SMART++: Reducing cost and improving efficiency of multi-hop bypass in NoC routers

Iván Pérez, Enrique Vallejo, Ramón Beivide
University of Cantabria, Santander, Spain.

ABSTRACT

Low latency and low implementation cost are two key requirements in NoCs. SMART routers implement multi-hop bypass, obtaining latency values close to an ideal point-to-point interconnect. However, it requires a significant amount of resources such as Virtual Channels (VCs), which are not used as efficiently as possible, preventing bypass in certain scenarios. This translates into increased area and delay, compared to an ideal implementation.

In this paper, we introduce SMART++, an efficient multi-hop bypass mechanism which combines four key ideas: SMART bypass, multi-packet buffers, *Non-Empty Buffer Bypass* and Per-packet allocation. SMART++ relies on a more aggressive VC reallocation policy and supports bypass of buffers even when they are not completely free. With these desirable characteristics, SMART++ requires limited resources and exhibits high performance.

SMART++ is evaluated using functional simulation and HDL synthesis tools. SMART++ without VCs and with a reduced amount of buffer slots outperforms the original SMART using 8 VCs, while reducing the amount of logic and dynamic power in an FPGA by 5.5× and 5.0× respectively. Additionally, it allows for up to 2.1× frequency; this might translate into more than 31.9% base latency reduction and 42.2% throughput increase.

KEYWORDS

SMART; SMART++; multi-hop bypass

ACM Reference Format:
Iván Pérez, Enrique Vallejo, Ramón Beivide. 2019. SMART++: Reducing cost and improving efficiency of multi-hop bypass in NoC routers. In *International Symposium on Networks-on-Chip (NOCS '19), October 17–18, 2019, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3313231.3352364

1 INTRODUCTION

Low latency in NOCs for a wide range of traffic loads is critical for multiprocessor performance. Different approaches have been considered for this goal, including very large crossbars (such as [19, 21]), low-diameter topologies based on high-radix routers (such as [1, 2]) or aggressive lookahead routing, speculative stages and router bypass mechanisms (such as [11, 12, 14]). SMART [11], 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 [9, 16]. 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. Finally, 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.

This work introduces SMART++, an efficient multi-hop bypass mechanism that avoids the main limitations of SMART and allows for much simpler implementations. SMART++ combines SMART bypass [11], multi-packet buffers, *Non-Empty Buffer Bypass* (NEBB, [20]) 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.

SMART++ is evaluated using both the functional simulator Booksim [18] and an HDL implementation based on OpenSMART [15]. SMART++ outperforms SMART requiring only simple changes, provides high performance using a single buffer of limited size per port, and is both area- and power-efficient.

Specifically, the main contributions of this paper are:

- SMART++, an efficient multi-hop bypass mechanism that outperforms the original SMART with much lower requirements on VCs, area and power.
- A performance evaluation by simulation, which proves that: SMART++ without VCs has similar performance than SMART with VCs for single-flit packets and the same buffer space; better performance for multi-flit packets; and 2x buffer reduction for the same performance for bimodal traffic.
- Resource utilization and power evaluations using HDL synthesis, showing the high cost of VCs, and the extensive and feasible variety of buffer configurations of SMART++.

NOCS '19, October 17–18, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6700-4/19/10…$15.00
https://doi.org/10.1145/3313231.3352364

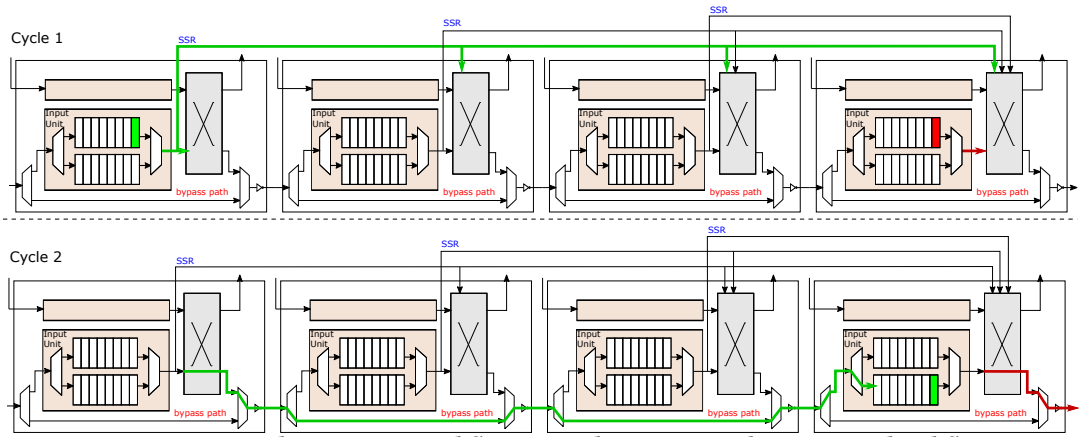


Figure 1: SMART bypass setup and flit traversal overview with priority to local flits.

Section 2 presents the required background. Section 3 details the SMART++ foundations, while Section 4 discusses implementation details. Section 5 evaluates the proposal. Finally, Section 6 compares to related work and Section 7 concludes the paper.

2 BACKGROUND

2.1 NoC Router bypass

Router bypass [12, 13] 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.

2.2 Non-Empty Buffer Bypass (NEBB)

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 [20] 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.

2.3 SMART: multi-hop router bypass

SMART (Single-Cycle Multihop Asynchronous Repeated Traversal) [11] 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 in the next cycle, 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 multihop at the cost of much higher complexity.

SSRs request access to the bypass in each of the downstream routers, in 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 multihop 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 [11]. Figure 1 shows an example of bypass setup and flit transmission. In the first cycle, the green packet (first router) and the red 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 red 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 [11], flits only bypass the input buffers in the input unit, but they pass through the crossbar. Buffer bypass is required for SMART_2D and for ejection router bypass. Router bypass, depicted in Figure 1 and used in this paper, was introduced in OpenSMART [15]. 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 SMART++

This section introduces SMART++, which supports multi-hop bypass even when *inter-router* buffers are not completely empty, as long as no packet interleaving occurs in any buffer. SMART++ targets simple designs with only one buffer, or just a few ones (one per virtual network required by the coherence protocol), leading to high frequency and reduced area and power consumption.

SMART++ combines three improvements over the baseline SMART: i) Multi-packet buffers, ii) *Non-Empty Buffer Bypass* for single-flit packets and iii) packet-by-packet arbitration to support multi-flit *NEBB* bypass. These mechanisms are detailed in subsections 3.1-3.3. Section 3.4 compares the different mechanisms, detailing in which cases each of them supports packet bypass.

3.1 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++ allows to hold multiple consecutive packets in router buffers and can exploit buffers larger than a single packet size. Such approach is similar to previous proposals for NoCs [4, 7, 17, 22, 23]. The implementation is similar to Whole Packet Forwarding (WPF [17]). WPF implements an aggressive VC reallocation mechanism, which allows to reallocate a given VC if it has enough buffer slots to hold the whole packet and the tail of the previous packet has been already sent. According to [17], *WPF can be viewed as applying packet-based flow control in a wormhole network*. Note that in SMART flit-by-flit arbitration behaves similar to a WH network.

The use of multi-packet buffers allows to employ a lower amount 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. This is evaluated in Section 5.2.

3.2 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 when the amount of VCs is low.

SMART++ employs NEBB [20] to bypass a buffer even when it is not empty. Such bypass is allowed as long as no two packets are interleaved in a given buffer. For single-flit packets, this never occurs as long as the destination buffer has already received the tail of the previous packet. For this reason, SMART++ relies on NEBB to bypass single-flit packets even when the input buffer is not empty. This only requires a policy change in bypass conditions.

For multi-flit packets, the flit-by-flit allocation mechanism in SMART implies that the bypass operation might be interrupted at any cycle, if a higher-priority SSR is received at an intermediate router. When this happens, the remainder of the packet is stored in the intermediate router buffer, and if it is not empty, packets would be interleaved and corrupted. For this reason, *NEBB* does not support multi-flit packet bypass with non-empty buffers when flit-by-flit allocation is employed.

3.3 Packet-by-packet arbitration (PPA)

As discussed in Section 3.2, flit-by-flit allocation prevents using NEBB with multi-flit packets. SMART++ solves this issue using packet-by-packet arbitration, which is implemented using a grant-hold circuit [6] coupled to the round-robin arbitration stages (SA-G and SA-L, discussed in Section 4.1).

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 no flit is 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 interleaving. Only packet headers generate SSRs. When a higher-priority SSR (lower distance, using *local* priority) is received in an intermediate router while a packet is being bypassed, it loses arbitration and is stored in the router buffers. 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].

3.4 Comparative analysis of the mechanisms

Table 1 summarizes the different cases in which bypass is supported in SMART and SMART++, detailing the specific contribution of each of the mechanisms SMART++ comprises. SMART can only forward data to and bypass empty buffers. MPB supports forwarding packets to non-empty buffers, but not bypassing. NEBB adds support for single-flit packet bypass of non-empty buffers, and the complete SMART++ design including PPA supports bypass of non-empty buffers for any packet size.

Figure 2 presents two examples of the conditions presented in Table 1, for both single- and multi-flit packets, sending a packet from R_0 to R_4. The original SMART mechanisms only sends data when empty VCs are available, so packets stop at R_1, which is the only transit router with empty buffers. MPB allows to employ non-empty VCs, but not bypass them, so packets stop at R_2, which is the first

[1]There are no cascading invalidations, as occurs with SSRs using Prio=bypass [11].

Table 1: Allowed bypass depending on the buffer status. *Bypass* and *Dest. buffer* refers to the buffers in the bypass router and in the next router. When multiple routers are bypassed, intermediate buffers are both *bypass* and *dest.* buffers. They may need to be completely *empty*, or may accommodate at least a whole *packet*.

Bypass mechanism	Bypass buffer: empty		Bypass & Dest. buffer: packet	
	Dest. buffer: empty	Dest. buffer: packet	1-flit packet	Multi-flit packet
SMART (Baseline)	✓	X	X	X
SMART+MPB	✓	✓	X	X
SMART+MPB+NEBB	✓	✓	✓	X
SMART++ (SMART+MPB+NEBB+PPA)	✓	✓	✓	✓

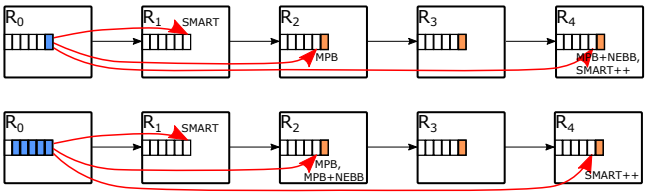


Figure 2: Stop router of each mechanism in SMART++ for single-flit (up) and multi-flit (bottom) packets. R_4 is the destination of the blue packet in R_0. Routers only have one buffer.

non-empty buffer in the path. MPB+NEBB allows to bypass non-empty VCs, but only for single-flit packets, so the packet reaches the destination only in the first (upper) case of single-flit packet. Finally, SMART++ allows to bypass non-empty VCs for any size of the packet, so in both cases the packet reaches the destination.

4 SMART++ IMPLEMENTATION DETAILS

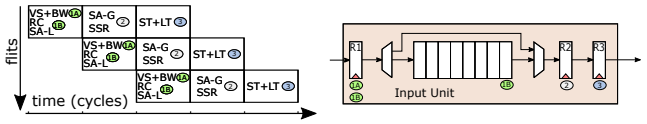
This section details the organization of the input units, and the buffer backpressure mechanisms.

4.1 Pipeline and input buffer architecture

SMART++ implements a three stage pipeline as depicted in Figure 3. In the **first stage**, flits are written in the VC selected in VC Selection (VS). In parallel, the router performs Route Computation (RC) for the next multi-hop and Switch Allocation Local (SA-L). SA-L grants access to SA-G to local flits. In the **second stage**, SA-L winners broadcast their SSRs. In parallel, Switch Allocator Global (SA-G) grants access to the crossbar. In the **third stage**, flits traverse the crossbar (ST) and link (LT) of the routers until finding the first bypass disabled.

SMART++ targets designs with few input buffers, ideally one. Such organization may introduce idle cycles, or bubbles, imposed by the architectural dependencies between the operations of consecutive packets reusing the same buffer, as described in [8]. Note that the three stages may need to access flit information in the same cycle, as presented in Figure 3a, generating architectural dependencies and stalls. Specifically, this would occur if flits were dequeued from the input buffer when they win SA-G. In a design without VCs, this would interrupt the transmission of packets sharing the same input buffer, because the second packet in the queue cannot place the request in SA-L while the front packet is doing SA-G. This pipeline bubble increases latency and reduces throughput.

SMART++ implements the input unit represented in Figure 3b to address this issue. The input unit has three pipeline registers *R1-R3*. Using *R2*, flits are dequeued when winning SA-L, transferring the following flit to the front position of the buffer. In this organization,



(a) Router pipeline. **(b) Input unit organization**

Figure 3: Router input unit organization and pipeline.

VS and BW in stage 1 read the flit from the first register (*1A*), whereas RC and SA-L also in stage 1 read their data from *1B*, i.e. the front of the input buffer (if it is not empty) or register *R1* (if the buffer is empty). In the latter case, the flit advances directly to *R2*. SA-G reads the flit from register *R2*, while ST reads the flit from *R3*.

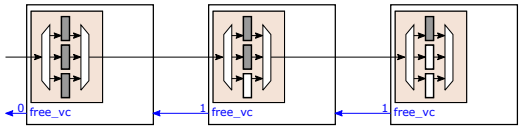
SMART++ implements VCT as defined in Section 3.3, so buffer credits may be handled per-packet. For single-flit packets, the credit is sent back to the upstream router when the flit advances to *R2*. However, flits may wait indefinitely in *R2*. For multi-flit packets, we notice that when the header advances to *R3*, it is sure that the packet flits will be transferred consecutively. For this reason, credit handling for multi-flit packets may be optimized as follows: when a packet header advances to *R2*, one credit is generated; when the first body flit advances to *R2*, the remaining credits are generated.

4.2 Buffer backpressure and VC selection

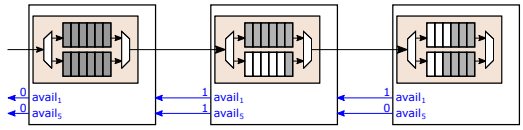
This section discusses different backpressure mechanisms supported by SMART routers, mainly credits and *free_VC* signals, and their extension to support SMART++ routers.

The buffer backpressure mechanism notifies the availability of buffer slots in the downstream router to receive a packet or flit. Credits and ON/OFF signals are frequently used mechanisms that notify the availability of each VC individually. Both of them allow the upstream router to track VC availability and select the destination VC for a given packet in the VC allocator (VA). When conservative VC reallocation is used, the ON/OFF mechanism is reduced to a single ON signal per VC, sent when its packet is completely forwarded. By contrast, when multiple packets are allowed per buffer and multiple packet sizes are supported using VCT, credits track the amount of free slots per VC. The implementation in OpenSMART [15] relies on credits, and SMART++ also supports this mechanism directly, with the optimization described in Section 4.1. The use of ON/OFF signals is only supported for single-size packets.

The original SMART implementation in [11] employs a different approach based on *free_VC* signals, depicted in Figure 4a. Each cycle, routers activate a *free_VC* 1-bit signal on those ports that have at least one free VC to receive a packet. Upstream routers do not know which VC is free, and instead send the packet blindly without VC Allocation (VCA); downstream routers assign a VC upon reception, in a VC selection (VS) operation.



(a) *free_vc* signals in SMART



(b) *Availability* signals in SMART++ with packet sizes of 1 and 5 flits.

Figure 4: Buffer signaling mechanisms.

The *free VC* mechanism requires minimal changes to support SMART++ routers. Instead of a single-bit flag, SMART++ employs one independent 1-bit signal per packet size allowed in the network, as depicted in Figure 4. This supports a case with free space for a single-flit packet but not enough space for a multi-flit packet. These signals are denoted *avail_i* to indicate the *availability* of some VC with enough free slots to hold packets of *i* flits (for example, *avail_1* and *avail_5* for 1 and 5 flits respectively). Obviously, when *avail_j* is set, *avail_k* will be also set $\forall k < j$.

SMART++ works with either credits or availability signals. In both cases, it needs to know in advance the size of the packet to be received, for the SA-G stage to decide to send the packet, if there is space for it, or store it in a buffer. The SSR signal is extended to indicate the packet size, using $\lceil log_2(N) \rceil$ additional lines for N packet sizes. For $N = 2$, this implies that SMART++ SSRs will employ only one additional line. In NOCs supporting a single packet size, the SSR network in both models is the same.

5 EVALUATION

This section evaluates SMART++. Section 5.1 describes the simulation infrastructure; Section 5.2 presents the cycle-accurate performance results of SMART++; Section 5.3 shows synthesis estimations of power, resource utilization and maximum frequency.

5.1 Simulation Infrastructure

Two development platforms compose the simulation infrastructure: BookSim [18] and Bluespec SystemVerilog (BSV). BookSim is an open-source functional simulator written in C++. We have implemented cycle-accurate models of SMART and SMART++, including the partial versions detailed in Table 1. The model implemented supports variable size packets. We have also implemented SMART++ in BSV based on OpenSMART [15][2]. We had to make significant modifications to the SMART model provided by OpenSMART, for it to compile and work properly (e.g. move ST from the second pipeline stage to the third). Like OpenSMART, this model is limited to single-flit packets and works with credits. *Router bypass* (Section 2.3) is implemented in both platforms.

The BSV implementation is used to validate the latency and throughput results of the BookSim model, through BSV functional simulations. To estimate power, resource utilization and frequency, the BSV compiler is used to generate Verilog code, and Quartus Prime 18.1 Lite Edition to synthesize and measure the desired metrics on an Arria II EP2AGX45DF29I5 FPGA.

[2]Based on OpenSMART's most recent official commit **84aa93b** on 27 Sep 2017.

Table 2: Simulation parameters.

Parameter	BookSim	Bluespec System Verilog
Topology	4x4 and 8x8 meshes	4x4 mesh
Bypass mechanism	SMART, SMART++ and partial versions	SMART and SMART++
Bypass type	SMART_1D with router bypass	
Router size	5 ports	
VC number & backpressure	1, 2, 4 or 8 VCs using credits	
Buffer size	1, 2, 4, 5, 8, 10, 15 or 20	1, 2, 4 or 8
Packet size (flits)	1, 5 or bimodal (80% of 1 + 20% of 5)	1
Routing	DOR XY	
VC selection policy	Shortest queue	First available VC
SSR policy	One dimension (SMART-1D) + Priority to local flits	
HPC_{MAX}	4 or 8	4
Flit size	128 bits	32 bits

Performance simulations on BookSim evaluate 8x8 meshes with $HPC_{Max} = 8$. Validation simulations are evaluated in 4x4 meshes with $HPC_{Max} = 4$ due to the large requirements of BSV compiler. In both cases local flits have priority over bypass. All the simulations use synthetic traffic. Three traffic patterns are evaluated: random uniform, bit-reversal and transpose. Moreover, we evaluate three packet sizes: single-flit packets, 5-flit packets, and bimodal traffic that combines single-flit and 5-flit packets following a distribution of 80% and 20%, respectively. It emulates the packet distribution observed in full-system simulations of the PARSEC benchmarks [17]. Table 2 gathers the most relevant simulation parameters.

5.2 Cycle-level Performance Results

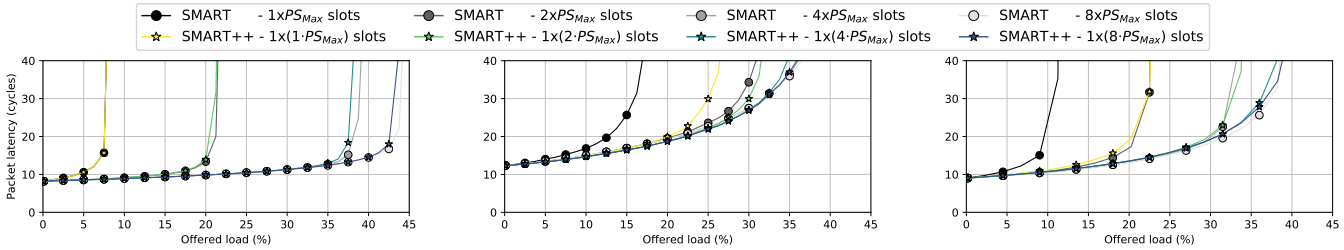
This section evaluates SMART++ without VCs, then with VCs, and finally the contribution of each of its mechanisms.

5.2.1 SMART++ without VCs. Figure 5 compares SMART with multiple VCs and SMART++ without VCs (or 1 VC), showing the average packet latency using single-flit packets, 5-flit packets and a combination of single-flit packets and 5-flit packets (bimodal traffic). The total buffering size (VCs × Buffer size) is the same in both cases, injecting random uniform traffic. With single-flit packets and the same total amount of buffering the performance of SMART++ is similar to SMART. In the case of 5-flit packets SMART++ achieves better performance when the buffer space is low, 5 and 10 slots, and similar for 20 and 40. With a 5 slots buffer, SMART++ outperforms SMART throughput by 48.7% with 1 VC. For bimodal traffic SMART++ is even better, requiring half of the buffer space of SMART to practically obtain the same performance.

Figure 6 depicts the packet latency using transpose and bit-reversal traffic patterns, with bimodal packet size distribution. Results with both traffic patterns are very similar, requiring buffer of only 10 slots to reach the maximum throughput, while SMART requires 4 VCs of 5 slots. Additionally, SMART++ with a buffer size of 5 slots improves the throughput of SMART with 2 VCs by 18.3% and 10.9% for transpose and bit-reversal.

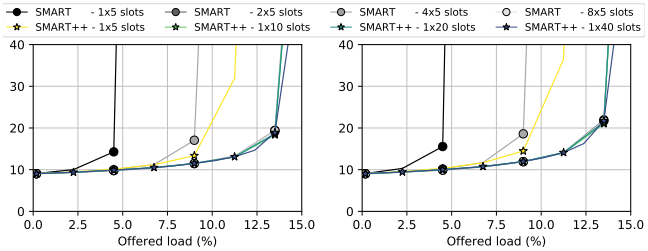
5.2.2 SMART++ with multiple VCs. Figure 7 compares the packet latency of SMART and SMART++ with the same number of VCs and buffer depth. The results of SMART++ with multiple VCs are slightly better than without VCs for the same buffer space, beating SMART in every configuration. The cause of the improvement is the reduction of HoLB.

5.2.3 Partial implementations of SMART++. Figure 8 depicts a breakdown of the improvements of the partial implementations of



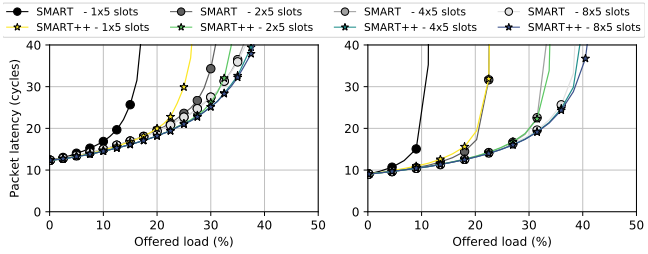
(a) Single-flit packet latency. (b) 5-flit packet latency. (c) Bimodal traffic packet latency.

Figure 5: SMART vs SMART++ packet latency for different packet sizes. SMART++ only employs 1 buffer (no VCs). PS_{Max} stands for maximum Packet Size in the simulation, e.g. $4 \times PS_{Max}$ with bimodal traffic means 4 VCs of 5 flits.



(a) Transpose. (b) Bit-reversal.

Figure 6: Latency of SMART and SMART++ without VCs for transpose and bitreversal traffic, with bimodal packets.

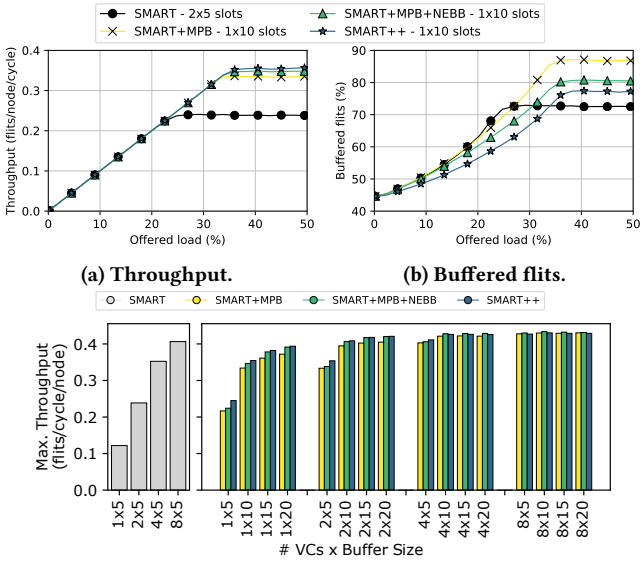


(a) 5-flit packet latency. (b) Bimodal traffic latency.

Figure 7: SMART vs SMART++ packet latency with multiple VCs and minimal buffer size per VC.

SMART++ (SMART+MPB and SMART+MPB+NEBB). Figures 8a and 8b show throughput and buffer utilization, respectively, injecting bimodal traffic. The buffer utilization represents the percentage of flits that performs BW when arriving a router. It shows that: SMART-MPB uses slightly more the bypass path than SMART before saturation; SMART-MPB+NEBB reduces the buffer utilization of SMART-MPB for every load achieving more throughput; SMART++ has the lowest buffer utilization and the highest throughput. Regarding the throughput obtained with buffers of 10 flits, SMART-MPB almost achieves the same throughput of SMART++: SMART+MPB increases SMART's throughput by 39.7%, SMART+MPB+NEBB by 45.1% and SMART++ by 48.5%.

Figure 8c depicts the maximum throughput injecting bimodal traffic, for different combinations of VCs and buffer depths. There are three remarkable conclusions. First, SMART requires a big amount of VCs to reach the limit of the NoC (0.5 flits/node/cycle in an 8×8 mesh). Secondly, SMART++ and its intermediate versions are very close to the maximum performance of SMART, but



(a) Throughput. (b) Buffered flits.

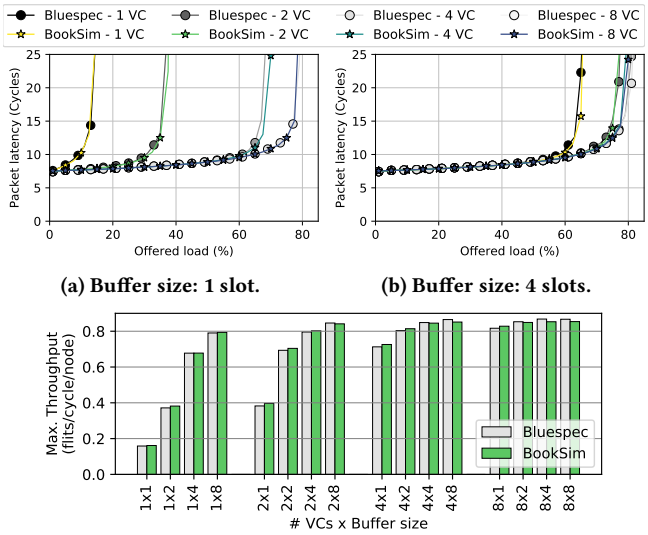
(c) Maximum throughput.

Figure 8: Performance metrics of SMART++ intermediate versions using bimodal traffic.

without requiring VCs and with half of the total buffer size. Particularly, SMART++ with one buffer of 20 slots achieves only 3.0% less throughput than SMART with 8 VCs of 5 slots. Thirdly, the main source of throughput improvement is forwarding packets to partially empty buffers (MPB) instead of empty. The improvement of SMART++ over its intermediate versions is more significant when the amount of resources is low.

5.3 SMART++ synthesis results

5.3.1 Model Validation. We first validate the models implemented in Booksim and BSV by comparing their results. The network is a 4× 4 mesh with $HPC_{max} = 4$, single-flit packets and random uniform traffic. Figure 9 shows the packet latency and maximum throughput of both implementations for multiple buffer configurations. The packet latencies obtained form both models are equal until reaching the saturation region where there is a negligible difference. In terms of maximum throughput the highest relative error between models is only 3.53%, when using 2 VCs of 1 slot. Hence, the SMART (buffer size 1) and SMART++ functional models implemented in BookSim cycle accurately simulate the router architecture and the pipeline, according to the HDL implementation.



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

(c) Maximum throughput.

Figure 9: Comparison of packet latency and throughput of the SMART++ models implemented in BSV and BookSim.

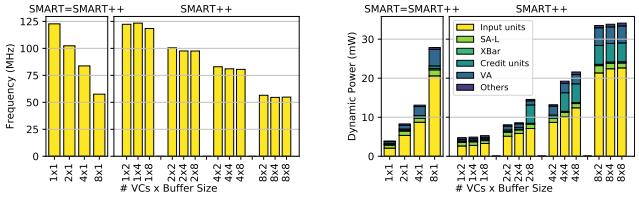
5.3.2 Resource Analysis. This and next sections analyze the resource requirements, the maximum frequency and the power consumption of a single SMART++ router. Figure 10 shows FPGA resources used by the synthesized routers, represented by the number of Adaptive Look-Up Tables (ALUTs), Adaptive Logic Modules (ALMs), dedicated registers and internal block memory bits. The results shows the high impact of the amount of VCs on resource demands as they directly affect to 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 with 2 VCs of 1 slot increases the number of ALUTs by 86.9%, ALMs by 82.3% and registers by 77.63% with respect to 1 VC of 1 slot. Alternatively, when using deeper buffers, the resource utilization grows in a much lighter way. For example, using 1 VC of 8 slots requires 22.3%, 31.4% and 63.14% more ALUTs, ALMs and registers, respectively, than using 1 VC of 1 slot. The compiler employs block memory (internal FPGA RAM) when buffers larger than 10 slots are instantiated. This occurs in the credit unit for configurations with more than 8 buffer slots. The credit units of OpenSMART has two FIFO structures per port to store credits pending to be transmitted. Each one has as many entries as buffer slots has an input port.

5.3.3 Timing and Power Analysis. Figure 11 depicts the maximum operation frequency and the dynamic power consumption for multiple router configurations. 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 50MHz, which fits in all configurations and is consistent with the cycle-level results in Section 5.2. In both cases, the results reveal that the number of VCs is a critical design factor. Doubling the number of VCs decreases frequency by 18% to 29% in each step. In terms of dynamic power, it almost doubles when duplicating the number of VCs. For example, using 2 VCs of 1 slot multiplies the power of 1 VC of 1 slot by 1.99. Increasing buffer depth has a negligible impact on frequency and moderately increases dynamic

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

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

Figure 10: FPGA resources employed by each configuration.



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

Figure 11: FPGA frequency and dynamic power results.

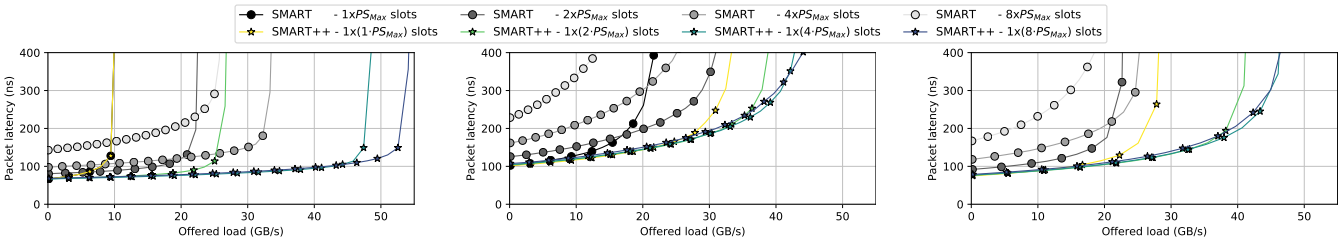
power, 75.84% in the worst case (2×8 compared to 2×4 slots). Comparing the 8 × 1 SMART and the 1 × 8 SMART++ configurations, resources are reduced by 5.49× on average and dynamic power by 4.99×. Notice that when using more than 8 slots in total, there is an abrupt increase as a consequence of using block memory bits in the credit units as depicted in Figure 10d.

5.3.4 Scaled SMART++ performance results. Section 5.2 presents cycle-accurate performance results of SMART and SMART++. However, Figure 11a shows that frequency in SMART++ may be significantly higher, further improving performance. Frequency is determined by the first router stages. When a given HPC_{max} value is considered, the delay of LT increases, and it might lower the router frequency. In such case, frequency in SMART and SMART++ would be similar, and their performance would be proportional to the figures in Section 5.2.

However, for moderate HPC_{max} and in FPGA evaluations, this typically does not occur because propagation delay is significantly lower than logic delay. In this case, performance will be determined by the maximum frequency in Figure 11a. Figure 12 presents frequency-scaled latency results of SMART and SMART++. The simpler SMART++ design using a single buffer with 4 packets (4 to 20 flits) clearly outperforms any SMART implementation. Comparing to a competitive SMART using 4 VCs, base latency is reduced at least by 31.9% and throughput increased by 42.2% in all cases.

6 RELATED WORK

Sections 1 and 2 have already discussed several alternatives to reduce latency on NoCs, including SMART details. OpenSMART [15]



(a) Single-flit packet latency. **(b) 5-flit packet latency.** **(c) Bimodal traffic packet latency.**

Figure 12: Frequency-scaled latency of SMART and SMART++ using different packet sizes.

is a NoC generator that generates verified RTL of SMART. We have extended it to support SMART++, within its limitations.

An SSR network is proposed in [5] to replace SSR broadcast wires and complex allocators to reduce wire and energy overheads. The approach is particularly relevant in SMART-2D, which we do not study for its complexity. WiSMART (wireless-enabled SMART [10]) 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. The combination with SMART++ is possible, but left for future work.

Task mapping techniques to reduce conflicts between packets in SMART are presented in [24]. They focus on communication contention, rather than communication distance, as contention degrades bypass utilization. An analytical model of SMART is presented in [3] to speed up simulations, reducing simulation time in two orders of magnitude with respect to cycle-accurate simulators. These works can be adapted to SMART++.

7 CONCLUSIONS

Power and area efficiency are essential features of NoC design. Their optimization is crucial for integrating NoCs in many-core processors. So far, SMART achieves the lowest latency in meshes. However, it requires a large amount of VCs to exploit the advantages of the mechanism due to its conservative VC reuse policy.

This work proposes SMART++, a multi-hop bypass mechanism that does not require VCs. SMART++ targets VC and bypass utilization, to allow multiple packets share the same buffer and bypass routers when their buffers are not empty. SMART++ exhibits high performance without VCs, reducing drastically power, area and critical path delay compared to configurations of SMART with multiple VCs. Moreover, SMART++ presents a more efficient utilization of buffers and bypass. Contrary to traditional wisdom, the use of multiple VCs in SMART++ provides just a marginal improvement.

For the same frequency and similar performance, SMART++ reduces area and power by 5.5× and 5.0×. Selecting the maximum frequency, SMART++ may reduce base latency by up to 31.9% and increase throughput by up to 42.2%. Altogether, the simple design in SMART++ simultaneously provides near-optimal performance with a small footprint and reduced implementation cost.

ACKNOWLEDGMENTS

This work was supported by the Spanish Ministry of Science, Innovation and Universities, contract TIN2016-76635-C2-2-R (AEI/FEDER, UE) and FPI grant BES-2017-079971, and the HiPEAC Network of Excellence. Bluespec Inc. provided access to Bluespec tools.

REFERENCES

[1] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge. 2013. Scaling towards kilo-core processors with asymmetric high-radix topologies. In *HPCA*.
[2] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler. 2018. Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability. In *ASPLOS*.
[3] D. Bhattacharya and N. K. Jha. 2017. Analytical Modeling of the SMART NoC. *IEEE TMSCS* (2017).
[4] L. Chen, R. Wang, and T. M. Pinkston. 2011. Critical Bubble Scheme: An Efficient Implementation of Globally Aware Network Flow Control. In *IPDPS*.
[5] X. Chen and N. K. Jha. 2016. Reducing Wire and Energy Overheads of the SMART NoC Using a Setup Request Network. *IEEE VLSI* (2016).
[6] W. Dally and B. Towles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Inc.
[7] B. Daya, C. Chen, S. Subramanian, W. Kwon, S. Park, T. Krishna, J. Holt, A. Chandrakasan, and L. Peh. 2014. SCORPIO: A 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC in-network ordering. In *ISCA*.
[8] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis. 2015. *Microarchitecture of Network-on-Chip Routers*. Springer New York.
[9] J. Duato. 1993. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE TPDS* (1993).
[10] K. Duraisamy and P. P. Pande. 2017. Enabling High-Performance SMART NoC Architectures Using On-Chip Wireless Links. *IEEE VLSI* (2017).
[11] T. Krishna, C.H. O. Chen, W. C. Kwon, and L.S. Peh. 2013. Breaking the on-chip latency barrier using SMART. In *HPCA*.
[12] A. Kumar, P. Kundu, A. Singh, L.S. Peh, and N. Jha. 2007. A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *ICCD*.
[13] A. Kumar, L.S. Peh, and N. K. Jha. 2008. Token flow control. In *Micro*.
[14] A. Kumar, L.S. Peh, P. Kundu, and N. K. Jha. 2007. Express Virtual Channels: Towards the Ideal Interconnection Fabric. In *ISCA*.
[15] H. Kwon and T. Krishna. 2017. OpenSMART: Single-cycle multi-hop NoC generator in BSV and Chisel. In *ISPASS*.
[16] X. Lin, P. K. McKinley, and L. M. Ni. 1993. The Message Flow Model for Routing in Wormhole-Routed Networks. In *ICPP*.
[17] S. Ma, N. Jerger, and Z. Wang. 2012. Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip. In *HPCA*.
[18] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. 2013. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *ISPASS*.
[19] G. Passas, M. Katevenis, and D. Pnevmatikatos. 2012. Crossbar NoCs Are Scalable Beyond 100 Nodes. *IEEE TCAD* (2012).
[20] I. Perez, E. Vallejo, and R. Beivide. 2018. Efficient Router Bypass via Hybrid Flow Control. In *NoCArc*.
[21] 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. 2012. Swizzle-Switch Networks for Many-Core Systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2012).
[22] B. Towles, J. P. Grossman, B. Greskamp, and D. E. Shaw. 2014. Unifying On-chip and Inter-node Switching Within the Anton 2 Network. In *ISCA*.
[23] R. Wang, L. Chen, and T. M. Pinkston. 2013. Bubble Coloring: Avoiding Routing- and Protocol-induced Deadlocks with Minimal VC Requirement. In *ACM ICS*.
[24] L. Yang, W. Liu, P. Chen, N. Guan, and M. Li. 2017. Task Mapping on SMART NoC. In *DAC'17*.