This is an early accepted version of the paper; the final version of this work is published in IEEE Transactions on Computers, DOI 10.1109/TC.2021.3068615. Copyright belongs to IEEE.

S-SMART++: a Low-Latency NoC Leveraging Speculative Bypass Requests

Iván Pérez, Enrique Vallejo and Ramón Beivide

Abstract—

Many-core processors demand scalable, efficient and low latency NoCs. Bypass routers are an affordable solution to attain low latency in relatively simple topologies like the mesh. SMART improves on traditional bypass routers implementing multi-hop bypass which reduces the importance of the distance between pairs of nodes. Nevertheless, the conservative buffer reallocation policy of SMART requires a large number of Virtual Channels (VCs) to offer high performance, penalizing its implementation cost. Besides, SMART zero-load latency values highly depend on HPC_{Max}, the maximum number of hops that can be jumped per cycle.

In this paper, we present Speculative-SMART++ (S-SMART++), with two mechanisms that significantly improve multi-hop bypass. First, zero-load latency is reduced by speculatively setting consecutive multi-hops. Second, the inefficient buffer reallocation policy of SMART is reduced by combining multi-packet buffers, *Non-Empty Buffer Bypass* and per-packet allocation.

These proposals are evaluated using functional simulation, with synthetic and real loads, and synthesis tools. S-SMART++ does not need VCs to obtain the performance of SMART with 8 VCs, reducing notably logic resources and dynamic power. Additionally, S-SMART++ reduces the base-latency of SMART by at least 29.2%, even when using the biggest HPC_{Max} possible.

Index Terms—SMART; SMART++; Speculative-SMART++; multi-hop bypass

1 Introduction

Low latency in NOCs for a wide range of traffic loads is critical for the performance of multiprocessors and other accelerators. Different approaches have been considered for this goal, including very large crossbars (such as [27], [31]), low-diameter topologies based on high-radix routers (such as [1], [3]) or aggressive lookahead routing, speculative stages and router bypass mechanisms (such as [17], [19], [21]). SMART [17], which belongs to the last group, is a very effective solution which implements multi-hop bypass, this is, it skips several intermediate transit routers in a single hop to dramatically reduce latency. SMART combines the simplicity and regularity of traditional 2D tiled designs with near-optimal latency (close to an ideal point-to-point interconnect) and very high throughput.

However, practical implementations of SMART result in overly large, power-hungry and slow router designs, for several reasons. First, the Virtual Channel (VC) reallocation scheme employed requires the corresponding buffer to be empty to reassign any VC buffer. This is often required in several contexts such as wormhole (WH) networks using fully-adaptive routing protocols [11], [23]. However, SMART neither employs WH nor is fully adaptive. Additionally, obtaining good performance using this reallocation scheme requires a large number of VCs, each of them holding a whole packet. This large number of VCs makes allocators more complex, which increases the critical path latency, and drastically increases router area and power consumption. Second, the buffers to bypass must be empty; otherwise the packet would not be forwarded. With a limited amount of VCs, this increases Head-of-Line Blocking (HoLB), reducing performance. Third, even though traffic is sent following Virtual Cut-Through (VCT) flow control, flit-by-flit arbitration collisions may make a packet spread through multiple routers, blocking the buffers in the intermediate routers. Finally, the setup of each consecutive multi-hop requires three clock cycles, significantly larger than simpler single-hop bypass mechanisms based on LookAheads (LAs); as evaluated in this work, SMART is not competitive in small networks or with small HPC_{Max} because of this per-multihop latency.

This work presents Speculative-SMART (S-SMART++), which extends SMART++, first introduced in [29]. SMART++ is an efficient multi-hop bypass mechanism that avoids the main limitations of SMART and allows for much simpler implementations. SMART++ combines SMART bypass [17], multi-packet buffers, *Non-Empty Buffer Bypass* (NEBB, [28]) and per-packet allocation using grant-hold circuits. SMART++ supports efficient configurations with a small amount of deeper buffers, rather than the large number of individual VCs required in SMART, which results in much better area, power and critical path delay. S-SMART++ additionally reduces per-multihop latency by

- *The authors are with the Department of Computer Science and Electronic Engineering, University of Cantabria, Cantabria, 39005 Santander, Spain.*
- *R. Beivide is also with the Department of Computer Sciences - Runtime Aware Architecture, Barcelona Supercomputing Center, Catalonia, 08034 Barcelona, Spain.*

Manuscript received August 15, 2020; This work was supported by the Spanish Ministry of Science, Innovation and Universities, FPI grant BES-2017-079971, the Spanish Ministry of Science, Innovation and Universities under contracts TIN2016-76635-C2-2-R (AEI/FEDER, UE) and TIC PID2019-105660RB-C22, and the European HiPEAC Network of Excellence. The Mont-Blanc project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671697. (Corresponding author: Iván Pérez)

speculatively configuring the subsequent multi-hop while data are traversing the previous one.

S-SMART++ is evaluated using the BST simulation framework [30], which combines the functional simulator Booksim [15], the full-system simulator gem5 [6] and HDL implementations based on OpenSMART [22]. S-SMART++ combines the resource efficiency of SMART++, which provides high performance using a single buffer of limited size per port and is both area- and power-efficient, with the zero-load latency reduction of its speculation mechanism, reducing the effect of HPC_{Max} in the latency.

Specifically, the main contributions of this paper are:

- It reviews SMART++, an efficient multi-hop bypass mechanism that outperforms the original SMART with much lower requirements on VCs, area and power.
- S-SMART++, a speculative mechanism that acquires the multi-hop bypass in advance to decrease base latency and reduce the latency sensitivity to HPC_{Max}.
- A performance evaluation by simulation, which proves that S-SMART++ without VCs has similar performance than SMART with VCs for single-packet buffers and the same buffer space, and that it reduces the zero-load latency of SMART significantly even halving HPC_{Max}.
- Resource utilization and power evaluations using HDL synthesis, showing the high cost of VCs, and the extensive and feasible variety of buffer configurations of S-SMART++.

Section 2 presents the required background. Section 3 describes S-SMART++. Section 4 evaluates the proposal. Finally, Section 5 compares to related work and Section 6 concludes the paper.

2 Background

2.1 NoC Router bypass

Router bypass [19], [20] is a mechanism that reduces latency by skipping some pipeline stages of the router. This type of NoC has additional communication signals denominated LookAheads. LookAheads contain the routing information of packets and are sent one cycle before the transmission of packet flits. With the routing information, the next router allocates the crossbar one cycle before the arrival of flits. If the allocation succeeds, the flit takes a bypass path to the crossbar, avoiding the allocation stages and buffer write, saving time and energy. Multiple LookAheads from different sources and local flits, may compete for the same output port in a router, so a new unit called LookAhead Conflict Check or LookAhead Arbiter is defined to arbitrate them in case of conflict.

Switch allocation in traditional LookAhead router bypass is done flit by flit. Therefore, part of a packet might bypass a router while the remaining is buffered. When multiple packets are allowed to be written in the same buffer, flits of different packets might interleave in the buffer, corrupting data. Requiring empty buffers to forward packets avoids this issue in a conservative way.

NEBB [28] is an alternative bypass policy that removes the empty buffer limitation, as its name implies, maximizing the utilization of the bypass. Different variants of NEBB are defined for different flow controls, allowing to bypass a buffer which is non-empty, but not advancing a packet to an output port. In general, they allow the bypass of a non-empty buffer for single-flit packets in any case, or when both the bypass and destination buffers have room for the whole packet and VCT is assumed.

Maximizing the bypass utilization reduces dynamic power consumption and the amount of VCs required to obtain the maximum throughput from the NoC.

2.2 SMART: multi-hop router bypass

SMART (Single-Cycle Multihop Asynchronous Repeated Traversal) [17] is a NoC router bypass that allows flits to cross multiple routers in a single cycle. In this design, LookAheads are called SMART-hop Setup Request (SSR). When flits are ready to be transmitted after winning Switch Allocation Local (SA-L), SSRs are broadcast to the next routers in the path. HPC_{Max} defines the maximum number of hops per cycle allowed, limited by the operation frequency of the NoC. Two variants are defined: SMART_1D only broadcasts SSRs in a row or column of the NoC mesh; SMART_2D employs additional lines to broadcast SSRs in both mesh dimensions, allowing for dimension change in a single multi-hop at the cost of much higher complexity.

SSRs request access to the bypass in each of the downstream routers, managed by a Switch Allocator Global (SA-G) function. In SA-G, SSRs from different sources and local buffered flits may conflict. If a conflict occurs, flits may suffer a *premature stop* and be buffered in an intermediate router of the desired multi-hop path. A single priority policy is enforced in all the network to guarantee correctness. In this work, local flits always have priority over bypass flits as it attains the best performance [17]. Figure 1 shows an example of bypass setup and flit transmission. In the first cycle, the green packet (first router) and the blue packet (last router) broadcast the SSRs in their route direction. The SSR of the green packet setups the bypass path of the second and third routers because there are no conflicts with other flits or SSRs. However, it loses against the local blue flit in the last router. In the second cycle, the green packet crosses the first router crossbar and the bypass paths of the second and third routers to get to the last router.

Routers in SMART follow VCT requirements to send a packet: the destination buffer needs to hold the complete packet, and all the flits of the packet are *sent* consecutively. However, the flits of the packet are not always *received* consecutively at the destination of the multi-hop. Indeed, global arbitration is performed per-flit, not per-packet, in all routers in the multi-hop. Thus, a new packet to be transmitted in one of the intermediate routers may receive higher priority and cause a *premature stop* of part of the flits of another packet. For this reason, flits from a packet may be received with *gaps*, and flits from different packets may be interleaved in the physical links between routers, similar to a WH network.

There are two alternatives for the bypass path in SMART NoCs: *buffer bypass* and *router bypass*. With buffer bypass [17], flits only bypass the input buffers in the input unit, but they pass through the crossbar. Buffer bypass is

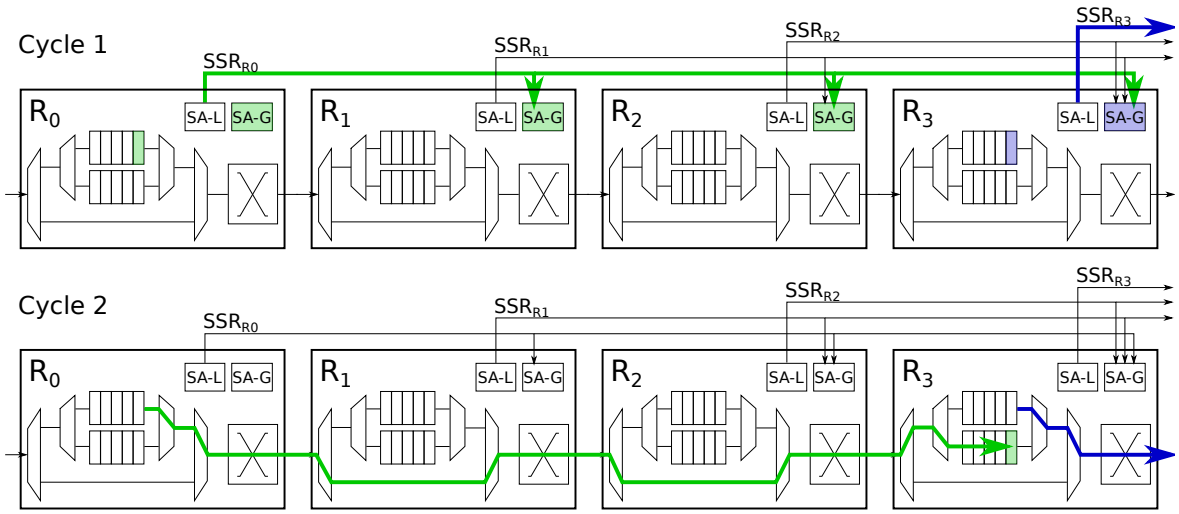


Fig. 1: SMART bypass setup and flit traversal overview with priority to local flits.

appropriate for SMART_2D and for ejection router bypass. Router bypass, depicted in Figure 1 and used in this paper, was introduced in OpenSMART [22]. In this case, flits take a dedicated path from the input port to the output port of the router following the same dimension. Router bypass is more suitable for SMART_1D because avoids conflicts between SSRs and local flits that share the same input port but request different output ports.

3 SPECULATIVE-SMART++

Speculative-SMART++ (S-SMART++) overcomes the main limitations of SMART described in Section 1 using two mechanisms. First, base latency is reduced by pre-allocating the bypass mechanisms based on speculative SSRs. This mechanism chains consecutive multi-hops requiring only a single cycle per multi-hop. An overview of this speculation mechanism is presented in Section 3.1, followed by the router architecture in Section 3.2, and a detailed analysis of its behavior in Section 3.3. Second, buffer management is simplified to support long multihops even with few buffers that may not be empty. This mechanism, originally introduced as SMART++ in [29], allows for a simpler router implementation and hence improves frequency, area and power. An overview is presented in Section 3.4 and implementation details in Section 3.5.

3.1 S-SMART overview

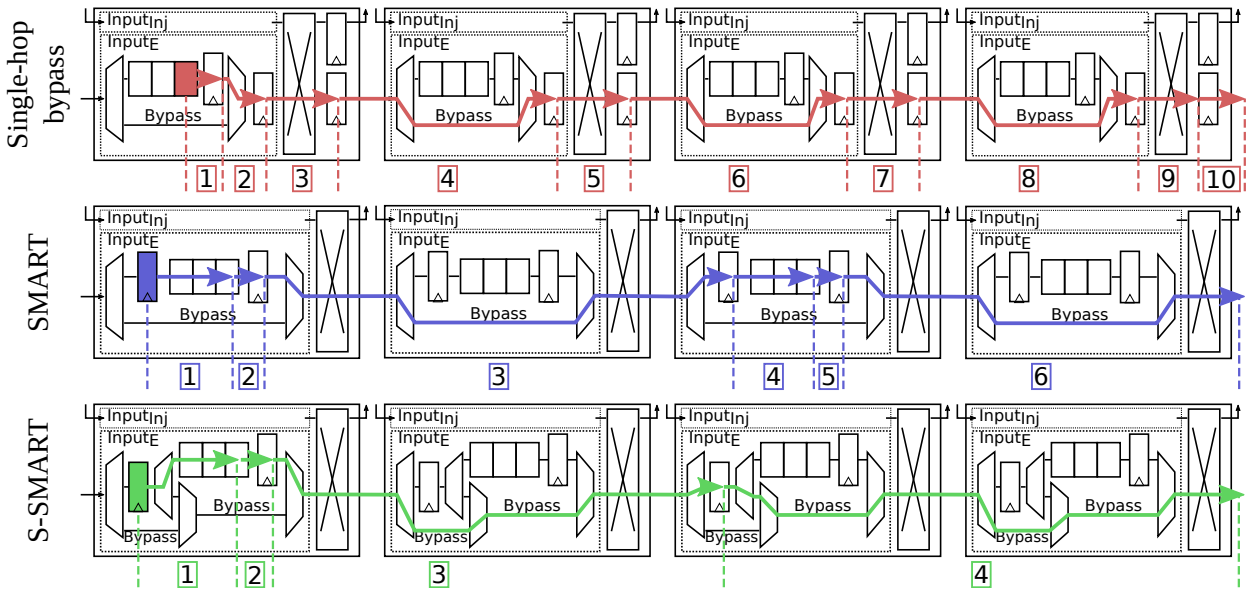
Single-hop bypass networks [18], [19], [20] preallocate bypass paths to skip the buffering and allocation stages. The idea of SMART of conforming multi-hop paths by preallocating bypass paths is very similar to them, but their principles are slightly different. SMART seeks to minimize latency by conforming the largest multi-hop paths, whereas single-hop bypass chains multiple hops in consecutive cycles. SMART is more effective in terms of latency, specially in large networks. However, single-hop bypass optimizes per-hop latency and allows to skip all the buffers in the whole route, optimizing dynamic power. In SMART this is not possible, as the packet is buffered after each multi-hop.

Speculative-SMART combines both approaches to support chaining several multi-hops in consecutive cycles, skipping the first and second stages in subsequent multi-hops. In S-SMART, SSRs are generated speculatively in the last router of each multi-hop, where the packet would be buffered. Therefore, while a packet is traversing the routers, a speculative SSR (spec-SSR) is requesting the subsequent bypass paths.

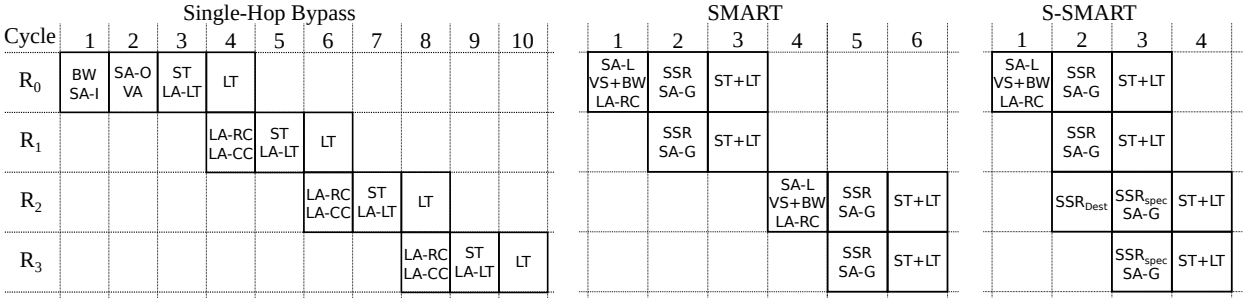
In SMART, global arbitration is speculative, since a packet may not reach the maximum desired hop length because of a conflict. However, after sending an SSR, SMART always sends data on the multi-hop. S-SMART exploits speculation even further, since the SSR itself is also speculative (spec-SSR), given that it is generated before even knowing if the associated packet will reach the router in time for the multi-hop. Therefore, a spec-SSR may not be followed by any data, when it conflicts in an intermediate router in the previous multi-hop.

Figure 2a presents a comparative example of a flit crossing the network when using single-hop bypass, SMART, and S-SMART, the latter two with $HPC_{Max} = 2$. Note that the behaviour of S-SMART with $HPC_{Max} = 1$ would be equivalent to single-hop bypass. Figure 2b depicts the pipeline stages of the router and their temporal behaviour. They are reviewed next.

- **Single-hop bypass**: we follow the router architecture in [18], similar to the SMART router architecture but with some differences. First SA-L is divided in two phases, Switch Allocation Input (SA-I) and Switch Allocation Output (SA-O), to distribute the delay in two pipeline stages. LA-CC (LookAhead Conflic Check) is equivalent to SA-G and arbitrates in case of conflict between LookAheads (equivalent to SSRs) and/or local flits. And third, Switch and Link Traversal (ST and LT) are separated into two pipeline stages, while in SMART they are merged.
  The packet in R_0 (in red) wins SA-I, SA-O and VA (VC Allocation) in the first 2 cycles. In the next two cycles it traverses the crossbar and the link, while the LookAhead travels through the link and acquires



(a) Flit forwarding in different bypass-router architectures. Boxed numbers represent the cycle of each step.



(b) Pipelines of different bypass-router architectures.

Fig. 2: Comparison of single-hop bypass routers, SMART and S-SMART ($HPC_{Max} = 2$). The boxes placed together the arrows indicate the cycle when the flit progress through the router paths.

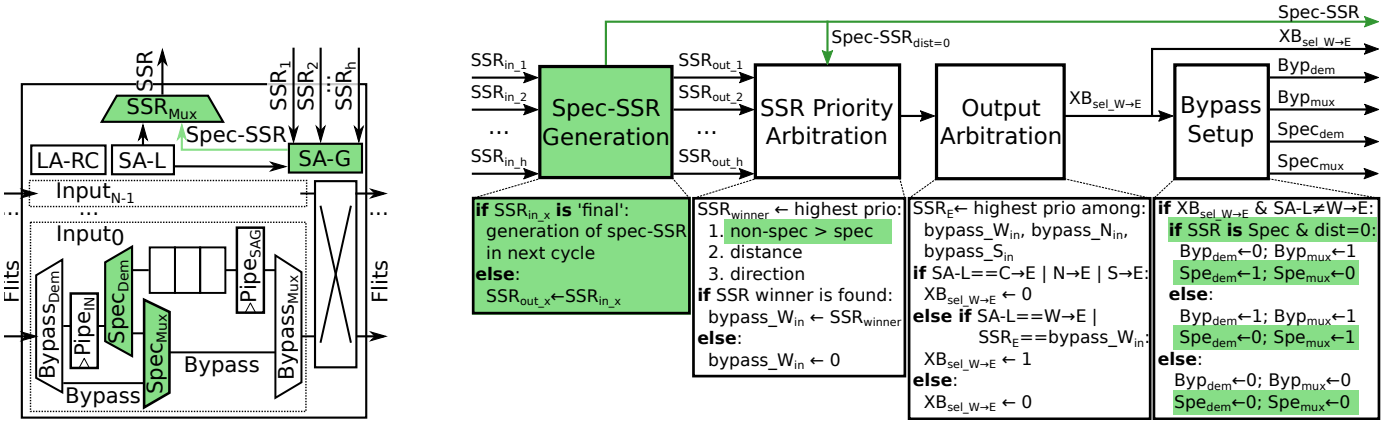
access to the crossbar and the bypass in LA-CC of R_1. In cycle 4, the LA succeeds in LA-CC, and originates the creation of another LA from R_1 to R_2. In this mechanism, the arrival of the packet to R_1 is guaranteed in cycle 5 so the second LA is not speculative. In the next two routers, the bypass and crossbar acquisitions and the packet traversal follow the same procedure. Overall, The packet spends 10 cycles to cross the four routers.

- **SMART**: In R_0, the packet (in blue) performs SA-L, VC Selection (VS, similar to VA), and LookAhead Routing Computation (LA-RC) in the first cycle. In the second cycle it propagates the SSR to R_1 ($HPC_{Max} - 1 = 2 - 1 = 1$) to acquire the bypass through SA-G and in the third cycle it performs the multi-hop to reach R_2. The SSR does not reach R_2, since this final router does not setup any bypass. In R_2, the packet repeats the same process. Overall, the packet spends 6 cycles to cross the routers.
- **S-SMART**: Like in SMART, the packet (in green) spends 3 cycles in R_0. However, in cycle 2 the SSR broadcast is extended to reach R_2. The extended SSR does not request SA-G in R_2, but produces the broadcast of a speculative SSR (spec-SSR) in cycle 3, which acquires the bypass of R_2 and R_3. Thus, in cycle 4 the packet takes the bypass path (instead of performing BW like in SMART). Overall, the packet requires 4 cycles to traverse the four routers.

3.2 Router architecture

S-SMART introduces modifications in three elements of the design of SMART: the generation of SSRs, the SSR priority scheme and the bypass control. Figure 3 shows the router architecture of S-SMART and the implementation of SA-G in the input and output ports. The highlighted elements are the changes from the SMART design.

In S-SMART the length of SSRs is extended by one unit, so the SSR reaches the final router of the multi-hop. SSRs carries the final destination of the packet to the determine the next multi-hop direction and length. The *'final'* router of a multi-hop is determined using the multi-hop length requested the SSR, information already carried in SMART. The multi-hop length is compared with the distance to the multi-hop requester, which is known from the ID of the SSR input port. The *'final'* router uses the final destination of the packet to generate the spec-SSR in the next cycle, instead of computing SA-G. This is depicted in Figure 3b, with the other modules of SA-G: input arbitration (SSR



(a) S-SMART router architecture.

(b) Implementation of SA-G. This allocator implements one *SSR Priority Arbitration* module per input and one *Output Arbitration* module per output; only the ones corresponding to W_{in} and E_{out} are depicted here. $XB_{sel_{W->E}}$ is the crossbar control signal that enables the path from West input to East output.

Fig. 3: Router Architecture and Switch Allocator Global implementation, with S-SMART additional elements highlighted in green. Byp_{Mux}, Byp_{Dem}, Spe_{Mux} and Spe_{Dem} are the control signals of $Bypass_{Mux}$, $Bypass_{Dem}$, $Spec_{Mux}$ and $Spec_{Dem}$, respectively.

Priority Arbitration), output arbitration, and bypass setup logic. Additionally, the router that generates a spec-SSR also propagates the request to its own SSR Priority arbitration logic in the SA-G unit ($Spec - SSR_{dist=0}$ in Figure 3b) to activate its bypass.

SSR priority scheme: The SSR priority scheme is modified to resolve conflicts between spec-SSRs and standard SA-G requests (SSRs or local flits). We give absolute priority to standard requests over spec-SSRs, to minimize unnecessary premature stops. The reason comes from the *local priority* policy used in SMART, i.e. priority to the SSR originated in the nearest router, which is the best policy evaluated in [17]. With this shortest-distance policy, packets may not complete their whole multi-hop. For this reason a spec-SSR may win SA-G in a router but leave the bypass path unused in the following cycle, because the associated packet was stopped in an intermediate router of the previous multi-hop. Giving low priority to spec-SSRs prevents other packets from stopping prematurely due to a speculative bypass acquisition that is not used.

Conflicts between spec-SSRs and standard requests can occur in the input phase of SA-G or in the SSR output phase. The first kind of conflict, in the input phase of SA-G, occurs when standard SSRs and/or spec-SSRs share the same input port. When the conflict is between a standard SSR and a spec-SSR, the spec-SSR is ignored as mentioned before. This requires one extra bit to identify spec-SSRs. When the conflict is between two or more spec-SSRs the arbitration follows the same policy used for standard SSRs, i.e. the SSR with the shortest distance has priority. The second kind of conflict, in the SSR output phase, can occur between a standard SSR (generated by a local flit) and a spec-SSR, or between two or more spec-SSRs. In the first case, the conflict is solved with a multiplexer (SSR_{Mux} in Figure 3a) at each SSR output port, which discards the spec-SSR. The second conflict occurs because multiple spec-SSRs for the same output port can be generated in a router from SSRs from different input directions at the same cycle. In such case, only one of them is chosen. We choose the one with the longest multi-hop to maximize the utilization of the bypass paths.

Bypass control: The standard bypass of SMART is implemented by a pair of demultiplexer ($Bypass_{Dem}$) and multiplexer ($Bypass_{Mux}$). S-SMART implements an additional path to bypass the input buffer from the input pipeline register ($Pipe_{In}$) when spec-SSRs win SA-G. This path is formed by a pair of demultiplexer ($Spec_{Dem}$) and multiplexer ($Spec_{Mux}$), depicted in Figure 3a. These pairs of muxes are controlled together. Overall, there are three paths. The standard bypass path from $Bypass_{Dem}$ to $Bypass_{Mux}$ is used when an SSR wins SA-G, except for speculative SSRs generated in the local router (generated distance 0). The second bypass path from $Pipe_{In}$ to $Bypass_{Mux}$ is used when a spec-SSR with distance equal to 0 wins SA-G. Finally, the traditional path from $Pipe_{In}$ to the buffer is used when an SSR does not win SA-G.

3.3 Speculative bypass walk-through

This section describes the process of using the bypass speculatively through an example. Figure 4 shows the state of the network during the four cycles that takes the highlighted packet to make two multi-hops. The process is divided into two phases. The first phase extends from cycles 1 to 3, when the packet prepares and performs the multi-hop bypass like in the original design of SMART. The second phase shows the bypass via speculative arbitration, in cycles 3 and 4.

In cycle 1, the packet in R_0 requests a local output port in SA-L while the route for the next multi-hop is computed in LA-RC. The packet wins SA-L (there are no competitors) and moves to the next pipeline stage. In cycle 2, the SSR is propagated to the next two routers ($HPC_{Max} = 2$ in this example) to prepare the multi-hop. The SSR received in R_1 performs SA-G, while R_2 uses the destination information to compute LA-RC and generate the spec-SSR in the next

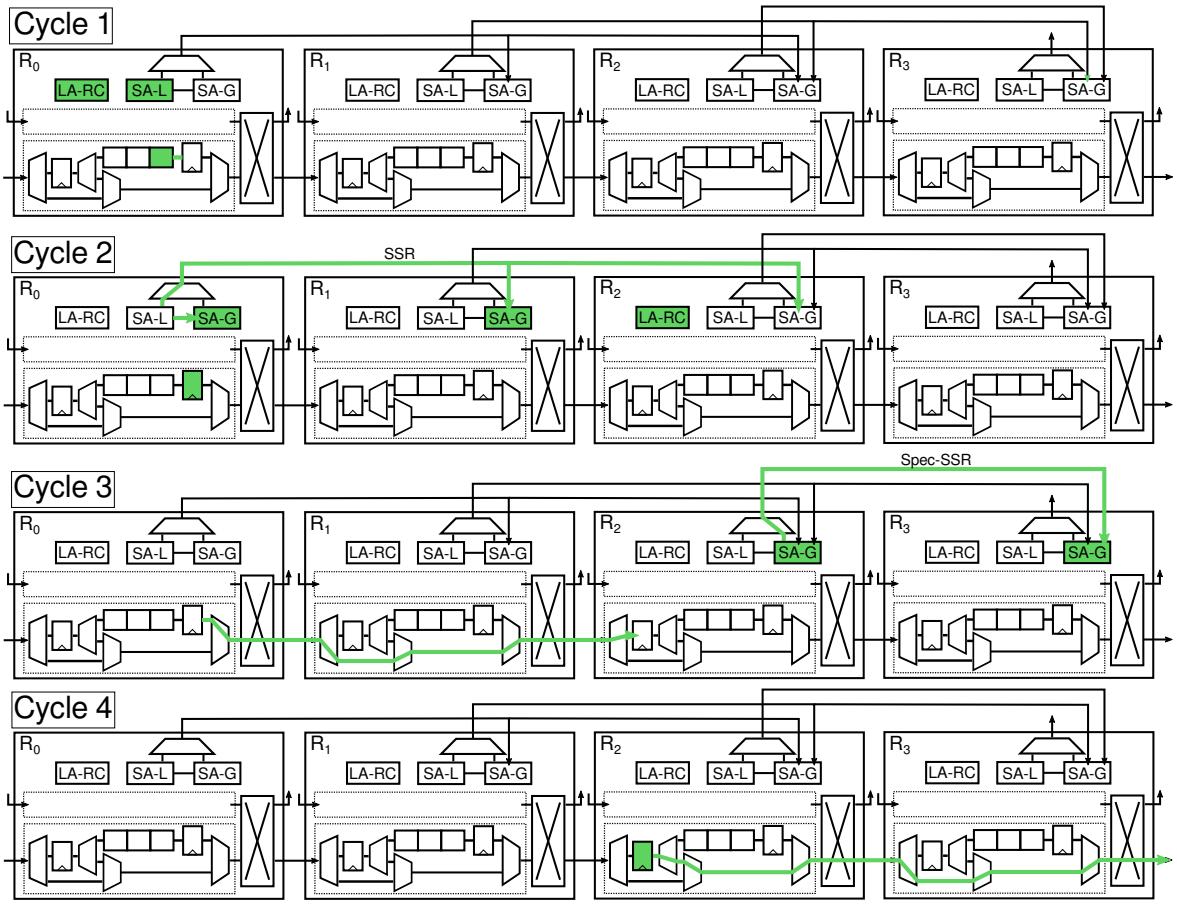


Fig. 4: Example of speculative SA-G arbitration in S-SMART. HPC_{Max} is 2.

cycle, because it is the 'final' router (multi-hop length equal to the distance between routers). The packet wins SA-G in R_0 and R_1, advancing to the next stage and preparing the control signals of the crossbars and paths for the next cycle. In cycle 3, the packet traverses R_0 and R_1, reaching R_2, where it is saved in the input port latch ($Pipe_{In}$ in Figure 3a). Simultaneously, R_2 generates the spec-SSR from the destination information saved in the previous cycle, and sends it to its local SA-G and R_3 (and R_4 in case it exists). Both spec-SSRs win SA-G because there are no competing requests. Finally, in cycle 4, the packet in $Pipe_{In}$ of R_2 travels through $Spec_{Dem}$ and $Spec_{Mux}$ towards the bypass path of R_2 and R_3.

3.4 SMART++ overview

Speculative-SMART++ leverages the SMART++ buffer management initially introduced in [29]. This section presents an overview of the mechanism.

The original SMART implementation requires empty buffers to allow bypass, both in the bypassed routers and the final router that holds the packet after the multi-hop. Such implementation requires a significant number of independent buffers to be effective. This large amount of buffers increases both the area and power of the router and the complexity of the allocators, which eventually reduces maximum frequency.

Instead of using many short buffers, SMART++ relies on few deep buffers, ideally just one buffer holding multiple packets. Such organization is more efficient, reducing both area and power consumption, even though the overall buffering remains the same. Additionally, it simplifies allocation stages, increasing maximum frequency. However, finding a completely empty buffer to allow bypass with this organization will be more infrequent. Instead, SMART++ relies on *Non-Empty Buffer Bypass* (NEBB, [28]), which allows to bypass routers when an input buffer has room for at least one packet but it is not necessarily empty. Virtual Cut-Through is required for this bypass when using multi-flit packets, but this is not a limitation since the original SMART implementation already requires several buffers holding a complete packet.

Figure 5 compares the length of the multihop when using a single buffer per router input. SMART requires empty buffers and does not exploit multipacket buffers, so it prematurely stops before the first router whose buffer is empty. SMART++ does not restrict the multihop, as long as all the router buffers in the path may hold one additional packet. This limitation is mitigated in SMART by implementing many buffers, and in such case SMART++ does not reduce cycle count since long multihops will likely occur in both implementations. However, SMART++ simplifies the design and improves area, power and frequency by relying on one or few buffers.

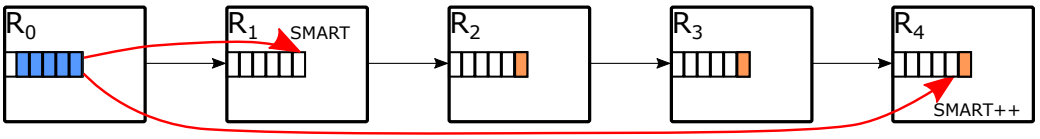


Fig. 5: Stop router of each mechanism in SMART and SMART++. R_4 is the destination of the blue packet in R_0 and routers only have one buffer.

3.5 SMART++ implementation details

This section summarizes the implementation details of the main components of SMART++. Low-level implementation details of SMART++ are presented in [29].

Multi-Packet Buffers (MPB): SMART requires buffers sized for the largest packet in the network, since it implements VCT flow control, but it only holds a single packet due to its VC reallocation policy. SMART++ holds multiple consecutive packets in router buffers and exploits buffers larger than a single packet size. Previous proposals for NoCs rely on similar approaches [7], [10], [24], [32], [33].

The use of multi-packet buffers allows to employ a lower number of VCs with deeper buffers, leading to simpler memory organizations that exchange width (#VCs) by length (deeper FIFOs). Such VC reduction simplifies allocation and reduces overall chip area even though the total storage remains the same. Additionally, combining multiple packets in the same buffer increases its efficiency, particularly with different-size packets (bimodal traffic), which often occurs in NoCs.

Packet-by-packet arbitration (PPA): SMART++ implements packet-by-packet arbitration using a grant-hold circuit [9] coupled to the round-robin arbitration stages (SA-G and SA-L). Grant-hold circuits hold the arbiter outcome for a certain amount of time. When a multi-flit packet header wins arbitration, SMART++ logic locks the arbiter to the winning packet. However, winning SA-G does not guarantee that a flit will be transferred in the following cycle: the flit could suffer a premature stop in an upstream router in the multi-hop. To cover this case, SMART++ releases the grant in two cases: when the packet tail is received, or when a head flit is not received. Grant holding is not required for single-flit packets.

Effectively, this makes SMART++ behave exactly as VCT, receiving all packets without holes from upstream channel or flit interleaving. Only packet headers generate SSRs or spec-SSRs. When an SSR is received in an intermediate router while a packet is being bypassed, it loses arbitration and the data is stored in the router buffers regardless of its priority. This behavior does not conflict with the *single priority enforced in the network* requirement of SMART because it does not introduce false positives (flit received when it is not expected), only premature stops. Additionally, these premature stops do not reduce performance: they always occur because other packet is actually being transferred on the desired output[1].

Non-Empty Buffer Bypass (NEBB): SMART requires an empty buffer in all of the routers to be bypassed. Such policy is forced by its conservative VC reallocation scheme (if no free buffer exists, packet is not sent to the bypass router in the first place), but is also overly conservative, and reduces performance, particularly with few VCss.

SMART++ employs NEBB [28] to bypass a buffer even when it is not empty. This only requires an additional buffer size check before putting the request in SA-G. NEBB mitigates the Head-of-Line Blocking issues caused by the use of few buffers, since it may bypass a buffer while the packet in the header waits for a dimension turn.

1. There are no cascading invalidations, as occurs with SSRs using Prio=bypass [17].

TABLE 1: Network simulation parameters.

Parameter	BookSim	gem5	BSV
Mesh size	4x4, 8x8 & 16x16	4x4 & 8x8	4x4
Bypass mechanism	SMART and S-SMART++		
Bypass type	buffer bypass		router bypass
Router size	5 ports		
Backpressure	Credits		
SMART VCs	8	12	1, 2, 4 & 8
S-SMART++ VCs	1	3	1, 2, 4 & 8
SMART buf. size	1 packet		
S-SMART++ buf. size (packets)	8	4	1, 2, 4, 8
Packet size (flits)	1 & 5		1
Routing	DOR XY		
VC selection policy	Shortest queue	Shortest queue	First available VC
SSR policy	Priority to local flits		
HPC_{Max}	1, 2, 3, 4, 7 & 15	3	
Flit size	128 bits		32 bits

4 Evaluation

This section evaluates S-SMART++. Section 4.1 describes the simulation infrastructure; Section 4.2 presents cycle-accurate performance results with synthetic traffic and Full-System (FS) simulations; Section 4.3 shows synthesis estimations of power, resource utilization and maximum frequency.

4.1 Simulation Infrastructure

We employ the BST framework [30], which combines three development platforms: BookSim [15], gem5 [6] and OpenSMART [22]. BookSim is an open-source functional simulator written in C++. BST BookSim implements cycle-accurate models of SMART and SMART++, and we have extended it with a detailed model of S-SMART++. These implementations supports variable size packets. We use the same models for the FS simulations by integrating BookSim in gem5 using the API of BST.

TABLE 2: gem5 configuration parameters

Parameter	Value
CPU model	16/64x ARMv8 Out-of-Order (DerivO3CPU) @2GHz
Memory model	Ruby @2GHz
Coherence protocol	MESI with two levels of cache (MESI_Two_Levels)
Cache line size	64 Bytes
L1-I cache	private, 32KB, associativity 2
L1-D cache	private, 64KB, associativity 2
L2 cache	shared and distributed among cores, 16/64x 256KB, associativity 8
Memory controllers	8/16x DDR3-1600 11-11-11 (Location: first and last mesh rows)

We have implemented S-SMART++ in Bluespec System Verilog (BSV) from OpenSMART. Like OpenSMART, this model is limited to single-flit packets and works with credits. The implementation uses *router bypass* instead of *buffer bypass*, despite the fact that *buffer bypass* is preferable for S-SMART++. The reason is that packets can change the traveling dimension when taking the bypass speculatively. *Buffer bypass* requires extra logic and paths to forward the packets in any direction instead of allocating the crossbar of the switch. This produces conservative power, area, and frequency results for S-SMART++. The BSV implementation is also used to validate the latency and throughput results of the BookSim models, through BSV functional simulations. The BSV compiler is used to generate Verilog code that is synthesized with Quartus Prime 18.1 Lite Edition to measure power, area and maximum frequency on an Arria II EP2AGX45DF29I5 FPGA.

We employ three types of simulations: functional with synthetic traffic, full-system simulations and validations with the BSV simulator. Most of the performance simulations in BookSim evaluate 8×8 meshes with SMART_1D and $HPC_{Max} = 7$. Some experiments also evaluate 8×8 SMART_2D meshes. Validation simulations are evaluated in 4×4 meshes with $HPC_{Max} = 3$ due to the large requirements of the BSV compiler and the large number of simulation points obtained. In all cases local flits have priority over bypass.

Synthetic traffic simulations evaluate multiple traffic patterns [9]: random uniform, bit-complement, bit-reversal, transpose, tornado and different configurations for hotspot traffic (with traffic distributed evenly among 4 hotspots at the corners or at the center of the mesh; with or without background uniform traffic). We evaluate single-flit and 5-flit packets, which are equivalent to control and data packets. Full-system simulations evaluate SMART and S-SMART++ under real workloads in gem5. We simulate the execution of PARSEC [5] benchmarks[2] under Linux 4.15.0 in an ARMv8 processor with 16 or 64 cores operating at 2 GHz. Simulations employ the detailed Out-of-Order processor (O3) and interconnection (Ruby) models in gem5. The cache coherence messages are distributed in 3 Virtual Networks (VNs) to break cyclic protocol dependencies. The 16-core model employs a a 4×4 mesh with $HPC_{Max} = 3$ and runs the complete Region of Interest of each benchmark with the simsmall input sets. The 64-core model employs an 8×8 meshes with $HPC_{Max} = 7$ and runs the simlarge input sets, but only for 500 million cycles because of its simulation time. Tables 1 and 2 gather the most relevant network and gem5 simulation parameters, respectively.

2. Raytrace is missing because of incompatibilities found with our simulation toolset.

4.2 Cycle-level Performance Results

This section evaluates the performance of S-SMART++ based on the packet latency in terms of cycles, abstracting differences in the maximum operation frequency of the NoCs.

4.2.1 Bypass mechanisms comparison

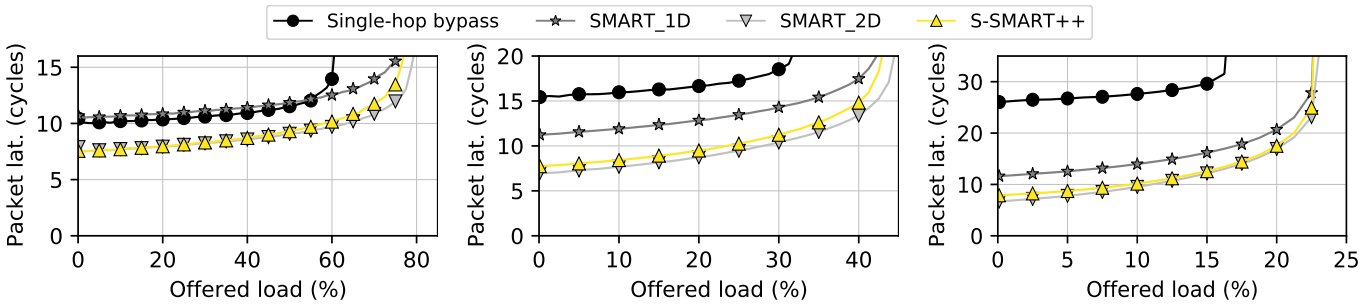
This section compares single-hop bypass (based on NEBB-Hybrid [28]), SMART_1D and SMART_2D with S-SMART++. Figure 6 shows the packet latency of the mechanisms in 4x4, 8x8 and 16x16 meshes with HPC_{Max} equal to 3, 7 and 15, respectively. These values cover a dimension of the meshes in one multi-hop. Injected traffic follows a random-uniform distribution and packets have one flit. NEBB-Hybrid and S-SMART++, which allow multi-packet buffering, have a single buffer of 8 slots. SMART_1D and SMART_2D have 8 VCs of 1 slot each.

The results show that single-hop bypass is competitive in small networks because of its shorter per-hop latency. However, its performance quickly degrades with the network size because the average hop count grows. In contrast, the zero-load latency with multi-hop bypass is almost constant with the network size, given that the effective number of hops is practically constant. S-SMART++ outperforms SMART_1D and almost reaches the performance of SMART_2D. The gain over SMART_1D is practically constant with the size of the meshes due to the values of HPC_{Max}. For example the base latency reduction in the 4x4 mesh is 29.2% while in the 16x16 mesh 32.1%. In all cases the base latency remains almost constant since HPC_{Max} always covers a full dimension, whereas throughput halves when the mesh size duplicates due to halving the ratio between the bisection bandwidth and the number of nodes.

4.2.2 S-SMART++ with different traffic patterns

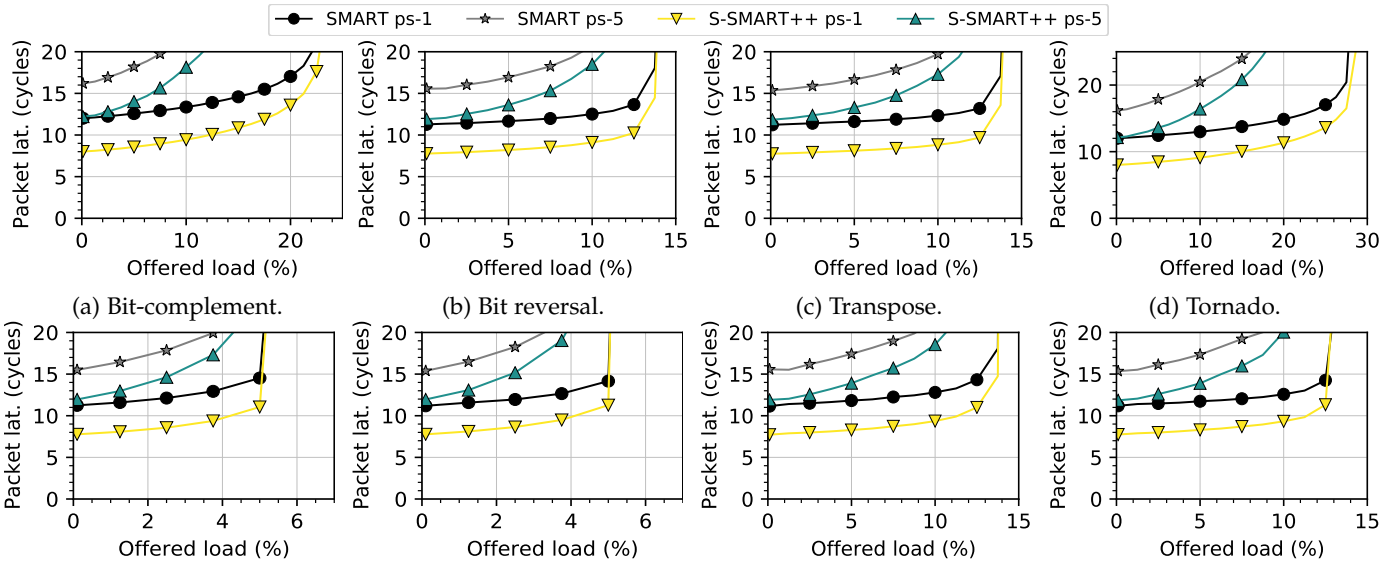
This section compares SMART and S-SMART++ for additional synthetic traffic patterns and packet size (ps) of 1 or 5 flits in an 8x8 mesh. Figure 7 depicts the results for bit-complement, bit-reversal, transpose and tornado traffic patterns in the upper row, and different hotspot configurations in the lower row: with the four hotspots in the corners or the center of the mesh, and with all the traffic to the hotspots or 40% traffic to the hotpots and the remaining 60% background traffic following a uniform distribution.

The results for all these different patterns are similar to the random-uniform traffic in Section 4.2.1. S-SMART++ has lower latency than SMART in every case, with similar throughput in spite of not using VCs. Besides, using multi-flit packets does not have any effect other than increasing the latency due the flit serialization of packets.



(a) 4x4 mesh, $HPC_{Max} = 3$ (b) 8x8 mesh, $HPC_{Max} = 7$ (c) 16x16 mesh, $HPC_{Max} = 15$

Fig. 6: Latency of single-hop bypass, SMART_1D, SMART_2D and S-SMART++ for different mesh sizes.



(a) Bit-complement. (b) Bit reversal. (c) Transpose. (d) Tornado.

(e) Hotspot (corners). (f) Hotspot (center). (g) Hotspot (corners) + UN. (h) Hotspot (center) + UN.

Fig. 7: Latency for various traffic patterns in an 8x8 mesh with $HPC_{Max} = 7$ and packet sizes of 1 (*ps-1*) and 5 flits (*ps-5*).

4.2.3 HPC_{Max} and scalability analysis

This section studies the impact of HPC_{Max} on the latency of SMART and S-SMART++ for different network sizes. We model 8x8, 16x16 and 32x32 meshes with different values of HPC_{Max}, from 2 to 15. Figure 8 shows the packet latency for each configuration. We focus on single-flit packets to simplify the analysis, given that the trend is similar with other packet sizes. Note that in the largest configuration the maximum multihop length does not reach the mesh side, taking into account the impact that a large multihop length would have on the delay of the multihop and the number of inputs of the SA-G unit.

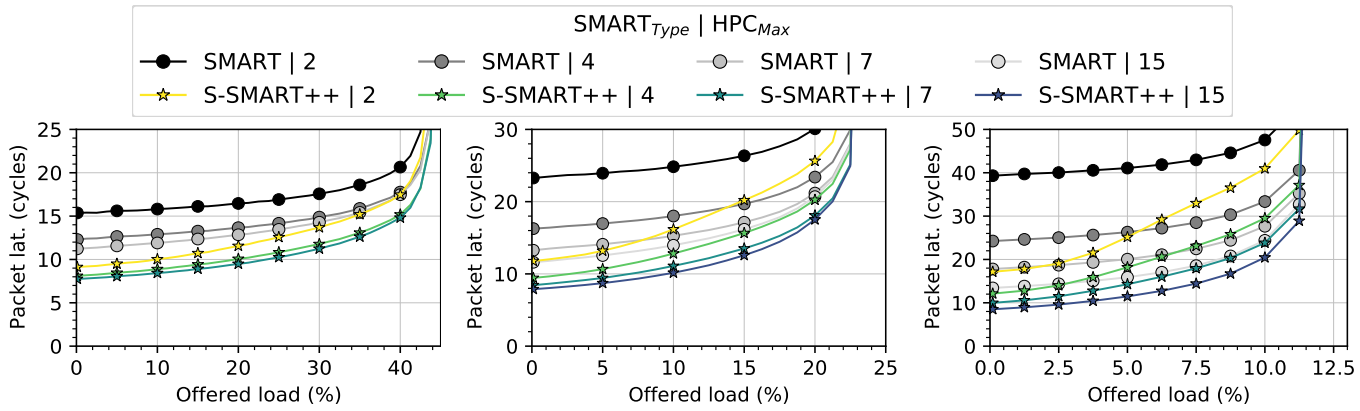
As expected, latency improves with larger HPC_{Max} in all cases. For a given HPC_{Max} value, S-SMART++ always outperforms SMART. Additionally, S-SMART++ outperforms SMART even with lower HPC_{Max} thanks to its speculation mechanism. For example, S-SMART++ with a moderate value of $HPC_{Max} = 4$ obtains lower zero-load latency than SMART with $HPC_{Max} = 15$ in all configurations. Finally, the influence of the multihop length on the packet latency is also reduced, specially at low offered loads. In the 8x8 mesh, between $HPC_{Max} = 2$ and $HPC_{Max} = 7$ the reduction of zero-load latency is 4.14 cycles in SMART, and only 1.38 in S-SMART++; in the 32x32 mesh, these differences are 21.53 and 7.17 respectively.

Considering the scalability of the network, the evolution of the latency is also improved with S-SMART++. When upscaling the network from 8x8 to 32x32 with a fixed $HPC_{Max} = 7$, SMART base latency increases by 6.57 cycles (a 58.5%), whereas S-SMART++ only increases by 2.24 cycles (a 28.9%). This difference is even more exacerbated for lower HPC_{Max} values. Maximum throughput decreases with the network size for all mechanisms, since it depends on the network Bisection Bandwidth (BBW); applying S-SMART++ to alternative topologies with higher BBW such as torus, while feasible, is out of the scope of this paper.

4.2.4 Evaluation with real traffic

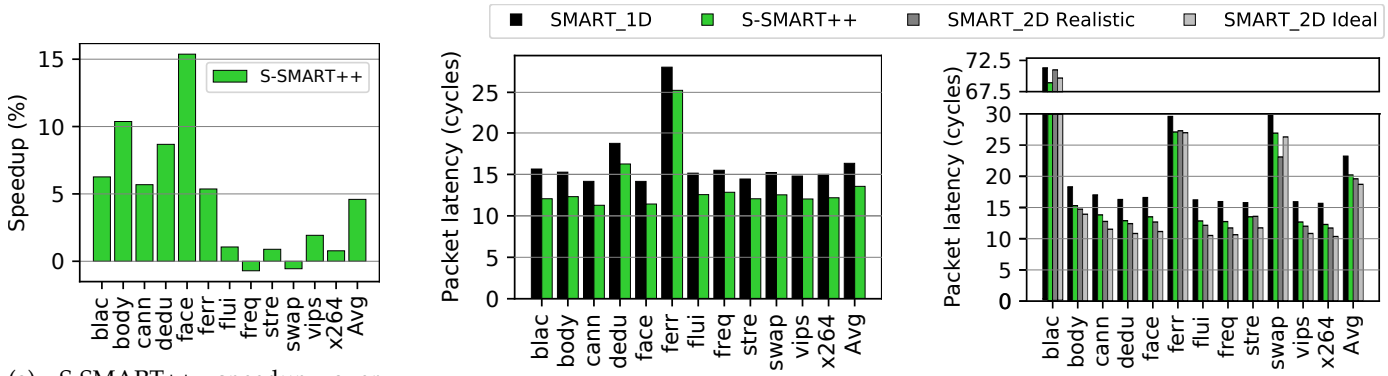
This section explores performance using full-system simulation running the PARSEC benchmarks.

Figure 9a presents speedup results of S-SMART++ over SMART with a 16-core system. The average speedup of S-SMART++ is 4.59%. Half of the applications present similar performance in both models, considering that the distribution of micro-architectural and operating system events along time introduces some performance variabil-



(a) 8x8 mesh (b) 16x16 mesh (c) 32x32 mesh

Fig. 8: Packet latency varying HPC_{Max} and network size.



(a) S-SMART++ speedup over SMART with 16 cores running the complete ROI.

(b) Average packet latency with 16 cores running the complete ROI.

(c) Average packet latency with 64 cores, 500M cycles.

Fig. 9: S-SMART++ performance on full-system simulations.

ity. The other applications present notable speedups for S-SMART++, with a maximum of 15.4%.

Figure 9b presents average packet latency results for the same executions. We measured an average network load of only 4.7%, so average latency results are very close to the base latency in most cases, with additional constant delays that correspond to the injection and ejection from the NoC in the memory sub-system. The speculative mechanism of S-SMART++ systematically improves latency in all cases, with an average reduction of 2.77 cycles over SMART, or a 17.4%.

Figure 9c presents latency with 64 cores in an 8x8 mesh, but only for the initial 500M cycles of the ROI. In this case the initialization is more relevant, and some applications (especially Blackscholes) present larger latency than when observing the full ROI. However, the systematic improvement of S-SMART++ observed with 16 cores is preserved, with an average reduction of 3.01 cycles. This graph also includes results for two variants of SMART_2D, one of them with a very costly but *realistic* HPC_{Max} value ($HPC_{Max} = 7$) and an *ideal* implementation that reaches the destination in a single multihop ($HPC_{Max} = 15$). S-SMART++ is very close to the realistic implementation of SMART_2D, and only 1.50 cycles higher than the ideal model on average.

4.3 Synthesis results

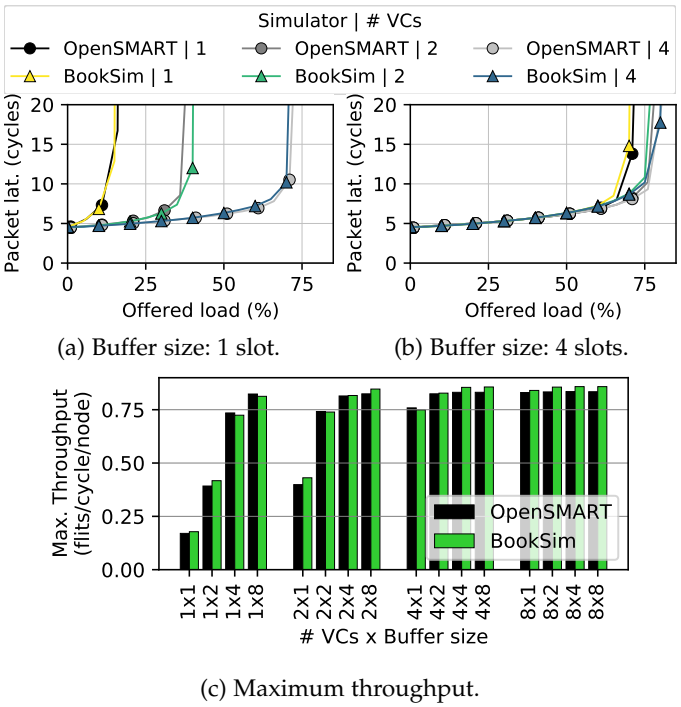
This section evaluates the S-SMART++ model implemented in OpenSMART with the Bluespec System Verilog tools and Quartus.

4.3.1 *Model Validation*

This section validates the models implemented in Booksim and BSV by comparing their results. The network evaluated is a 4×4 mesh with $HPC_{Max} = 3$, single-flit packets and random uniform traffic. Figure 10 shows the results of S-SMART++ in both platforms. The base latency of both implementations is the same until reaching saturation, where most of the differences are negligible. The highest relative error is 9.77% when using 2 VCs of 1 slot. From these results we consider that the functional models implemented in BookSim, cycle accurately simulate the router architecture and the pipeline, according to the HDL implementation.

4.3.2 *Resource Analysis*

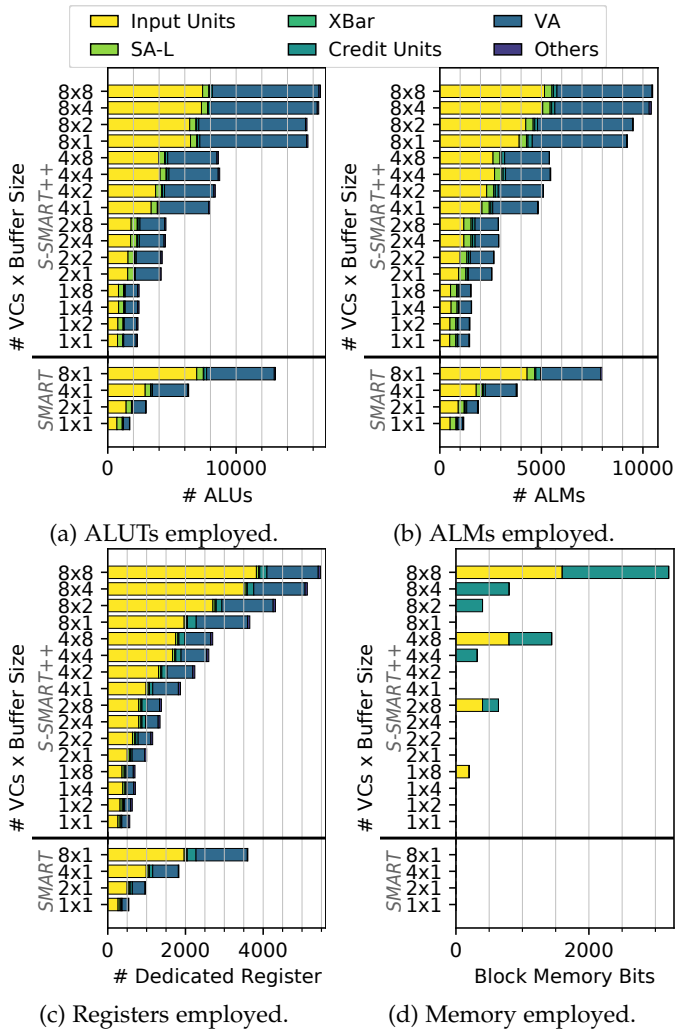
This section analyzes the resource requirements of a single SMART or S-SMART++ router, for different values of buffer count and depth (note SMART is limited to only 1 packet) using packets of 1 flit. Note that S-SMART++ targets designs with few deep buffers (1x8 instead of 8x1 in SMART) but configurations with multiple buffers are also considered.



(a) Buffer size: 1 slot. (b) Buffer size: 4 slots.

(c) Maximum throughput.

Fig. 10: Comparison of packet latency and throughput of the S-SMART++ models implemented in BSV and BookSim.



(a) ALUTs employed. (b) ALMs employed.

(c) Registers employed. (d) Memory employed.

Fig. 11: FPGA resources employed by SMART and S-SMART++.

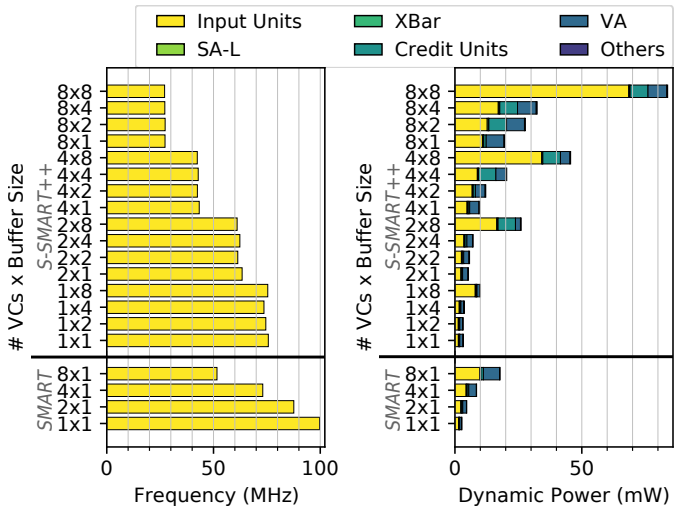
Figure 11 shows FPGA resources used by the synthesized routers, represented by the number of Adaptive Look-Up Tables (ALUTs), Adaptive Logic Modules (ALMs), dedicated registers and internal block memory bits. First, the results show the high impact of VCs on resource demands as they are part of the input units, credit units (credit handling logic) and VA. When duplicating the number of VCs, the number of resources is almost doubled. For example, the configuration of SMART with 2 VCs of 1 slot increases the number of ALUTs by 74.3%, ALMs by 62.1% and registers by 80,9% with respect to 1 VC of 1 slot. By contrast, when increasing buffer depth, the resource utilization grows at a much slower rate. For example, in S-SMART++ with 1 VC of 8 slots the number of ALUTs increases by 5.1%, ALMS by 4.6% and registers by 25.9% compared to 1 VC of 1 slot. In some configurations of S-SMART++, like 1x8 slots, the synthesis employs block memory (internal FPGA RAM) to build the buffers of the input and/or credit unit because the FPGA employed does not have enough dedicated registers. This causes a significant power increment for this configuration as shown in Section 4.3.3.

The overhead of S-SMART++ over SMART is similar between common configurations. For example with 1 VC of 1 slot, S-SMART++ uses 31.0% more ALUTs, 24.9% ALMs and 1.3% registers than SMART. With 4 VCs of 1 slot, these values are 26.92%, 26.18% and 0.06%, respectively. The logic increase is localized in the VA of OpenSMART, which integrates a large part of SA-G.

However, since S-SMART++ targets few deep buffers, effective configurations are more efficient than in SMART. For example, S-SMART++ with 1 VC of 8 slots employs 80.88% less ALUTs, 82.06% less ALMs and 81.71% less registers than SMART with 8 VCs of 1 slot. Finally, Figure 11 does not compare to single-cycle routers. SMART has been reported [22] to increase area by only 15% with respect to traditional single-cycle routers with an ordinary VC allocation scheme and multiple VCs. According to the previous comparison, as S-SMART++ does not require the use of VCs, it is expected to entail about 78% less area than a traditional single-cycle router.

4.3.3 *Timing and Power Analysis*

Figure 12 depicts the maximum operation frequency and the dynamic power consumption for multiple SMART and S-SMART++ router configurations. Again, note that S-SMART++ targets deep buffer arrangements such as 1x8, compared to 8x1 in SMART. To obtain dynamic power results, we feed the power analysis tool with VCD (value change dump) files generated from ModelSim functional simulations with a clock frequency of 25 MHz, which is under the minimum operation frequency of the configurations depicted. The results reveal that the number of VCs is a critical design factor. Focusing on SMART, doubling the number of VCs reduces frequency by between 12.04 and 21.44 MHz in each step. The overhead of S-SMART++ reduces the maximum frequency of SMART by 23.98 to 29.76



(a) Max. operation frequency. (b) Dynamic power.

Fig. 12: FPGA frequency and dynamic power results of SMART and S-SMART++.

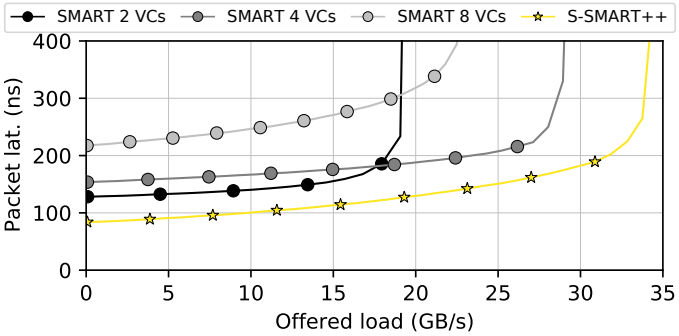


Fig. 13: Frequency-scaled latency of SMART and S-SMART++ for different number of VCs (1 in S-SMART++).

MHz for equivalent configurations. However, increasing the buffer depth has a negligible impact on frequency, and S-SMART 1x8 obtains a frequency 46.1% faster than SMART 8x1.

Dynamic power, depicted in Figure 12b, almost doubles when duplicating the number of VCs. For example, SMART with 2 VCs of 1 slot multiplies by 1.73 the power of 1 VC of 1 slot. Moreover, increasing the buffer depth in S-SMART++ has a negligible impact on frequency and moderately increases dynamic power. Section 4.3.2 mentions how the abrupt increment in dynamic power for some configurations (1x8, 2x8, 4x8, 8x2, 8x4 and 8x8) is caused by the use of memory instead of dedicated registers to allocate part of the buffers of the input unit. Despite this, S-SMART++ 1x8 reduced dynamic power by 44.1% with respect to SMART 8x1. Again, with respect to single-cycle routers, SMART has been reported [22] to increase dynamic power by only 3%. Thus, S-SMART++ should reduce dynamic power by more than 40% in respect to the traditional solution as it does not require VCs to exploit the network bandwidth.

4.3.4 *Scaled performance results*

Section 4.2 presents cycle-accurate performance results. However, Figure 12a shows that frequency in S-SMART++ may be significantly higher when not using VCs, further improving performance with respect to SMART, which requires multiple VCs. Frequency is typically determined by the allocation stages of the router. However, when a large HPC_{Max} value is considered, the total switch and link traversal (ST+LT) delay increases, and it might lower the router frequency. In such case, frequency in SMART and S-SMART++ would be similar, and their performance would be proportional to the figures in Section 4.2.

However, for moderate HPC_{Max} and in FPGA evaluations this may not occur because propagation delay is significantly lower than logic delay. In this case, performance will be determined by the maximum frequency in Figure 12a.

Figure 13 presents frequency-scaled latency results of SMART and S-SMART++ with single-flit packets and random uniform traffic. The figure shows SMART for different VC configurations (2, 4 and 8 VCs) given that more VCs increase throughput but reduces the maximum frequency. For S-SMART++ we only show a configuration with a single VC of 8 slots, equivalent in space and throughput than SMART with 8 VCs of 1 slot, because the frequency variations with the buffer depth are negligible.

From the results, it is clear that S-SMART++ outperforms SMART with lower costs. In terms of latency, S-SMART++ reduces the zero-load latency of SMART with 2 VCs by 45.3%. In terms of throughput, S-SMART++ increases the maximum throughput of SMART with 4 VCs by 13.9%.

5 Related Work

Sections 1 and 2 have already discussed several alternatives to reduce latency on NoCs, including SMART details. OpenSMART [22] is a NoC generator that provides verified RTL of SMART. We have extended it to support SMART++ and S-SMART++.

An SSR network that replaces SSR broadcast wires and complex allocators is proposed in [8] to reduce wire and energy overheads. However, this solution adds an extra pipeline stage to process SSRs, increasing the latency of each multi-hop by one cycle. SHARP (Smart Hop Arbitration Request Propagation) [2] is an alternative solution that does not add an extra pipeline stage and eliminates the quadratic SSR arbitration and false negative SA-G allocations by only propagating winning SSRs from the previous hop routers. However, like occurs with S-SMART++, SHARP increases the critical path delay, reducing HPC_{Max}. OpenSMART already implements a similar mechanism based on propagating only winning SSRs, however limited to one dimension. An adaptation of SHARP to S-SMART++ might allow generating spec-SSRs at intermediate routers when a premature stop occurs in a multi-hop, as it stops the propagations of SSRs at the first SA-G that is not granted.

WiSMART (wireless-enabled SMART [12]) is a hybrid of SMART and wireless NoC (WiNoC). This combination allows to operate at high frequencies independently of HPC_{Max}, using wireless communication for long distances. A comparison with S-SMART++ is left for future work, as well as a possible adaptation.

Task mapping techniques to reduce conflicts between packets in SMART are presented in [34]. They focus on

communication contention, rather than communication distance, as contention degrades bypass utilization. An analytical model of SMART is presented in [4] to speed up simulations, reducing simulation time in two orders of magnitude with respect to cycle-accurate simulators. These works can be adapted to S-SMART++.

An already mentioned alternative to the SMART approach is the use of low-diameter topologies based on high-radix routers such as asymmetric high-radix topologies [1] or the Slim NoC (SN) [3]. Other high-radix designs that result in higher scalability are the concentrated mesh, flattened butterfly [16], express cube topologies [13] or and meshes with ruche channels [26]. The latter refers to channels that interconnect distant routers (at a distance determined by the ruche factor) with a focus on a tiled physical design methodology. These networks are not necessarily globally synchronous and do not require broadcasting global control signals, so their verification is simpler than SMART. However, the increase of the router's radix does not only increase the number of ports with their respective logic (buffers, VC registers, control logic, etc) but also the complexity of the switch and allocators, which typically impact router frequency and power. S-SMART++ is originally designed for mesh topologies, but combined designs that apply multihop bypass to some of these low-diameter topologies might be considered, particularly in meshes with concentration or ruche channels.

Although the baseline of the paper is the SMART implementation, there are alternative router architectures that reduce the router delay to 1 cycle, even including Link Traversal (LT). One example is the case of bufferless routers like [14], [25]. Bufferless NoCs employ neither buffers nor flow control, simplifying drastically the router microarchitecture and as a consequence the critical path. Nevertheless, they typically rely on deflection routing to avoid packet loss, which quickly degrades performance when the load grows. Traditional designs, such as one included in OpenSMART [22], may also implement all the router logic in a single stage. This significantly reduces the average cycle count. However, comprising all the functionality of a traditional router in one stage drastically increases its critical path. In [22] it is shown that SMART can perform multi-hops of 3 routers with the same cycle period as the 1-stage router, effectively neglecting any benefit of merging all the functions in a single stage. As observed in Figure 12a, S-SMART++ supports higher frequency, which may be translated to longer HPC_{Max}, and reduces the latency of consecutive multihops, resulting in clearly better performance than single-cycle routers. Finally, according to the comparison in Sections 4.3.2 and 4.3.3, S-SMART++ is expected to improve area and dynamic power by more than 78% and 40% respectively, thanks to its efficient buffer organization.

6 Conclusions

Multiprocessor designers target cost-efficient low latency NoCs as they have a direct effect on the memory hierarchy. Routers with single-hop bypass reduce router pipeline length while multi-hop bypass networks like SMART reduce the effective number of hops. SMART achieves very low latency in meshes, but each multihop incurs a significant delay requiring large values of HPC_{Max} to reduce latency and it requires multiple VCs to reach competitive bandwidth, which results in overly complex designs.

This work introduces S-SMART++, which combines different mechanisms to mitigate the limitations of SMART. The speculative bypass setup bypasses pipeline stages in consecutive multi-hops and makes the design effective even when short multi-hops are used. Thus, it reduces base latency and mitigates the impact of HPC_{Max}. Additionally, S-SMART++ does not require VCs to exploit the bandwidth of the network, avoiding the conservative buffer and bypass allocation policies of SMART. This drastically reduces the area, power and critical path delay.

The experimental evaluations show that, for similar frequency and performance, S-SMART++ reduces the amount of logic resources by more than 80% and power by 44.1% with respect to SMART. S-SMART++ drastically reduces the dependency of the base-latency with HPC_{Max} and even in the most conservative comparison, i.e. when using the maximum HPC_{Max}, base latency is reduced by at least 29.2% in terms of cycles or 45.3% in terms of time when scaling the frequency. Overall, S-SMART++ constitutes a highly efficient alternative for low-latency NOCs.

References

[1] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge. Scaling towards kilo-core processors with asymmetric high-radix topologies. In *HPCA*, 2013.

[2] Y. Asgarieh and B. Lin. Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(6):1–25, 2019.

[3] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler. Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability. In *ASPLOS*, 2018.

[4] D. Bhattacharya and N. K. Jha. Analytical Modeling of the SMART NoC. *IEEE TMSCS*, 2017.

[5] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[7] L. Chen, R. Wang, and T. M. Pinkston. Critical Bubble Scheme: An Efficient Implementation of Globally Aware Network Flow Control. In *International Parallel Distributed Processing Symposium (IPDPS)*, pages 592–603, May 2011.

[8] X. Chen and N. K. Jha. Reducing Wire and Energy Overheads of the SMART NoC Using a Setup Request Network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3013–3026, Oct. 2016.

[9] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, USA, 2003.

[10] B. K. Daya, C. O. Chen, S. Subramanian, W. Kwon, S. Park, 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 (ISCA)*, pages 25–36, June 2014.

[11] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE TPDS*, 1993.

[12] K. Duraisamy and P. P. Pande. Enabling High-Performance SMART NoC Architectures Using On-Chip Wireless Links. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(12):3495–3508, Dec. 2017.

[13] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Express Cube Topologies for on-Chip Interconnects. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pages 163–174, Raleigh, NC, USA, Feb. 2009. IEEE.

[14] M. Hayenga, N. E. Jerger, and M. Lipasti. Scarab: A single cycle adaptive routing and bufferless network. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 244–254. IEEE, 2009.

[15] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96, 2013.

[16] J. Kim, J. Balfour, and W. Dally. Flattened Butterfly Topology for On-Chip Networks. In *International Symposium on Microarchitecture (MICRO)*, pages 172–182, 2007.

[17] T. Krishna, C.-H. O. Chen, W. C. Kwon, and L.-S. Peh. Breaking the on-chip latency barrier using SMART. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 378–389, Feb. 2013.

[18] 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 *International Conference on Computer Design (ICCD)*, pages 439–446, 2010.

[19] A. Kumar, P. Kunduz, A. Singhx, L.-S. Pehy, and N. Jhay. A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *25th International Conference on Computer Design (ICCD)*, pages 63–70, Oct. 2007.

[20] A. Kumar, L.-S. Peh, and N. K. Jha. Token Flow Control. In *International Symposium on Microarchitecture (MICRO)*, pages 342–353, 2008.

[21] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express Virtual Channels: Towards the Ideal Interconnection Fabric. In *International Symposium on Computer Architecture*, ISCA '07, pages 150–161, 2007.

[22] H. Kwon and T. Krishna. OpenSMART: Single-cycle multi-hop NoC generator in BSV and Chisel. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 195–204, 2017.

[23] X. Lin, P. K. McKinley, and L. M. Ni. The Message Flow Model for Routing in Wormhole-Routed Networks. In *ICPP*, 1993.

[24] S. Ma, N. E. Jerger, and Z. Wang. Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2012.

[25] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 196–207, 2009.

[26] Y. Ou, S. Agwa, and C. Batten. Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology. In *International Symposium on Networks-on-Chip (NOCS)*, Hamburg, Germany, Sept. 2020. IEEE.

[27] G. Passas, M. Katevenis, and D. Pnevmatikatos. Crossbar NoCs Are Scalable Beyond 100 Nodes. *IEEE TCAD*, 2012.

[28] I. Perez, E. Vallejo, and R. Beivide. Efficient Router Bypass via Hybrid Flow Control. In *2018 11th International Workshop on Network on Chip Architectures (NoCArc)*, pages 1–6, Oct. 2018.

[29] I. Pérez, E. Vallejo, and R. Beivide. SMART++: Reducing Cost and Improving Efficiency of Multi-hop Bypass in NoC Routers. In *International Symposium on Networks-on-Chip (NOCS)*, pages 5:1–5:8, 2019.

[30] I. Pérez, E. Vallejo, M. Moretó, and R. Beivide. BST: A BookSim-based toolset to simulate NoCs with single- and multi-hop bypass. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.

[31] K. Sewell, R. G. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. F. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge. Swizzle-Switch Networks for Many-Core Systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2012.

[32] B. Towles, J. P. Grossman, B. Greskamp, and D. E. Shaw. Unifying On-chip and Inter-node Switching Within the Anton 2 Network. In *International Symposium on Computer Architecuture (ISCA)*, pages 1–12, 2014. event-place: Minneapolis, Minnesota, USA.

[33] R. Wang, L. Chen, and T. M. Pinkston. Bubble Coloring: Avoiding Routing- and Protocol-induced Deadlocks with Minimal Virtual Channel Requirement. In *International Conference on Supercomputing (ICS)*, pages 193–202, 2013. event-place: Eugene, Oregon, USA.

[34] L. Yang, W. Liu, P. Chen, N. Guan, and M. Li. Task Mapping on SMART NoC. In *DAC'17*, 2017.

BIOGRAPHY

Iván Pérez is a PhD student in Computer Architecture in the University of Cantabria, Spain. He received the B. Sc. degree in Telecomunication Engineer form the University of Cantabria in 2014. His research interests include networks on chip and memory hierarchy of manycore processors.

Enrique Vallejo received the PhD degree in computer architecture from the University of Cantabria, in 2010. He is an associate professor with the University of Cantabria, where he lectures Interconnection Networks in the Computer Science studies. His research interests cover different areas of parallel computing: Interconnection networks, processor microarchitecture, networks on chip, and synchronization mechanisms.

Ramón Beivide received a PhD degree in computer science and engineering from the Universidad Politecnica de Catalunya (UPC), Barcelona, in 1985. He has been an associate professor at UPC and the Universidad del Pais Vasco. In 1991, he joined the Universidad de Cantabria in Santander, Spain, where he is a full professor of telecommunication and computer engineering and he has served as the dean of the School of Computer Science. His research interests include parallel computers, interconnection networks, memory hierarchies and graph theory. He has published more than 100 technical papers on such topics.