traffic. Spending more than a single cycle in a ShortPath router happens only when flits encounter contention. ShortPath skips all uncontested stages and "fast-forwards" the flits to the first point of contention. As a result, the (nominally) 3-stage pipeline dynamically collapses to the minimum possible number of stages (i.e., 1- or 2-stage), based on traffic conditions.

Anastasios Psarras, Ioannis Seitanidis, and Giorgos Dimitrakopoulos are with the Electrical and Computer Engineering department of the Democritus University of Thrace (DUTH), Xanthi, Greece.
(e-mail: apsarra@ee.duth.gr; iseitani@ee.duth.gr; dimitrak@ee.duth.gr).

Chrysostomos Nicopoulos is with the Electrical and Computer Engineering department of the University of Cyprus, Nicosia, Cyprus.
(e-mail: nicopoulos@ucy.ac.cy).

- A radical re-organization of the NoC router's micro-architecture allows for minimal intra-router latency (in terms of cycles), while still enabling high-frequency pipelining. The router's allocation and switching tasks benefit from a new parallel setup, which is further aided by a pair of instrumental low-cost request queues.
- ShortPath's operation is *always productive*: even if the bypassing flit loses the arbitration at the first point of contention it encounters, it will never be forced to repeat any of the already bypassed stages. This is unlike existing bypassing approaches, which force flits to "rewind" and traverse a longer pipe. The lack of pipeline stage repetitions is the result of the *always non-speculative* operation of ShortPath, which allows each flit to keep (and not kill, as done in speculative architectures) any of the already allocated resources, even if the allocation of the subsequent resources – deeper in the pipeline – is delayed due to contention. This behavior ensures consistent effectiveness (without the performance variability instilled by speculation) under *all* traffic conditions.

The efficacy and efficiency of the proposed architecture are validated using extensive cycle-accurate network simulations and detailed hardware analysis of synthesized and placed-and-routed designs (using commercial-grade 45 nm standard-cell libraries). The obtained results demonstrate the high performance, low complexity, and highly scalable nature of ShortPath, as compared to the state-of-the-art. More specifically, ShortPath is shown to outperform Scorpio [11] – the most efficient pipelined router organization with pipeline-stage-bypassing capabilities – by up to 30% in terms of latency and 25% in terms of throughput, with *no* additional hardware cost (i.e., no adverse impact on area/power/delay). In fact, ShortPath's new micro-architecture yields a shorter critical path than Scorpio [11], which translates to an equivalent increase (if desired) in the maximum operating frequency.

The rest of the paper is organized as follows: Section 2 briefly reviews existing state-of-the-art NoC router architectures. Section 3 introduces the basic pipelined ShortPath architecture, while Section 4 presents ShortPath's fine-grained pipeline-bypassing mechanism. Section 5 presents experimental results and accompanying analysis. Finally, Section 6 concludes the paper.

2 Background & Related Work

The design space of VC-based NoC routers has evolved significantly over the past decade [10]. An abstract illustration of a generic VC-based router architecture is illustrated in Figure 1. Arriving packets are written into the input VC buffers of the router (Buffer Write, BW). In the following cycles, they must find their way to the proper output port, after going through several allocation steps [10]. The head flit of a packet first calculates its destined output port through Routing Computation (RC)[1]. Once the desired output port is known, the head flit uses it to allocate an output VC (i.e., an input VC in the downstream router) during VC Allocation (VA). The allocated VC is "inherited" by all constituent flits of the packet. Then, each individual flit independently tries to gain exclusive access to its desired output port, on a cycle-by-cycle basis, through Switch Allocation (SA). The winners of the SA stage traverse the crossbar during Switch Traversal (ST), and are then written in an output pipeline register. Finally, the flits move to the next (downstream) router through Link Traversal (LT).

1. The selected output port could have been pre-computed in the upstream router, using Look-ahead RC (LRC) [12].

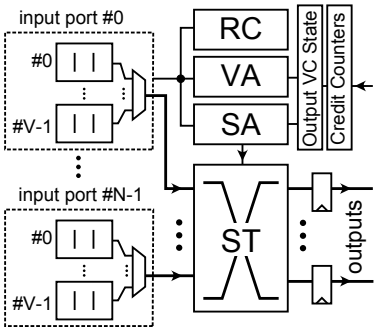

Fig. 1. The main modules of a generic baseline VC-based NoC router.

The router can implement all the required allocation steps in one cycle, or in multiple cycles, when following a pipelined organization. In the baseline case, shown in Figure 2(a), the VA and SA stages are executed serially, one after the other. Only the packets that have allocated an output VC are allowed to move to the SA stage.

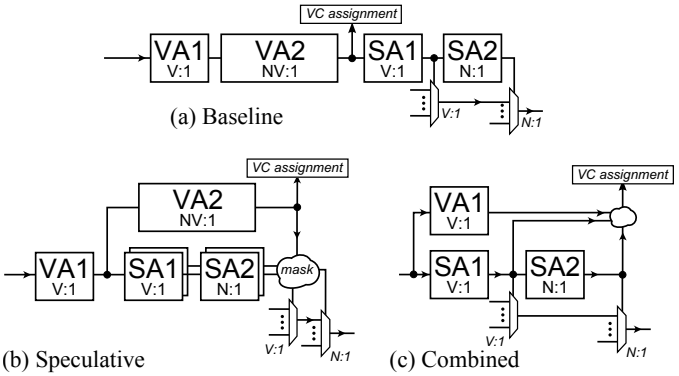

Fig. 2. The sequence/order of arbitrations and their dependencies in (a) baseline, (b) speculative, and (c) combined allocators.

In VA, the head flit of a packet selects one output VC to be requested – among possibly V candidates – in the VA1 phase. Each output VC then selects at most one input VC requesting it, in VA2 ($N \times V$ input VCs in total – there are N input/output ports in a router, with V VCs per input port). The number of requests in VA1 and VA2 is reduced when VCs are used to implement VNs, and each VN serves a subset of the V VCs per input.

Upon completion of VA, the SA stage commences. During SA1, every input port independently selects one input VC that will attempt to reach the selected output port (one out of V input VCs is selected). Thereafter, in SA2, each output port selects which input can access it (N inputs in total). The winners of both the SA1 and SA2 steps are allowed to move to the downstream router. The serial execution of the arbitration steps limits the operating frequency of the router, with VA2 typically being the slowest step.

As an optimization of the baseline setup, the allocation process can be parallelized by speculatively performing SA, in parallel to VA2, as shown in Figure 2(b), thereby allowing a flit to arbitrate for port access, even though it has not been allocated an output VC yet [13], [14]. In the case of misspeculation, a packet (still not owning an output VC) wins in SA, but loses in VA2, and the output port is left unused in the current cycle. In order to decrease the probability of misspeculation, the non-speculative requests (i.e., those made by packets that already own an output VC) have a higher priority over speculative ones. However, this approach requires two full SA units to separately serve speculative and non-speculative requests, and extra masking logic to handle the two allocators' conflicts. The VA1 stage is still performed in series with SA, since we need to know which output VC is selected by each input VC, in order to check its availability (needed in VA2) and the existence of the required downstream credits (needed in SA1 and SA2) [15].

Recently proposed architectures move one step forward – beyond speculative allocation – by removing the VA2 stage completely [15], [16], [17]. The VA process is simplified by combining its functionality with that of SA in what is termed *combined allocation*. Output VC assignment happens in the SA stage, where flits may participate even if they do not own an output VC yet (e.g., newly arrived head flits). If a head flit wins in SA, it receives access to the selected output port and, concurrently, it is assigned to the selected output VC. The assigned output VC is selected in VA1, which is executed in parallel to SA1, by exploiting the fact that it is not necessary to know that the selected output VC is available and with credits, but it suffices to know that at least one of the candidate output VCs fulfills the necessary conditions [18]. The organization of such combined allocators is shown in Figure 2(c). In some implementations [16], [19], [20], the VA1 stage is not explicitly used, and the allocated VC is received by a queue/pool of free VCs at the output.

In addition to optimizing the allocation processes, state-of-the-art router architectures combine fine-grained pipelining with pipeline stage bypassing to reduce the number of cycles spent within each router when the traffic in the router is low. For instance, the 3-stage pipelined router presented in [21] and elaborated in [22] executes SA1, SA2, and ST in different stages, and allows flits to bypass SA1 when only one input VC is active within a specific input port.

SWIFT [19] and Scorpio [11] enhance the bypass option by taking advantage of the fact that SA2 and ST are executed in different pipeline stages, i.e., SA2 finishes one cycle before ST. Hence, the result of SA2 is augmented with additional header information, bundled in so called *Look-Ahead (LA) signals*, and sent to the downstream router to set up the allocation decisions one cycle prior to the arrival of the actual flit. In this way, the incoming flit is able to bypass all intermediate arbitration stages and directly perform ST in the next router. However, if this lookahead procedure fails for any reason (mostly due to contention from other inputs), the flit has to *traverse all 3 stages of the pipeline*. In cases where multiple LA requests are contending for the same output port, the winner is declared through a simple masking process, named LA Conflict Check (LA-CC), instead of a full-fledged arbitration, in an effort to reduce the bypassing delay overhead as much as possible. This approach however, leads to *unproductive decisions*, where none of the contending flits manages to actually succeed in bypass, but instead, all of them are forced to "re-start," in order to contend again in the normal pipeline 3 cycles later.

Additionally, even in normal (non-bypass-capable) pipelined operation, not all allocation steps are used productively when the SA1 and SA2 stages are performed in different pipeline stages. For example, assume that a flit that needs to reach an output VC – which has currently enough credits available – performs SA1 and wins. In the next pipeline stage, the flit participates in SA2, but loses. If the flit only retries the SA2 step (assuming that it remains the winning flit in SA1), then the selected output VC may no longer have credits available. Thus, the flit is unable to move forward and its request – that is now obsolete – should be killed, until the credits of the corresponding output VC are increased [11], [19], [21]. Thus, the SA1 winning step is not used productively (behaving like mis-speculation), and the allocation process should restart from the beginning.

Scorpio [11] constitutes the latest rendition of the three key micro-architectural principles, i.e., combined allocation, pipeline bypassing, and look-ahead signaling, and removes many of the deficiencies of its predecessor [19]. For example, Scorpio uses the same bypass mechanism and LA signals as SWIFT, but in a more cost-effective way, without resorting to Token Flow Control [23], as done in SWIFT [19], which increases buffering requirements and incurs significant wiring overhead across routers. More importantly, Scorpio's critical path is improved, thus reaching higher clock frequencies, while still retaining the favorable characteristics of a balanced pipelined organization and efficient pipeline bypassing. However, Scorpio remains a *speculative* architecture and still suffers from the *inefficiencies of LA-based bypassing*.

Nevertheless, the Scorpio design was actually fabricated in silicon, based exclusively on standard-cell libraries, and incorporated into a fully-functioning 36-core CMP. Clearly, the Scorpio architecture is a provably feasible/practical state-of-the-art implementation of the latest micro-architectural innovations in NoC design. Being the most recent incarnation of all aforementioned techniques, Scorpio [11] is the **most relevant work** to the newly proposed ShortPath architecture. Hence, **Section 5 will present an in-depth and all-encompassing comparison between the two designs**.

3 The ShortPath Router Architecture

This section describes the basic ShortPath pipelined micro-architecture, which comprises three pipeline stages, as depicted in Figure 3. ShortPath is – by construction – *non-speculative* by imposing to head flits a certain order of acquiring the needed resources. A head flit should first allocate an output VC for the whole packet, and then gain access to its output port.

VC allocation is performed in the first pipeline stage (only applicable to head flits) that executes the VA1 and VA2 stages in parallel, driven by a newly introduced FIFO structure, called the *VA Request Queue (VARQ)*. Whenever a head flit arrives at an input port, necessary *control* information is enqueued in the corresponding VARQ. The exact operation

of VC allocation is analyzed in Section 3.1, while the details of the VARQ's operation are presented in Section 3.2.

The second stage allows all flits to perform SA1 and push their requests into the novel *SA Request Queue (SARQ)*, which separates the SA1 and SA2 arbitration stages. Requests to the SA1 stage can only be made by flits that already own an output VC and refer to an output VC that has available credits. Pushing the requests that win in SA1 in the SARQ makes the allocation decisions always productive. Even if a request is not satisfied in SA2, due to contention, it does not have to replay the previous pipeline stage, and it stays in the SARQ. In the third pipeline stage, all flits perform SA2 and the ones that win traverse the crossbar (ST), and are written in the output pipeline register that separates router operation from link traversal (LT). If desired, the SA2 and ST modules could be merged into one combined fast module, using the structure of [24]. The details of the switch allocation performed in the second and the third pipeline stages of ShortPath are clarified in Section 3.3.

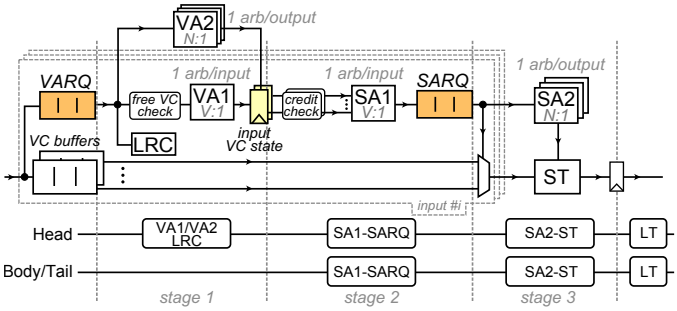

Fig. 3. The organization of the 3-stage pipelined ShortPath router, showing both the input and output arbitration logic, which guarantees non-speculative operation at low cost, and the operations performed for each flit in each pipeline stage.

Although the general order of resource allocation resembles that of a baseline router architecture, ShortPath's allocation logic is: (1) significantly simplified, (2) it allows for fine-grained pipeline partitioning of the allocation tasks, and (3) it enables an always-productive pipeline-stage bypassing mechanism. Note that three-stage router traversal is the *maximum* (worst-case) pipeline depth that a flit may experience. As will be described in Section 4, ShortPath's pipeline is *dynamically collapsible* to the minimum possible depth allowed by the prevailing traffic conditions. Thus, flits may traverse a single-, two-, or three-stage pipeline, based on the encountered contention.

3.1 Virtual-Channel Allocation

In ShortPath, acquiring an output VC (relevant only for the head flits of each packet) is performed in the first pipeline stage and is governed by two allocation rules that greatly simplify the hardware implementation of VC allocation, without affecting negatively networking performance: in each cycle, (a) at most one VC per input port is allowed to allocate an output VC, and (b) at most one output VC may be allocated per output port.

Based on the first rule, only one input VC (still not owning an output VC) from each input port is allowed to allocate an output VC. Specifically, the packet chosen to participate in VC allocation is the earliest arriving one. The order of arrival is maintained through the VARQ, as shown in Figure 3, which enqueues the necessary control information for every arriving head flit. The VARQ only stores **a few bits of control information** for each head flit, i.e., its VC ID, its destined output port (as calculated in the previous router during LRC), and 2-4 bits of metadata required for the VARQ's operation. The actual flit still resides in its designated VC buffer. Once the packet succeeds in allocating an output VC, the control information is dequeued from the VARQ and the next head flit (in order of arrival, as preserved by the VARQ) can allocate an output VC to its destined output port.

Since at most one input VC is allowed to try to allocate an output VC from each input port, and only one output VC can be allocated per output port in a single cycle, there is no need to make any distinction between requests made for different VCs of the same output port. It is merely necessary to select the input port that will receive an output VC in the specific output port. Thus, one arbiter per output port suffices to handle all the requests for that port, even if they refer to different output VCs. This results in a VA2 arbiter (one per output port) with only N inputs (requests), where N is the number of input/output ports in the NoC router. Thus, as opposed to a baseline organization, ShortPath reduces both the number of VA2 arbiters per output (one versus V) and the requests that each arbiter receives (N versus $N \times V$). Even if the baseline VC allocation allows for a marginal increase in throughput, its delay complexity and the delay imbalance that it causes across the pipeline stages, do not justify its adoption.

Once the head flit of an input VC (at most one per input port) wins in VA2, it is assigned to the output VC selected in parallel by VA1. If a packet is allowed to change VC while moving from router to router (i.e., in-flight), the head flit should prepare a vector of candidate output VCs from its destined output port, while the unavailable ones should be crossed off. From those VCs left, the head flit of each packet should choose one. Thus, a single $V : 1$ arbiter per input is enough to perform VA1.

3.2 VARQ Attributes & Operation

The VARQ is a low-cost hardware queue, which complements the VA operation and maintains the order of packet arrival into each input port. Each incoming *head* flit (i.e., a new packet) stores its VC ID and destined output port ID into an empty slot in the VARQ. The head-of-line entry of the VARQ feeds this pertinent control information to the VA logic of the router, thereby enabling the corresponding head flit to compete in the VA stage for an output VC.

Since packets from different VCs enqueue control information in a single VARQ, it is imperative to ensure that this serialization does *not* create any dependencies among the VCs. The VARQ achieves this feat by employing a *rotation mechanism*. If the packet at the head-of-line position of the VARQ is unable to find a free output VC (i.e., all output VCs are already allocated), the VARQ "rotates" this entry by sending it to the back of the VARQ. This rotation allows the next VARQ request in line to move into the head-of-line position. Thus, if a packet is unable to find available

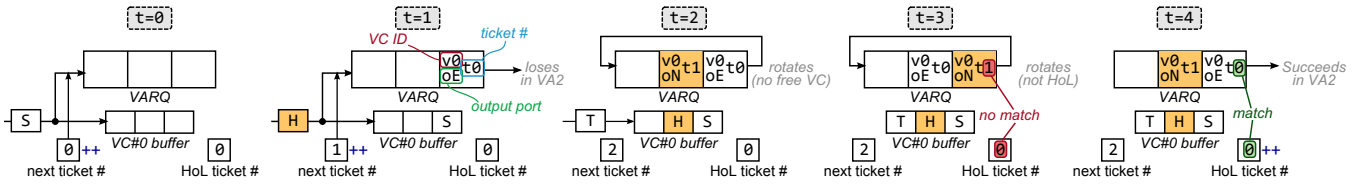

Fig. 4. A cycle-by-cycle example of the operation of ShortPath's VARQ. Only the activity of a single input VC is depicted here for clarity.

VC resources, subsequent packets belonging to different VCs will not be blocked indefinitely from participating in VA2. Since *a packet of one VC can never block the progress of a packet from another VC* (due to the rotation mechanism), the proposed VARQ *guarantees* the absence of across-VC dependencies and, thereby, ensures deadlock freedom. Moreover, the rotation mechanism preserves all the Quality-of-Service (QoS) properties potentially imposed by switch allocation, when VCs are used as an isolation mechanism for the various QoS-driven message classes in the system.

However, the rotation process allows requests (already enqueued in the VARQ) that do not correspond to head-of-line packets in the actual VC buffers to appear at the head of the VARQ. Additionally, the continuing arrival of new packets – and, thus, new VARQ entries – while the rotation takes place may mix up the request order within the VARQ. Hence, the VARQ employs a simple mechanism to keep track of the VARQ request order to ensure that no VC allocation request is generated, unless it refers to a packet that is at the head of its corresponding VC buffer.

Each VC keeps two separate counters, each one storing a so called *ticket* #: (1) the *next ticket* # counter, and (b) the *HoL ticket* # counter. The size of both counters is extremely small (a few bits wide), since the *ticket* # field is bounded by the VC buffer depth. These two counters are part of the **three basic rules** governing the operation of the VARQ:

- Whenever a new packet arrives at a VC, it enqueues the following control information in the VARQ: (a) its VC ID, (b) its destined output port, and (c) the *ticket* # value currently stored in the *next ticket* # counter of the corresponding VC. This ticket # indicates the new packet's order of arrival among packets of the same VC. Once the new incoming packet receives its ticket #, the *next ticket* # counter is incremented.
- The request at the head of the VARQ is allowed to participate in VA2 only if its *ticket* # matches the value in the *HoL ticket* # counter of the corresponding VC. In other words, the request at the HoL position of the VARQ is only allowed to compete in VA2 if it belongs to the HoL packet in the actual VC buffer. If not, the **request rotates** to the back of the VARQ.
- If the *ticket* # of the request at the head of the VARQ matches the value in the *HoL ticket* # counter, the packet is allowed to participate in VA2. If there are *no available/free output VCs*, the HoL VARQ **request rotates** to the back of the VARQ. If there is at least one available output VC, the packet competes in VA2. If the packet succeeds, the *HoL ticket* # counter is incremented (to point to the next packet eligible to perform VA). If the packet loses in VA2, it retries in the following cycle, as long as the requested output VCs are still available. Simply losing in arbitration is not a reason for rotating at the end of the VARQ.

A cycle-by-cycle example of the VARQ operation is depicted in Figure 4, which focuses on the activity of a single input VC for clarity. A single-flit packet ('S') arrives at VC#0 in cycle 0, and a VARQ entry is generated, which consists of its VC ID (0), its destined output port (East), and its ticket # (0). At the end of this cycle, the *next ticket* # counter is incremented by one. In the following cycle (t=1), a new head flit ('H') arrives at VC#0 and generates a VARQ entry with the new ticket # of 1. Again, the *next ticket* # counter is incremented by one. In this cycle (t=1), the single-flit packet 'S' loses in VA2 arbitration, so it retries in cycle 2. This time (t=2), all output VCs of its destined output port are occupied, causing the VARQ to perform a rotation (as per the third rule above). However, the ticket # of the next VARQ entry in line does not match the value in the *HoL ticket* # counter, causing another rotation in cycle 3. In the next cycle (t=4), the request of the head-of-line packet in VC#0 reappears at the head of the VARQ, with a matching ticket #. Assuming that an output VC has been freed, the packet participates in VA2 and succeeds, causing a dequeue in the VARQ, and an increment of the *HoL ticket* # counter.

The total number of packets (or head flits) that can be present per input port of the router also determines the depth of the VARQ. In the worst case, when the input VC buffers are full with single-flit packets, the VARQ should hold $V \times B$ pieces of control info, where B represents the buffer depth per VC. Thus, even for small values of V and B, the depth of the VARQ grows quickly.

However, the depth of the VARQ can become significantly smaller than the worst-case scenario, by taking into account one important characteristic of NoC traffic. On average, the number of packets present in each input port is equal to $V \times B/avg\_packet\_size$. In most NoC configurations, the packets that flow in the network have multiple lengths. Read requests or control packets typically have lengths of 1 to 2 flits, while reply or write-request packets can have arbitrary sizes larger than 2 [25], [26]. The packet size also depends on the width of the links of the NoC. The average packet size is the average packet length seen by the NoC, after taking into account the distribution of each packet type in the injected traffic. For realistic scenarios, that includes both single- and multi-flit packets selecting a VARQ depth that is significantly shallower than the worst-case of $V \times B$ is enough for not compromising performance. Hence, it is a reasonable choice to limit the depth of the VARQ by controlling the number of *packets* (not flits) that can be present in the VC buffers of each input port of the router. Note that this limitation on the number of packets has **absolutely no impact on the performance of ShortPath**, as will be shown in Section 5.

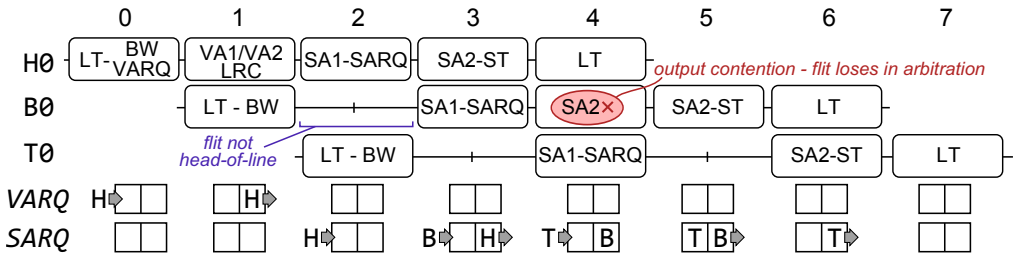

Fig. 5. A cycle-by-cycle example of the operation of ShortPath's 3-stage pipeline. The activity of only one input port is depicted here for clarity.

To limit the number of packets present in each input port, we maintain a counter at each output port of the router, which counts the number of packets (head flits) that have already passed from the corresponding output port. If the maximum packet number has been reached (this number is elaborated in Section 5), the VC allocator stops allocating any new output VC for the particular output port. In any other case, the VC allocation operates normally. For every head flit that leaves the output, the packet counter is incremented. Once the tail flit of a packet is dequeued from the VC buffers of the downstream router, the packet counter is notified and decreased, restoring the operation of the VC allocator for that output port. In order to avoid the scenario of a single VC consuming all of the packets of an output port, thereby introducing dependencies among VCs, multiple counters (one per VC) may be added, to guarantee that at least one packet is reserved for each output VC. The negligible overhead of these counters is accurately accounted for in the hardware evaluation of Section 5.1.

3.3 Switch Allocation

The flits belonging to packets that have already been allocated with an output VC can participate in SA, which is performed in two steps. During SA1, every input port independently selects one input VC that will attempt to reach the selected output port. Thereafter, in SA2, each output port selects which input port can access it. The winners of both the SA1 and SA2 steps traverse the crossbar (ST) and are written to an output register.

State-of-the-art implementations, such as [21] and [11], execute SA1 and SA2 in different pipeline stages, following a prediction-based allocation policy. Prediction is required, since the SA1 stage is only informed of the result of SA2 in the following cycle. If the winner of SA1 acts pessimistically and predicts that it will lose in SA2, it retries again in SA1 in the next cycle by sending a second request. If the prediction turns out to be wrong, and the first request is actually granted, then another request will appear at SA2 in the next cycle, which is possibly obsolete and should be killed (if no other flit follows in the input VC buffer, or credits are no longer available). Alternatively, if the winner of SA1 acts optimistically and does not send a second request (assuming that it will win in SA2), it may waste the successful SA1 allocation step, if the prediction turns out to be wrong and the flit actually loses in SA2.

On the contrary, ShortPath *avoids any predictions*, by guaranteeing that the winner of the SA1 stage has allocated all resources required (i.e., an output VC with credits) to be eventually granted in the next stage, SA2. Once a flit wins in SA1, necessary **control information** – the VC ID and the output port request of the winning flit – are enqueued into the SARQ (again, the actual flit data remains in the input VC buffers). The port request at the head of the SARQ feeds the corresponding SA2 arbiter, and the VC ID drives the $V : 1$ per-input data multiplexer to select the appropriate input VC's data (actual flit). If the SARQ is full, SA1 in that input port is blocked. When multiple flits exist in a certain input VC, multiple requests are sent to SA1 in consecutive cycles, causing multiple requests to be pushed to the SARQ (even if the result of SA2 is not yet known for the previous requests). Eventually, once the input succeeds in SA2, a dequeue operation takes place in both the input VC buffer and the SARQ, thereby allowing the next request in line to appear. If a request is not satisfied in SA2, due to contention, it does not have to replay the previous pipeline stage, and it stays in the SARQ, thus making the SA1 decision always productive.

A cycle-by-cycle example of the operation of ShortPath's 3-stage pipeline is shown in Figure 5. For clarity, the activity of only one input port is depicted. In cycle 0, a head flit (H0) arrives in VC 0, it is written in the input VC buffer (BW), and its VC ID is enqueued into the VARQ. In the following cycle, H0 successfully allocates an output VC in VA, as the following flit (B0) arrives and is stored behind it in the input VC buffer. In cycle 2, H0 performs SA1 and successfully enqueues a request into the SARQ, as seen at the bottom of Figure 5 (recall that only request identifiers are enqueued; no data leaves the input buffer). In cycle 3, H0 wins in SA2 and traverses the crossbar, while B0 participates in SA1 and wins, thereby enqueuing its request for SA2 into the SARQ. The same procedure is followed by the tail flit (T0) in cycle 4. At this point, however, B0 loses in SA2 by a flit from a different input port (not shown), and, thus, no dequeue operation occurs in the SARQ. The flit retries and succeeds in cycle 5, allowing T0's request to appear at the head of the SARQ in the next cycle.

Whenever the input VC enqueues a request into the SARQ, it also consumes a credit in the selected output VC. This ensures that the input side always stays in sync with the output VC's actual buffer state. This premature credit consumption cannot create any problems, since a request in the SARQ will succeed in SA2 within a finite amount of time. This is guaranteed by updating the priority of the SA2 arbiters whenever a grant is generated; thus, once a request appears at the head of the SARQ, it is **guaranteed to succeed after at most** $N - 1$ **cycles**. Since every request entering the SARQ (a) refers to an already allocated output VC, and (b) it has already consumed an existing credit (i.e., space is

guaranteed in the downstream router), *no* entry in the SARQ can cause indefinite blocking of any other entry. Hence, **the SARQ creates no dependencies and is deadlock-free by construction**.

For credit management, we adopt the proxy counter approach of [15]. In this case, once a head flit is allocated an output VC, its credit value is copied locally (per input VC) to a proxy counter. Thus, credit checking is done locally per input VC, without additional propagation and multiplexing overhead. Multiplexing is only used for updating the value of the proxy credit counter. Note that, due to the VA policy restrictions, at most one output VC credit value from each output may be copied to the input VC side in each cycle.

4 Fine-Grained Pipeline Bypassing

Pipelining translates into additional cycles to a packet's latency. Even if this extra latency is amortized by the increase in clock frequency (due to pipelining), the ability to skip pipeline stages whenever possible – with only a marginal impact to the clock frequency – will always be beneficial.

Ideally, pipeline bypassing should be enabled, or disabled, in any arbitration stage, depending on the contention encountered in each stage. For example, whenever a head flit is (a) the only flit in a certain input port, and (b) the only flit requesting a particular output port, then said flit should spend at most one cycle in the router. ShortPath's fine-grained pipeline bypassing mechanism facilitates the bypassing of *each and every pipeline stage* (whenever possible), under both low and high traffic load. In fact, any flit traversing a ShortPath router is guaranteed to spend *the minimum possible number* of cycles in each router, depending on the usage of the input/output ports.

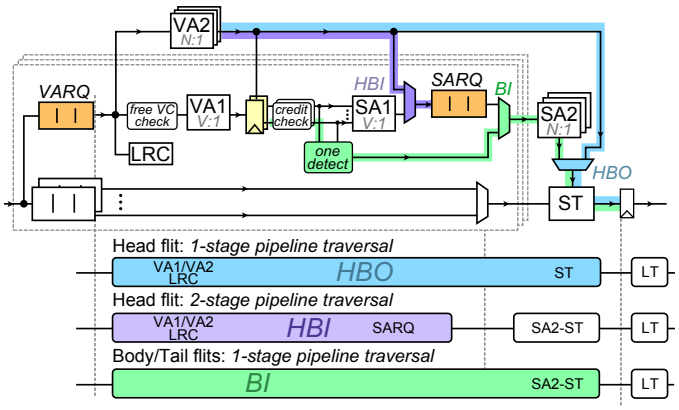

Fig. 6. The fine-grained pipeline-stage bypassing mechanism employed by ShortPath. Each individual pipeline stage may be skipped on-demand, and any flit traversing a ShortPath router is guaranteed to spend the minimum possible number of cycles in each router, based on the usage of the input/output ports.

In ShortPath, the head flit completes VA2 and is assigned to an output VC of a certain output port. If the SA2 stage corresponding to the same output port does not see any other request, then the grant of VA2 is, actually, the only grant given by this output port to any input VC. Therefore, the *VA2 grant can be re-used* and act as an SA2 grant, too. This feature is *unique* to ShortPath, which imposes the rule that at most one output VC can be assigned per cycle. In this way, VA2 bypasses both SA1 and SA2, and it allows the head flit to move directly to ST using the Head Bypass Output (HBO) multiplexer, as illustrated in Figure 6 (**blue path**). Note that the HBO path refers to control signals only – the actual flit still uses the input's multiplexer to access the crossbar in ST. Additionally, in terms of hardware delay, both the normal SA2-ST path and the bypass VA2-ST path exhibit the same delay, since the VA2 step in ShortPath is implemented using just a single $N:1$ arbiter per output port (similar to SA2).

The head flit may be impeded from traversing the router in a single cycle in the following three cases: (a) the flit's destined output port is currently being requested by another flit in SA2, either from the same, or from a different input port; (b) the head flit loses in VA2; or, (c) another flit makes a request in SA1 from the same input port. In all three cases, the bypass opportunity is only lost due to contention, and not because of any micro-architectural limitation. Actually, the same behavior is expected under the same traffic conditions in a single-cycle (non-pipelined) router.

It should be noted that the pipeline-bypass operation in ShortPath always makes a productive step. If, for instance, the head flit is the only one in its input port making a request, while there are many contenders for the selected output port (i.e., it cannot bypass SA2), then the flit can alternatively bypass the SA1 stage (which is uncontested) and access directly the SARQ. In that case, the head flit will be able to traverse the router in two cycles. In the first cycle, it completes output VC allocation (VA1/VA2) and moves directly in the SARQ, from where it can reach SA2 in the next cycle. This second bypass path is enabled by the Head Bypass Input (HBI) multiplexer in Figure 6 (**purple path**). The addition of this input bypass functionality enables the overlapped (in time) execution of VA and SA in a fine-grained manner; parallelism is even enabled for the first time across the smaller arbitration stages of VA1 ($V:1$ arbiter), VA2 ($N:1$ arbiter), and SA1 ($V:1$ arbiter).

ShortPath's fine-grained bypassing skips all uncontested stages, and "fast-forwards" a flit to the first point of contention. Even if the bypassing flit loses the arbitration at the first point of contention it encounters, it will never be forced to repeat any of the already bypassed stages. This is unlike existing bypassing approaches, which force flits to "re-wind" and traverse a longer pipe [11], [19], [21]. The truly non-speculative operation in each stage of ShortPath's pipeline and the careful use of queues (VARQ and SARQ) – instead of mere pipeline registers – is the enabler for this behavior. Most importantly, this bypass functionality is achieved without changing the flow control, and without introducing any look-ahead signals across routers, as needed in [11], [19]. Additionally, whether flits follow a bypass path, or not, depends solely on the traffic conditions (absence of contention, or not) of the current cycle. There is no reliance on past-cycle (probably stale) traffic conditions, or input-output matchings, to make future bypass [11], or allocation decisions [27], [28], [29].

Spending a single cycle within the ShortPath pipeline is also possible for all flits that have already been allocated with an output VC. If a flit finds no contention at its input port, i.e., no other input VCs are active and the SARQ is empty, the flit is allowed to bypass the SA1 stage and directly participate in SA2 through the Bypass Input

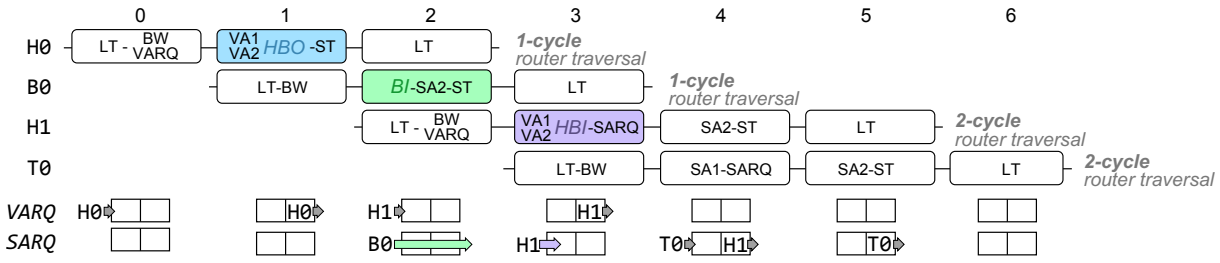

Fig. 7. A cycle-by-cycle example of the operation of ShortPath's fine-grained pipeline bypassing mechanism. All bypass paths are exercised here, leading to single-cycle router traversal for the H0 and B0 flits, and two-cycle router traversal for H1. Due to contention, the T0 flit cannot exploit any bypassing opportunities.

(BI) multiplexer, as shown in Figure 6 (**green path**). If no contention is encountered at the output port, the flit is granted access and departs immediately. If a flit bypassing SA1 encounters contention in SA2, it pushes its request in the SARQ, thus having the chance to retry SA2 directly in the following (or a subsequent) cycle. On the other hand, if the SA1-bypassing flit actually wins the SA2 arbitration, then no request is pushed into the SARQ.

Figure 7 presents an example of the operation of ShortPath's pipeline, which includes bypassing functionality. For clarity, we assume arrival of flits at only one input port of the router. In cycle 0, a head flit arrives at input VC 0, and its VC ID is enqueued into the VARQ. The flit succeeds in VA in cycle 1 and, since there is no other input VC active in the same input port, it tries to reuse the result of VA2 to immediately reach ST. Assuming there is no output contention, i.e., no other SA2 requests exist for that output port, the flit successfully bypasses SA2 using the HBO multiplexer (see Figure 6) and moves directly to the output. In the next cycle (cycle 2), a newly arrived body flit (B0) is the only active flit in its input port, and can, therefore, completely bypass the SA1 step using the BI multiplexer (Figure 6) and participate in SA2. As the flit wins in SA2 and moves forward, a new head flit (H1) arrives at input VC 1. The H1 flit succeeds in VA in cycle 3, but we now assume that its destined output port is also being requested by a flit from a different input port, thus blocking H1 from directly reaching the output. However, H1 is alone in its input port and can, therefore, bypass the SA1 stage and directly reach the SARQ through the HBI multiplexer in Figure 6; i.e., H1 effectively "fast-forwards" to the first contested junction in the pipeline. This scenario does not apply to subsequent flit T0, which has no bypassing possibilities in cycle 4, since flit H1 is trying to access ST. In cycle 6, all flits have left the router in the least possible number of cycles, by exploiting all bypass opportunities. The extra cycle spent by H1 is simply the result of *contention* and not a limitation of the router's micro-architecture; the output port was still utilized in that cycle, albeit by a different input port.

In general, spending more than a single cycle in a ShortPath pipelined router happens only when flits encounter contention. At ultra-low loads – e.g., as observed under bursty traffic in a single input VC – a packet's head flit would perform VA2-ST in a single cycle, while the other flits (body and tail) would perform SA2-ST, uninterrupted. As traffic increases across different input ports, VA2/SA2 contention also increases, and pipeline bypass opportunities are reduced. Flits may need to wait more cycles to be assigned an output VC in VA2, or wait for the SA1 winners in their input port to eventually win arbitration in SA2. Nevertheless, this is a latency overhead that packets would suffer in any NoC router architecture.

5 Experimental Evaluation

In this section, we evaluate the performance of the ShortPath architecture and compare it with Scorpio [11], which represents the most efficient router micro-architecture available in the open literature, in terms of hardware complexity and network performance.

5.1 Hardware complexity

ShortPath and Scorpio [11] routers (with look-ahead RC [12] in both cases) were implemented in SystemVerilog, mapped to a commercial low-power 45 nm standard-cell library under worst-case conditions (0.8 V, 125 °C), and **placed-and-routed** using the Cadence digital implementation flow. Our goal is to evaluate the hardware complexity of both routers under comparison for various number of input and output ports and number of VCs per input port.

In all NoC configurations, the flit width was set to 64 bits and each VC buffer can host 5 flits. For ShortPath, the VARQ was sized to 6 positions, meaning that it could host a maximum number of 6 packets per input port. Note that our network performance experiments indicate that increasing the maximum number of allowed packets to values higher than 6 (i.e., increasing the VARQ depth beyond 6) does not yield any further improvement in latency/throughput. The SARQ can host up to 2 pending requests for the third pipeline stage.

The area/delay curves were obtained for all designs, after constraining appropriately the logic-synthesis and back-end tools using the same parameters for all designs, and assuming that each output is loaded with a wire of 2 mm. During synthesis and placement-and-routing, we followed a mixed V_t approach, where a mix of regular V_t (RV_t), Low-V_t (LV_t), and High-V_t (HV_t) cells were used for the various implementations. LV_t cells can switch at a much faster speed than HV_t cells, at the cost of extra leakage power. All circuits were optimized by using LV_t cells on critical paths, while HV_t cells were employed on the non-critical paths to reduce leakage power [30].

Figure 8 shows the area-delay behavior of both routers under comparison, for 2 and 4 VCs per input port assuming

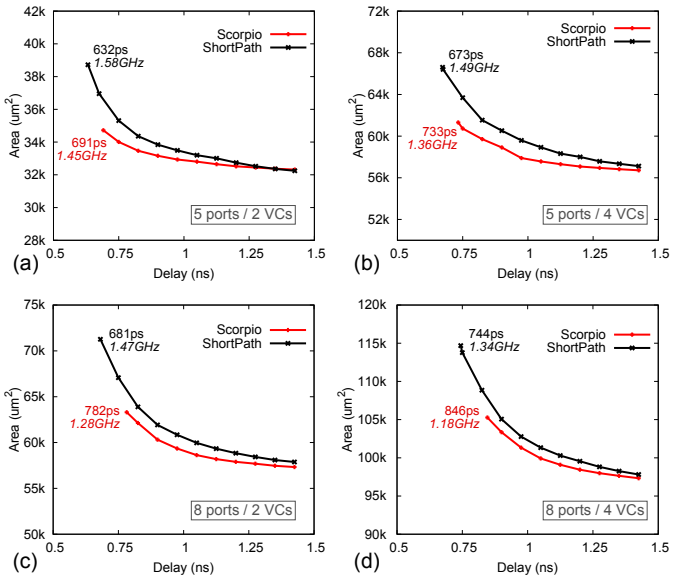

Fig. 8. Hardware implementation (area/delay) comparison of 5- and 8-port ShortPath and Scorpio [11] 3-stage pipelined organizations (including pipeline bypassing), for 2 VCs, and 4 VCs per input port.

5-port routers, as needed by a 2D mesh network, and 8-port routers that can be used in higher-radix topologies [31], or in 2D meshes that employ concentration [32]. In all cases, ShortPath is faster than Scorpio, achieving a 8% and 12% smaller minimum delay (i.e., critical path delay), on average, for the case of 5- and 8-port routers, respectively, due to its highly parallel allocation architecture.

It should be noted that the delay numbers reported here correspond to a low voltage of 0.8 V, which significantly increases the delay of the circuits. For example, a close inspection of the clock frequency of ultra-fast, 3-stage commercial routers [33], [34] optimized at the transistor level – which offers additional speed benefits over standard-cell-based design – reveals that their frequency marginally surpasses the 1 GHz mark when operated at 0.8 V.

ShortPath routers are equally area-efficient as Scorpio when sized under *equal delay*. Specifically, in the case of 5-port routers, at Scorpio's maximum operating frequency, ShortPath presents a minimal 5% area overhead, which decreases to 3% for the case of higher radix 8-port routers. Although the two architectures are based on different pipeline approaches, they eventually both require equal resources. For instance, even though ShortPath's use of the queues (VARQ and SARQ) presents an area overhead, an equivalent cost is paid by Scorpio, due to its data registers for the separate ST pipeline stage and the handling of the lookahead signals.

The hardware complexity analysis is completed by reporting the energy behavior of the routers under comparison. Energy (or area) comparisons are meaningful when the compared circuits are optimized for the same delay. Based on the delay profile reported in Figure 8, we select the designs that correspond to a delay of 850 ps for the cases of both 2 and 4 VCs. The energy consumed in each case is reported in Figure 9. The energy analysis is reported after taking into account all layout parasitics, as done in delay analysis, while the switching activity has been computed using delay-accurate simulations of the derived logic-level netlists. The evaluated routers are all driven by the same arriving packet sequence, which mimics uniform random traffic of 1-flit and 5-flit packets at an injection rate of 0.2 flits/cycle. The traffic characteristics determine the header contents of each packet, while the data contents – i.e., the payload – of each packet is produced using a uniform random number generator. In all cases, the energy required to drive the output links is also included. The results in Figure 9 clearly indicate that the energy efficiency of both architectures is comparable, i.e., ShortPath achieves, under equal delay, slightly larger – but comparable – energy per cycle than Scorpio, for both 2- and 4-VC configurations.

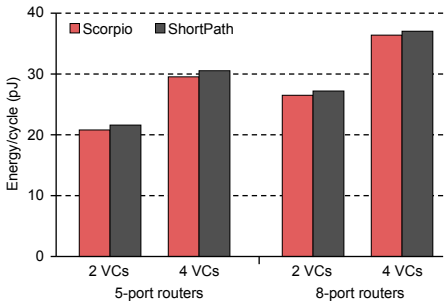

Fig. 9. The energy per cycle (in pJ) expended in ShortPath and Scorpio [11] routers, when operated and sized under equal delay (850ps), for 5- and 8-port routers, with 2 and 4 VCs per input port.

5.2 Network performance

Network performance comparisons were performed using the same SystemVerilog RTL models, thus offering maximum safety in terms of correctly/accurately modeling the micro-architectural components of a NoC router. We employ an 8×8 2D mesh network using 5-port routers with XY dimension-ordered routing, and each router supports 4 VCs per input port using the same parameters, in terms of buffering, as in the case of the hardware complexity evaluation.

In all cases, the injected traffic consists of two types of packets to mimic realistic system scenarios: 1-flit short packets (just like *request* packets in a CMP), and longer 5-flit packets (just like *response* packets carrying a cache line). For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, 1-flit packets, and the rest being long, 5-flit packets, in accordance to recent studies [35]. In many cases, the percentage of 1-flit packets may be even higher. However, this behavior – as tested by additional experimental results – does not change the relative performance differences between the routers under comparison. The 1-flit and 5-flit packet sizes are typical of general-purpose CMP environments. In application-specific embedded systems, larger packet sizes may also be encountered. To account for these scenarios, we have also conducted extensive experiments with larger packet sizes (up to 20 flits per packet). However, since no changes were observed in the overall trends, those results are omitted for brevity.

The first set of performance evaluation experiments involves Uniform Random (UR) and Bit Complement (BC) permutation traffic. In UR traffic, every node sends its packets to all other nodes of the network with equal probability,

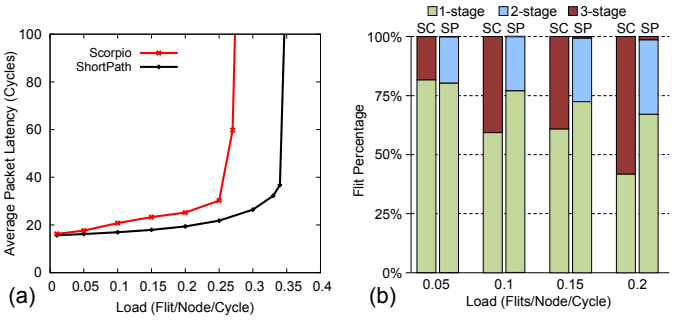

Fig. 10. (a) Latency vs. load curves, and (b) percentage of flits traversing the 1-, 2-, or 3-stage router pipeline under various loads, for an 8×8 2D mesh network using ShortPath ("SP") and Scorpio ("SC") routers under *uniform random* traffic.

while in BC, every node produces packets destined to a single node, whose address is the bit-wise complement of the source node. Other permutation traffic patterns have been tested and exhibit equivalent behavior. The results for UR and BC are depicted in Figures 10(a) and 11(a), respectively. In order to get a better insight into the bypassing efficiency of the two architectures and better interpret the latency results, we also plot the percentage of flits that traverse single-, two-, or three-stage router pipelines under various loads. The flit percentages for Scorpio ("SC") and ShortPath ("SP") are presented in Figures 10(b) and 11(b) for UR and BC traffic, respectively. Note that the number of pipeline stages traversed by a flit does not directly translate to an equal number of cycles spent for intra-router traversal. In fact, flits may actually spend more cycles in the router while waiting in the input buffers behind other flits, or while waiting to be granted access in various allocation stages.

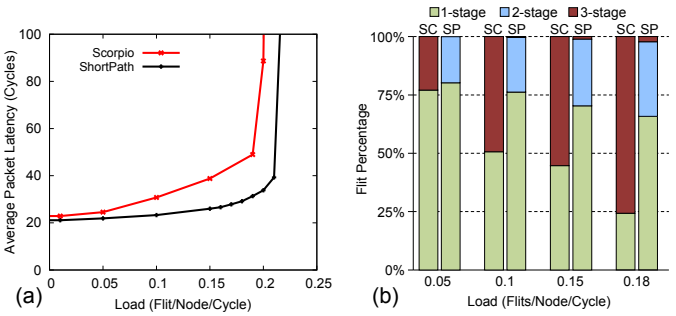

Fig. 11. (a) Latency vs. load curves, and (b) percentage of flits following the 1-, 2-, or 3-stage router pipeline under various loads, for an 8×8 2D mesh network using ShortPath ("SP") and Scorpio ("SC") routers under *bit complement* traffic.

At very low loads, both designs perform equally well, in terms of latency, since bypass paths are frequently activated and most flits spend a single cycle in each router. However, at medium and high loads, contention stresses the bypassing mechanism, causing Scorpio to miss bypassing opportunities, mainly due to its allocator's inherent speculation, and, partly, because of the absence of actual arbitration in conflicting LA bypassing requests. In any case, whenever bypass fails, Scorpio forces a full 3-stage pipeline traversal, as seen in Figures 10(b) and 11(b). Instead, ShortPath dominates at such loads, exhibiting up to 30% lower packet latency at medium loads, due to its dynamically collapsible pipeline. The latter allows flits to traverse only as many pipeline stages as imposed by the first point of contention. Apart from its efficient bypass mechanism, ShortPath's completely non-speculative and always productive allocation logic allows it to sustain much higher loads, leading to increased saturation throughput by 21% and 9% under UR and BC traffic, respectively.

In addition to using purely synthetic traffic patterns, we also employ traffic patterns that are derived from real application workloads. Specifically, we employ the hot-spot traffic model from [36], which synthesizes traffic that closely resembles the traffic behavior of PARSEC application benchmarks [37] running on a CMP. Under this PARSEC-derived Hot-Spot (HS) traffic pattern, 20% of the nodes receive $50\times$ more traffic than the rest, while the remaining injected traffic is uniformly distributed to all other destinations. Routers support 3 Virtual Networks, as required by the MOESI cache coherence protocol, with each one assigned a single VC (3 VCs in total). In order to mimic the behavior of real applications, VCs 0, 1 and 2 receive 77%, 22% and 1% of the injected traffic respectively, while packet distribution is skewed, with 1- and 5-flit packets being 70% and 30% of the total packets injected respectively [36]. As seen by the obtained results in Figure 12(a), ShortPath offers the highest saturation throughput, which is increased by more than 20% relative to Scorpio. It is worth mentioning that the PARSEC-derived HS traffic pattern is dominated by 1-flit packets, which significantly lower the average packet size, thereby increasing the total number of packets present in each input port, as described in Section 3.1. Despite this increase, however, the chosen VARQ depth of 6 was still sufficiently large to *not* generate any throttling.

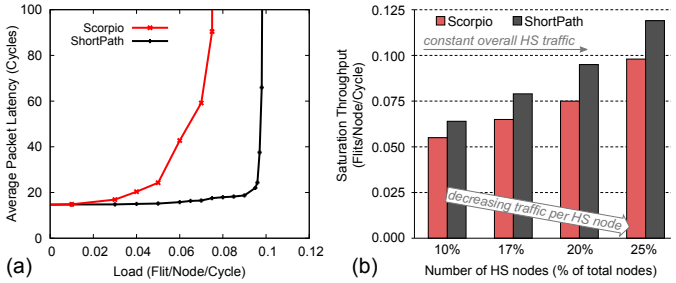

Fig. 12. (a) Latency vs. load curves for 8×8 2D mesh networks using ShortPath and Scorpio routers, under the PARSEC-derived HotSpot (HS) traffic model of [36], which closely resembles the traffic patterns of PARSEC application benchmarks [37]. (b) The saturation throughput of both router designs when decreasing the number of HS nodes.

In order to provide a better estimation of the performance under the PARSEC-derived traffic patterns, we also test scenarios with varying number of HS nodes and measure the saturation throughput of each design. In all experiments, the percentage of HS-directed traffic, the packet distribution, and the load for each VC are held constant, as described above. Since all HS nodes always receive $50\times$ more traffic than the rest, as the number of HS nodes is *increased*, the total traffic directed to these nodes *decreases*, thus causing less congestion around these nodes. The effect can be seen in the results of Figure 12(b), where the throughput of both designs increases, as the number of HS nodes increases. Still, in all cases, ShortPath outperforms Scorpio,

demonstrating an average increase of 18% in saturation throughput.

It should be noted that the new micro-architecture constitutes a fundamental overhaul of the router operation, which yields improvements under all traffic conditions. Hence, a different traffic pattern (as generated by some other benchmark application) would not change the observed trends; it would simply change the amount of performance improvement achieved by ShortPath.

In summary, ShortPath outperforms Scorpio in all conducted experiments, both in terms of latency and throughput. Moreover, since ShortPath provides a faster implementation (due to its shorter critical path), if one also takes into account the **maximum operating frequency** of each design, the achieved performance gains are magnified. For example, ShortPath's saturation throughput, when **measured in Gbps**, is increased – as compared to Scorpio – by 27%, 17%, and 35% under UR, BC, and PARSEC-derived HS traffic, respectively.

6 Conclusions

As the number of network nodes in multi-core systems increases, NoC routers should be able to operate under high frequencies through pipelining, while, at the same time, offering low-latency packet transmission through efficient pipeline bypassing. In this work, we present ShortPath, a high-speed router architecture that manages to offer both traits: a higher operating frequency, and an efficient allocation and pipeline-bypassing mechanism that achieves singificant latency reduction and throughput increase, as compared to a current state-of-the-art router architecture [11].

These performance attributes of ShortPath stem from the novel re-organization of the router's allocation logic, which allows – for the first time, to the best of our knowledge – a truly non-speculative and parallel allocation to be applied to pipelined routers combined with fine-grained pipeline-bypass capabilities. The key distinguishing features of ShortPath (relative to state-of-the-art architectures) are the generation of always-productive pipeline bypass steps, and the ability to "fast-forward" the flits to the first point of contention. These features cohesively yield a smooth linear increase in average packet latency with increasing load. Finally, since ShortPath is orthogonal to the link-level flow control policy, and it does *not* require any look-ahead signals to achieve its bypass functionality, it can easily incorporate partitioned input speedup as a low-cost vehicle (only modest additional area, with no adverse impact on delay) to further increase the saturation throughput.

References

[1] S. Borkar and A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, May 2011.

[2] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[3] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proc. of the 38th Design Automation Conference (DAC)*, Jun. 2001.

[4] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation," in *Proceedings of the 2015 Design, Automation & Test in Europe*, 2015, pp. 1090–1095.

[5] J. W. Lee *et al.*, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *ISCA*, 2008, pp. 89–100.

[6] A. Hansson, K. Goossens, and A. Rădulescu, "Avoiding message-dependent deadlock in network-based systems on chip," *VLSI design*, vol. 2007, 2007.

[7] M. M. K. Martin and et al., "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[8] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 12, pp. 1320–1331, 1993.

[9] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[10] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, *Microarchitecture of Network-on-Chip Routers: A designer's perspective*. Spinger, 2014.

[11] B. Daya, C.-H. Chen, S. Subramanian, K. Woo-Cheol, P. Sunghyun, T. Krishna, J. Holt, A. P. Chandrakasan, and L. Peh, "Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *International Symposium on Computer Architecture*, June 2014, pp. 25–36.

[12] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, 1997.

[13] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. Intern. Symp. on High-Performance Computer Architecture*, Jan. 2001, pp. 255–266.

[14] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proceedings of the International Symposium on Computer Architecture*. IEE, June 2004, pp. 188–197.

[15] D. Becker, "Efficient microarchitecture for network-on-chip routers," Ph.D. dissertation, Stanford University, 2012.

[16] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. K. Jha, "A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos," in *Proc. of the Intern. Conf. on Computer Design*, Oct. 2007, pp. 63–70.

[17] Y. Lu, C. Chen, J. V. McCanny, and S. Sezer, "Design of interlock-free combined allocators for networks-on-chip," in *EEE 25th International SOC Conference (SoCC)*, 2012, pp. 358–363.

[18] I. Seitanidis, A. Psarras, E. Kalligeros, C. Nicopoulos, and G. Dimitrakopoulos, "ElastiNoC: A self-testable distributed vc-based network-on-chip architecture," in *International Symposium on Networks-on-Chip (NOCS)*, 2014, pp. 135–142.

[19] T. Krishna, J. Postman, C. Edmonds, L.-S. Peh, and P. Chiang, "Swift: A swing-reduced interconnect for a token-based network-on-chip in 90nm cmos," in *Computer Design (ICCD), 2010 IEEE International Conference on*. IEEE, 2010, pp. 439–446.

[20] C. Nicopoulos, S. Srinivasan, A. Yanamandra, D. Park, V. Narayanan, C. Das, and M. J. Irwin, "On the effects of process variation in network-on-chip architectures," *IEEE Trans. Dependable Sec. Comput.*, vol. 7, no. 3, pp. 240–254, 2010.

[21] M. Azimi, D. Dai, A. Mejia, D. Park, R. Saharoy, and A. S. Vaidya, "Flexible and adaptive on-chip interconnect for tera-scale architectures," *Intel Technology Journal*, no. 4, pp. 62–77, 2009.

[22] M. Azimi, D. Dai, A. Kumar, and A. S. Vaidya, *On-chip Interconnect Trade-offs for Tera-scale Many-core Processors*. Designing Network On-Chip Architectures in the Nanoscale Era, Jose Flich and Davide Bertozzi, Eds., CRC Press, 2011.

[23] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2008, pp. 342–353.

[24] G. Dimitrakopoulos, E. Kalligeros, and K. Galanopoulos, "Merged switch allocation and traversal in network-on-chip switches," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2001–2012, 2013.

[25] J.-J. Lecler and P. Boucard, "Zero-latency network on chip (NoC)," US Patent 2011/0 085 550, 2011.

[26] D. Jayasimha, J. Chan, and J. Tomlison, "Use of common data format to facilitate link width conversion in a router with flexible link widths," US Patent 8 514 889, 2011.

[27] G. Michelogiannakis, N.Jiang, D.Becker, and W.J.Dally, "Packet chaining: Efficient single-cycle allocation for on-chip networks," in *Proc. Intern. Symp. on Microarchitecture*, 2011, pp. 83–94.

[28] H. Matsutani, M. Koibuchi, H. Amano, and T. Yoshinaga, "Prediction router: A low-latency on-chip router architecture with multiple predictors," *IEEE Transactions on Computers*, vol. 60, no. 6, pp. 783–799, 2011.

[29] Y.-Y. Chang, Y. S.-C. Huang, M. Poremba, V. Narayanan, Y. Xie, and C.-T. King, "Ts-router: On maximizing the quality-of-allocation in the on-chip network," in *Proceedings of the 2013*

IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), 2013, pp. 390–399.

[30] T. Karnik and et al., "Total power optimization by simultaneous dual-vt allocation and device sizing in high performance microprocessors," in *Proceedings of the 39th Annual Design Automation Conference*, ser. DAC '02, 2002, pp. 486–491.

[31] R. Manevich, L. Polishuk, I. Cidon, and A. Kolodny, "Designing single-cycle long links in hierarchical nocs," *Microprocessors and Microsystems*, vol. 38, no. 8, pp. 814 – 825, 2014.

[32] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, June 2006, pp. 187–198.

[33] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[34] P. Salihundam et al., "A 2Tb/s 6x4 Mesh Network with DVFS and 2.3Tb/s/W router in 45nm CMOS," in *VLSI Circuits*, 2010.

[35] S. Ma, N. Enright Jerger, and Z. Wang, "Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-Chip," in *Proc. of the Intern. Symp. on High Performance Computer Architecture*, Feb. 2012, pp. 467–478.

[36] J. Lee, C. Nicopoulos, H. G. Lee, and J. Kim, "TornadoNoC: A lightweight and scalable on-chip network architecture for the many-core era," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, Dec. 2013.

[37] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008, pp. 72–81.

Anastasios Psarras received the Diploma and master's degrees in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2012 and 2013, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include system-on-a-chip design, and in particular, on-chip interconnection networks.

Ioannis Seitanidis received the Diploma degree in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2013, where he is currently pursuing the Ph.D. degree in computer engineering.

His current research interests include computer architectures and on-chip interconnection networks.

Chrysostomos Nicopoulos received the B.S. and Ph.D. degrees in electrical engineering with a specialization in computer engineering from Pennsylvania State University, State College, PA, USA, in 2003 and 2007, respectively.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus. His current research interests include networks-on-chip, computer architecture, multi/many-core microprocessor and computer system design.

Giorgos Dimitrakopoulos received the B.S, MSc and Ph.D. degrees in Computer Engineering from University of Patras, Patras, Greece, in 2001, 2003 and 2007, respectively.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. He is interested in the design of digital integrated circuits and computer architecture, with emphasis in Network-on-Chip design and ultra-low power systems.