

CUTBUF: Buffer Management and Router Design for Traffic Mixing in VNET-based NoCs

Davide Zoni, José Flich, *Senior Member, IEEE* and William Fornaciari, *Senior Member, IEEE*

Abstract—Router's buffer design and management strongly influence energy, area and performance of on-chip networks, hence it is crucial to encompass all of these aspects in the design process. At the same time, the NoC design cannot disregard preventing network-level and protocol-level deadlocks by devoting ad-hoc buffer resources to that purpose. In Chip Multiprocessor Systems (CMPs) the coherence protocol usually requires different virtual networks (VNETs) to avoid deadlocks. Moreover, VNET utilization is highly unbalanced and there is no way to share buffers between them due to the need to isolate different traffic types.

This paper proposes *CUTBUF*, a novel NoC router architecture to dynamically assign VCs to VNETs depending on the actual VNETs load to significantly reduce the number of physical buffers in routers, thus saving area and power without decreasing NoC performance. Moreover, *CUTBUF* allows to reuse the same buffer for different traffic types while ensuring that the optimized NoC is deadlock-free both at network and protocol level. In this perspective, all the VCs are considered spare queues not statically assigned to a specific VNET and the coherence protocol only imposes a minimum number of queues to be implemented. Synthetic applications as well as real benchmarks have been used to validate *CUTBUF*, considering architectures ranging from 16 up to 48 cores. Moreover, a complete RTL router has been designed to explore area and power overheads.

Results highlight how *CUTBUF* can reduce router buffers up to 33% with 2% of performance degradation, a 5% of operating frequency decrease and area and power saving up to 30.6% and 30.7%, respectively. Conversely, the flexibility of the proposed architecture improves by 23.8% the performance of the baseline NoC router when the same number of buffers is used.

Index Terms—Networks-on-Chip, performance, power, router architecture, RTL, simulation

1 INTRODUCTION

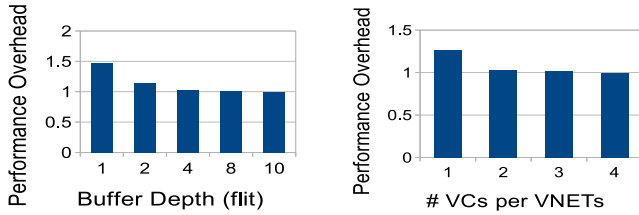
Networks-on-Chip (NoCs) is an accepted solution to cope with the increasing communication requirements in multi-core architectures. However, the huge amount of design parameters to be considered, i.e. buffer depth, number of VCs, pipeline organization, flit size and so on, makes the NoC design complex. Router buffers are recognized as those that mostly affect the overall behavior of the on-chip network and, consequently, the performance of the whole chip. Moreover, buffers consume a significant part of the NoC router power and storing a packet in a buffer consumes far more energy than its transmission [1]. Besides, the area occupied by an on-chip router is dominated by the buffers [2], [3]. Last but still important, the router buffer organization significantly impacts on the deadlock problem, both at network and protocol levels. This is more important when implementing strategies for Chip Multiprocessors (CMPs). In such systems it is typical to find a coherence protocol running on top of the NoC. This protocol injects messages of different types and the network needs to

separate them in different queues, i.e. Virtual Channels (VCs), thus imposing a multiplicative increase in the number of queues implemented in the NoC routers. Typically, each message type is steered through a so-called different virtual network (VNET), thus using only resources reserved for the specific VNET.

Indeed, we can relate the power/performance trade-off with the number of queues used in the NoC routers. The number of queues has to be sized in order to achieve a given performance, to avoid network deadlocks (provisioning enough queues for the routing algorithm) and to avoid protocol deadlocks (provisioning enough queues for the running protocol). Conversely, the power consumption of the NoC strongly depends on the number of queues. Thus, we have conflicting goals: reducing the queues for the power consumption without compromising the performance or even the correct system behavior. *CUTBUF* is a novel router architecture that efficiently manages the buffers by dynamically assigning more queues to the demanding VNETs and reusing each queue by packets of different types. The novelty is a NoC router with almost the same performance of the reference baseline NoC, but with a reduced number of implemented queues. To the best of our knowledge, this is the first comprehensive work addressing the reuse of virtual channels (VCs) by mixing the traffic pertaining to different VNETs at run-time with a per packet granularity, still avoiding deadlock at both routing and protocol levels.

The rest of this section is split in three parts. Section 1.1

- D. Zoni is with Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milano, ITALY. E-mail: davide.zoni@polimi.it
- José Flich is with the DISCA department, Universitat Politècnica de València, Valencia, SPAIN. E-mail: jflich@disca.upv.es
- W. Fornaciari is with Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milano, ITALY. E-mail: william.fornaciari@polimi.it



(a) Normalized buffer depth against the 10-flit buffer scenario. (b) Number of VCs per VNET, from 1 to 4, normalized w.r.t. the 4 VCs per VNET scenario.

Fig. 1: Performance impact varying both the count and depth of the buffer. Results are from a 16-core 2D-mesh running *qsort* benchmark (see Table 2 for parameters).

motivates the importance of a proper buffer design and management to address concurrently area, power and performance. The novel contributions of the paper are detailed in Section 1.2, while the structure of the whole paper is outlined in Section 1.3.

1.1 Rationale on Buffer Optimizations

The buffer design has a relevant impact on the NoC area and power as well as on the overall system performance [3], [4]. Aggressively reducing the number of buffers or their size to reduce silicon area and power consumption is not feasible, due to the impact on NoC performance. Buffers are a key resource because:

- they allow to decouple the communication in multiple separated hops, thus enhancing flexibility;
- increasing the depth of the buffers, mitigates the impact of the round trip time in NoCs exploiting a credit-based control flow (which can force to insert stalls in the communication between two routers);
- VCs allows multiple packets to share the physical channel, reducing the Head of Line (HoL) blocking effect, and increasing performance, even if multiple physical buffers per input port are required;
- coherence protocols impose multiple separated VNETs to route different types of messages avoiding deadlock. Each VNET requires dedicated buffers.

Figure 1 reports the normalized execution time for *qsort* (see MIBENCH [5]), considering a 16-core 2D-mesh multi-core and a link width of 64 bits. Different buffer counts per input port and buffer depths are considered in Figure 1b and Figure 1a, respectively. Figure 1a shows how the performance penalty is over 40% when single flit buffers are used, while increasing the buffer depth greatly improves the performance. On the other hand, increasing the buffer depth can have the drawback of low buffer utilization. In summary, a suitable (not trivial) buffer reuse strategy should be considered to tune the buffer depth. Results in Figure 1b are obtained considering a buffer depth of 4 flits and a buffer width of 64 bits. Figure 1b highlights a performance improvement by increasing the number of VCs per VNETs.

Conversely, Figure 2 reports the area breakdown considering a router split in five main components: buffers,

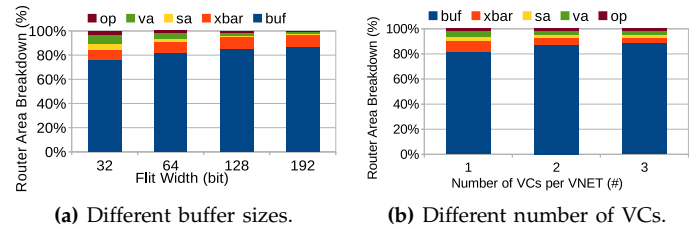


Fig. 2: Area breakdown varying the flit width and the number of virtual channels for three VNETs. Data have been obtained from an input-buffered wormhole router synthesized at 667GHz with 45nm technology. [6].

TABLE 1: Average buffer utilization per VNET, considering a 4×4 2D-mesh with MESI protocol implementing 3 VNETs. Data are normalized to the most loaded VNET and extracted from the router five (center of the mesh).

Benchmark	VNET-0	VNET-1	VNET-2
susan	0.21	1	0.045
qsort	0.29	1	0.05
dijkstra	0.24	1	0.03

crossbar, output port logic, Virtual Channel Allocation (VA) and Switch Allocation (SA). Different flit widths and number of virtual channels for each input port are considered. Again, buffers emerge as one of the first actors in determining the overall NoC area. Both buffer depth and buffer count reduction can satisfy the need to shrink the NoC footprint with a positive power reduction. For all the above reasons, buffer reuse and buffer sharing between VNETs schemes represent a suitable research path to balance power, performance and area.

The NoC design is burst oriented, i.e. it is tailored to support the maximum load due to the attached cores. Typically, the traffic is low for the majority of the time, with a poor resource utilization. Thus, a proper router design is crucial to reduce the buffers by compacting the traffic in few of them still ensuring similar performance.

Table 1 reports the average buffer usage between the VNETs, for some MIBENCH applications [5], considering a 4×4 2D-mesh and the MESI directory-based coherence protocol using 3 VNETs as implemented in GEM5 [7]. Results are normalized to the most used VNET (reference utilization of 1). It is evident the variable buffer utilization, spurring the search for solutions to exploit the unbalanced resource usage.

1.2 Novel Contributions

CUTBUF is a novel NoC architecture reusing at run-time the same buffer for different types of traffic. The goal is to improve NoC area and power by reducing the implemented buffers with a minimal impact on the overall performance. Three main contributions to the state of the art can be envisioned:

- *Dynamic VC allocation to VNET* - the possibility to assign any VC to any type of traffic at run-time depending on the actual needs of each VNET, allows to reduce the number of implemented VCs with minimal impact on performance.

- *Traffic Mixing and Queue Usage Minimization* - different types of message can be mixed on the same VC at different times still ensuring to be deadlock free at network and protocol levels. This confines the traffic in few buffers, increasing their utilization. Considering low traffic, only a single queue is used on every router port to manage all the traffic, thus enabling large power savings for the other ones (the power gating is not explored in this paper).
- *Power and Performance trade-off* - *CUTBUF* focuses on area and power savings by reducing the implemented buffers, with minimal impact on the overall system performance. Synthesis results highlight an area and power reduction up to 13.7% and 13.8%, respectively, for the *CUTBUF* implementing 5 VCs against the baseline with 6 VCs. Note that *CUTBUF* offers the same bandwidth of the baseline with 6 VCs using only 5 VCs. Although *CUTBUF* focuses on buffer reduction keeping the same performance, additional results show that by using the same number of queues *CUTBUF* outperforms the baseline router by up to 23.8%.

1.3 Paper Structure

The following Section 2 discusses the state of the art on NoC buffer design and management, while the *CUTBUF* architecture is detailed in Section 3. Area and power estimates based on RTL synthesis as well as simulation results for performance evaluation are provided in Section 4. Finally, conclusions are drawn in Section 5.

2 RELATED WORKS

Several works in literature focused on buffer design and management, due to their tight relation with NoC power, performance and area. [8] investigated the buffer sizing issues, but their solution is tailored to a specific application class, so that it lacks of flexibility when different traffic patterns are used.

[9], [10] proposed two buffer-less router microarchitectures coupled with a deflection-based routing algorithm ([9]) and a minimal routing algorithm with packet retransmission capabilities ([10]). The complete absence of buffers drastically reduces both area and power of the router, but it confines the solution to low traffic scenarios, thus imposing great performance penalties considering medium and high NoC loads.

Elastic buffers [11] makes use of the buffer space inherent in pipelined channels to reduce buffer cost. While removing virtual-channel buffers (VCBs) prevents efficient flow control and deadlock avoidance, elastic buffers exploit physical channels duplications by using them in the same way as VCs are used to prevent deadlock and define traffic classes. In this perspective, *CUTBUF* methodology relies on VCBs, although it is totally orthogonal to elastic buffers, since it can be easily adapted by replacing the use of buffers with the use of physical channels.

[4] discussed an extension of Duato's theory [12]. A packet that is using an escape path is allowed to be restored to an adaptive one thanks to the concept of *Whole Packet Forwarding* (WPF). WPF allows to allocate a complete packet from an escape VC to an adaptive one, still avoiding the generation of possible network deadlocks. *CUTBUF* exploits a somewhat similar scheme called *buffer reuse* as one of the implemented optimizations, whose differences w.r.t. WPF are discussed in Section 3.3. Our work is orthogonal to [4], since we focus on buffer reduction with guaranteed performance. Moreover, *CUTBUF* can be applied regardless to the NoC topology and to the routing algorithm.

[13] presented a *flit-reservation* flow control strategy. An additional network is used to send control flits before sending the data, to reserve the buffer resources, thus increasing the network throughput and the resource utilization. This work is similar to the *CUTBUF buffer reuse* strategy (see Section 3.3), but *CUTBUF* does not require an additional control network. Besides, the packet that will reuse the buffer, thus exploiting the *CUTBUF buffer reuse* solution, is not required to be of the same type, namely of the same VNET, of the previously stored one.

[14] discussed the NoC architecture exploited in the Single Cloud Chip (SCC) from Intel. The implemented VCs are partitioned between routing VCs and performance. The routing VCs are fixed and are used to ensure network deadlock avoidance while the performance ones are used to increase the performance. [14] focuses on the performance improvement. Conversely, *CUTBUF* focuses on buffer reduction keeping almost the same performance with three main differences from [14]. The SCC is a non-coherent chip, while *CUTBUF* expressly focuses on coherent multi-cores. Moreover, *CUTBUF* considers all the VCs freely allocable to each VNET and clearly states conditions to do so without generating both network and protocol level deadlocks. Last, *CUTBUF* can allocate different packet types in the same physical VC even at the same time. This is supported by the fact that no VC is reserved to a particular VNET.

ViChar [3] implemented a buffer size regulator design to dynamically resize buffers depending on the actual length of the stored packet. Buffers are assigned on a per flit basis following a daisy-chained list approach. *ViChar* can reduce the buffer size up to 50% while the area overhead due to the additional logic limits the gain up to 30% of the area reduction with the same performance. [15] proposed an *ViChar*-based scheme where dynamic buffer allocation is performed on a per router basis instead of on a per input port. It is worth noticing that the additional required logic can drive to excessive area if small flit sizes are considered, i.e. 32-bit flit width as in commercial Tiler solutions [16]. Thus, *CUTBUF* tries to obtain the same benefit in terms of area power and performance of *ViChar* by leveraging different properties of the NoC traffic. Moreover, *CUTBUF* analyses and solves both network and protocol-level deadlock considering both synthetic and real traffic, while *ViChar* does not

consider real traffic nor the protocol-level deadlock issue.

ElastiStore [17] proposed a buffer management scheme that exploits a single shared queue between all the VCs of a specific input port providing area and power reduction while ensuring almost the same performance of the original NoC. *CUTBUF* works in the same direction. However, it implements a totally different strategy aiming at the one-buffer-for-all the traffic types idea. Finally, *ElastiStore* requires a complex fully associative implementation for the shared queue, since at each cycle each flit in such a queue can be read.

3 PROPOSED METHODOLOGY

CUTBUF is a novel NoC router allowing to reuse the same VC for different message types. Section 3.1 presents the baseline router, while the other sections illustrate the specific optimizations of *CUTBUF*. The *Switch Allocation Flow* (SAF) strategy is presented in Section 3.2. Sections 3.3 and 3.4 detail the buffer and VNET reuse strategies, respectively. Section 3.5 addresses the *buffer remapping* strategy.

Figure 3 depicts the *CUTBUF* router. Each change to the baseline microarchitecture is highlighted, focusing on the required and provided information from and to each logic block. This figure is referenced multiple times and, for the sake of clarity, additional drawings are provided to highlight specific details of the optimizations.

3.1 Baseline NoC Router

We focus on a wormhole router which supports VCs and VNETs, with a standard 4-stage pipeline, i.e. Buffer Write/Route Computation (BW/RC), Virtual Channel Allocation (VA), Switch Allocation (SA), and Switch Traversal (ST). A single cycle for Link Traversal (LT) is assumed. The router implements atomic VC allocation. The proposed solution considers the VNET implementation to enable coherence protocol support at NoC level, preventing the traffic from one VNET to be routed on a different one. A packet is considered split in multiple atomic transmission units called flits. The first flit of each packet is the head flit. A body flit represents an intermediate flit of the original packet while the tail flit is unique for each packet and closes the packet. When a head flit arrives to the input port of the router it has to pass through the four pipeline stages before traversing the link. First, it is stored in the VC buffer (BW) which has been reserved by the upstream router, and the output port is computed (RC). Then, it competes in the VA stage to reserve an idle virtual channel in the selected output port. Note that assigned VCs belong to the set of VCs associated to the VNET of the packet. Once the VA succeeds, head, body and tail, competes in packet order for a crossbar switch path (SA). Finally, each winning flit in the SA has to pass the ST and link traversal LT before reaching the next router. Tail and body flits pass through fewer stages, since they reuse resources and information reserved by the head flit (i.e., VC and RC).

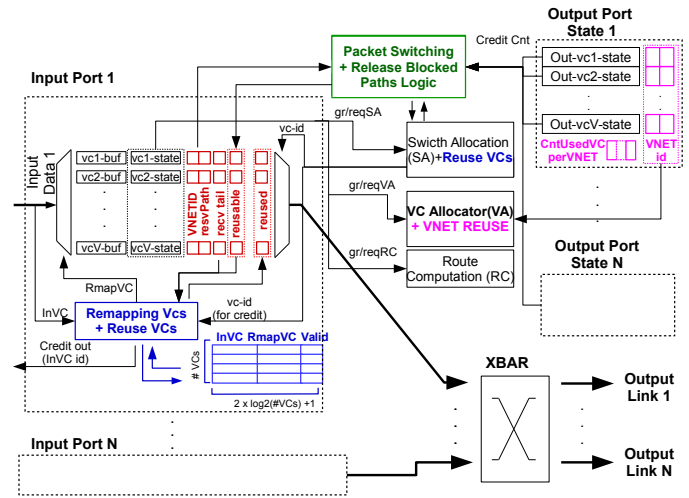
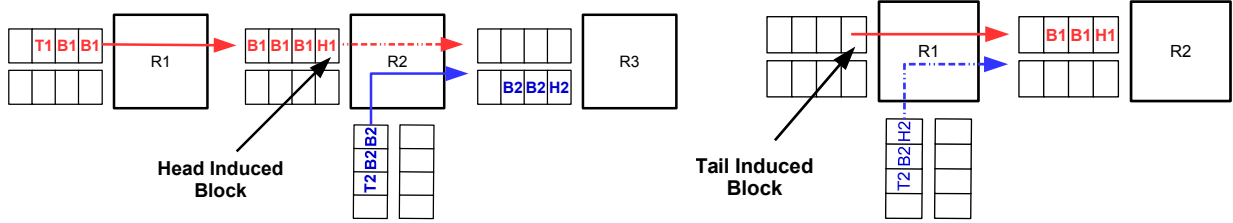


Fig. 3: Logical view of the *CUTBUF* router. The focus is on the added logic and the data/information required and provided by each additional logic block tailored to implement *CUTBUF*.

3.2 Switch Allocation Flow (SAF) and Release Blocked CrossBar Path Optimization

Traditionally, wormhole virtual channeled NoCs exploit flit-level switching, thus each flit of the packet must compete to access the crossbar. Flit-level switching coupled with a round-robin arbitration scheme is the standard solution to fairly arbitrate the crossbar. It allows for high throughput since, even if a packet has its output port blocked, a flit of another packet from the same input port and a different VC can be granted to cross the switch. However, multiple buffers in the same router are active, posing serious concerns on power saving and the number of buffers to be implemented. *CUTBUF* implements the switch allocation flow SAF mechanism proposed in [18] as an alternative solution to flit switching to significantly reduce the required number of concurrently active buffers. At the SA stage, the flits of the packets which in the previous cycle won the SA, have priority over the others. This allows to keep all the flits of the same packet as close as possible. Using *SAF*, an input-output path in the crossbar is assigned to a packet when its head flit wins the SA stage and the reserved path is released when either the tail flit traverses the same pipeline router stage or when the packet gets blocked (see below). In the ideal case, *SAF* allows to have a single active buffer per input port, minimizing the need of physical buffers in the NoC router.

The baseline *SAF* may exhibit performance degradation due to both the *Head Induced Block* (HIB) or the *Tail Induced Block* (TIB) (see Figure 4). *HIB* occurs when a packet having a reserved path in the crossbar is blocked because the downstream router input buffer slots are exhausted due to congestion. *TIB* happens when a packet having a reserved path in the crossbar has no flit ready to traverse the switch in the input buffer. The *Release Blocked Crossbar Path* (RBCP) optimization checks, at each cycle, if credits are available for each packet having a reserved crossbar path, and if the granted packet has at



(a) Head Induced Block by router R2 with respect to red packet. (b) Tail induced block by R1 with respect to blue packet.

Fig. 4: Two types of blocks that can limit the NoC performance while using the SAF strategy for crossbar allocation.

least one flit stored in the input buffer ready to traverse the crossbar. Both conditions must be satisfied, otherwise *RBCP* releases the path for that packet which has to compete again to access the crossbar.

Implementation - *SAF* and *RBCP* techniques affect the SA stage and the BW/RC stage (see in Figure 3 the green block and the red additional state bits in the input port). The input unit has an additional bit for each VC signaling (when set) that the VC has the path reserved in the crossbar. This bit is set when the input VC receives a grant from SA, and is kept while the VC is requesting the SA. If, for any reason, the VC does not request the SA or the grant does not arrive from the SA, in the next cycle the reserved path bit is cleared (see *resvPath* bit in Figure 3). In order to support *SAF*, the *resvPath* bit is forwarded to the SA (via the *SAF/RBCP* block) together with the request for SA. If it is set, a filter stage clears all the requests to the SA having an input or an output port shared with the input VC which has the *resvPath* bit set. Note that two VCs (from the same or different inputs) aiming at the same output port can not have both *resvPath* bits set at the same time (*SAF* is enforced).

3.3 Buffer Reuse

Low buffer utilization in wormhole virtual channeled NoCs has two motivations. First, it may be necessary to assign a single buffer for each packet to avoid deadlock at network level (atomic buffer allocation). Second, the round-robin switch allocation strategy to optimize the crossbar utilization leads to the underutilization of the buffers, since multiple buffers are kept active storing flits of packets waiting for the crossbar arbitration.

Thus, reassigning buffers to a new packet as soon as possible can increase their utilization. Traditionally, in conservative VC reallocation, a VC can be re-allocated to a new packet only if the tail of its last allocated packet is sent out, i.e. it is empty [19]. *Buffer reuse* allows to reassign non empty buffers, if the packet that is stored in the buffer is guaranteed to traverse the switch in a fixed, i.e. deterministic, number of cycles. In such a way, the newly allocated packet is guaranteed not to block because of the previous one stored in the same buffer. Moreover, the new packet can be of any type regardless the type of the previous one using the same buffer. In particular, *buffer reuse* requires to fulfill the following three conditions:

	Upstream Router					Downstream Router													
	RC	VA	SA	ST	LT	RC	VA	SA	ST	LT									
H1																			
B1																			
T1																			
H2																			
B2																			
T2																			
Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				

Fig. 5: Two packets, P1 and P2 composed of three flits have been sequenced on two routers, i.e. upstream and downstream. A single buffer assigned to P1 can be reused by P2 at cycle 9. The SA stage on flit B1 sets the reuse bit (yellow SA) that is forwarded to the input unit that is storing H2 at the same cycle (red RC).

- *Switch allocation flow* - the use of *SAF* eventually coupled with the *RBCP* optimization, ensures that a granted packet will exit the buffer in a finite number of cycles. However, the possibility of blocking imposes to check the following two additional conditions.
- *Tail Received* - to declare a buffer *reusable*, the tail flit of the packet that is using it, has to be in the buffer. In such a way, no other flits of the packet will reach the buffer in the future. Note that this check prevents the TIB effect (Section 3.2).
- *Downstream buffer credits* - let us consider an upstream/downstream router pair with a packet stored in the upstream router's input buffer. Once the packet has been granted by the upstream router and the tail flit is stored in the input buffer, it is required that the downstream router's input buffer has enough credits to store all the flits that are in the input buffer of the upstream router. This check prevents the HIB effect (Section 3.2).

Figure 5 reports a scenario where two packets, i.e. blue and green, traverse an upstream/downstream router pair in sequence (blue packet first), supposing infinite buffer size for the specific case, to ease the discussion. Cycle 9 represents the critical point where *buffer reuse* takes place. The yellow box highlights the SA stage for B1 flit of the blue packet. At this time, the buffer used by the blue packet can be declared reusable, since the corresponding T1 flit has been stored at cycle 8 and the packet will not block due to the infinite buffer assumption, i.e. the credit count in the downstream router buffer is greater than the stored flits in the considered buffer. Moreover, H1 of the green packet reaches the downstream router at cycle 9, thus it requires a buffer. In

this perspective, we implemented a forwarding mechanisms to guarantee that the buffer reuse decided in the SA stage, i.e. yellow box, is propagated to the BW/RC for the head of the green packet, i.e. red box, in the same cycle. As a result, a single buffer is required to host two subsequent packets.

Buffer reuse is similar to the *Whole Packet Forwarding* (WPF) scheme with two main differences. First, WPF is used in fully adaptive routing to restore a packet that is traveling in the escape VCs to the adaptive ones. Thus, to restore a packet from an escape VC to a fully adaptive VC, WPF requires that the destination VC has enough slots to completely store the packet to avoid network level deadlock. Second, WPF focuses on single-flit packets and both packets stored in the same buffer have to be of the same type.

On the other hand, the proposed *buffer reuse* strategy only requires that a non-idle VC can be used to store a new packet, i.e. reused, if and only if the actual stored packet is guaranteed to exit in a fixed amount of cycles. Note that no assumptions on the types of the two packets nor the need to have space in the destination buffer to entirely store the new packet are required, thus making our *buffer reuse* more general than WPF.

Implementation - Starting from the baseline router pipeline, the *buffer reuse* requires changes in two different stages. As detailed in Figure 3, the blue box highlights the required logic, while the red additional bits added for each buffer are used to store the required information. Two bits are used for each buffer. The *REUSEABLE_BIT* signals the possibility to reuse the buffer while the *REUSED_BIT* indicates if the buffer is actually being reused. When a head flit traverses the BW/RC stage, the logic checks the *REUSEABLE_BIT* and it is designed to use first the buffers with the *REUSEABLE_BIT* set, to optimally limit the number of active used buffers in the input port. The *REUSEABLE_BIT* is set by the SA stage when the three conditions described above are met. The *REUSEABLE_BIT* is cleared when the tail flit passes through the SA stage. This means that the buffer is idle starting from the next cycle or another packet is in the buffer, if the buffer has been reused before. In both cases, the buffer cannot be considered reusable.

The second bit, i.e. *REUSED_BIT*, signals if the buffer is actually being reused, since a single reuse per buffer is supported in the current implementation. The *REUSED_BIT* is set when the head flit of the packet is stored in a non empty buffer, according to the state of the *REUSEABLE_BIT*, while it is cleared when the tail of the packet on the front traverses the switch. Thus, the bit indicates the condition when parts of two packets coexist in the same buffer.

3.4 Virtual Network Reuse

Protocol-level deadlocks are typically avoided by imposing the use of multiple VNETs to separate different

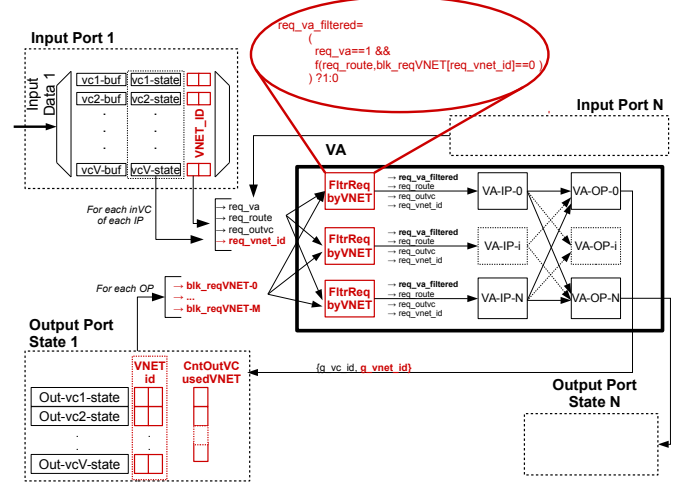


Fig. 6: Logical view of the *VNET Reuse* block. The focus is on the added logic and the data/information required.

types of message. In addition, VCs may be used to improve performance. In this scenario, multiple buffers ($VNET \times VC$) are dedicated, thus exacerbating the risk of low buffer utilization. The proposed *Virtual Network (VNET) Reuse* strategy allows to relax the constraint to restrict the use of separate VNET buffers for a specific type of message. In particular, it allows to allocate (at each hop) a packet on a buffer regardless the virtual network it belongs to. The proposed optimization increases the buffer utilization by mixing traffic that is originally routed to different VNETs, in fewer virtual channels.

The *VNET Reuse* strategy relies on the existence of at least one queue per VNET to avoid possible deadlocks. Indeed, by considering q queues, and M types of messages, at least one queue should be available for a given VNET, regardless of the status of the other VNETs. Thus, q should be equal or higher than $VNET$. Note that *VNET Reuse* may introduce out-of-order issues since packets in the same VNET may take different queues, thus reaching the destination garbled. However, this situation is also present in the baseline router (multiple VCs per VNET), being this issue orthogonal to *CUTBUF*.

Implementation - The changes to support the *Virtual Network (VNET) Reuse* affect the output ports, the virtual channel allocation (VA) stage and the Input Unit (see red blocks in Figure 6). The Input Unit has to store the VNET id for each incoming head flit, and to provide it to the VC allocator unit. The Output Unit is enhanced in two ways. First, each output VC has an additional register to store the used VNET id. Moreover, additional logic is used to compute the number of used output VCs dedicated to each VNET. Starting from these information, each output port computes the blocking N bits, i.e. one per VNET. Each bit, if set, means that an additional output VC can be allocated to the port. If the bit is cleared then all the requests to this output port, pertaining to traffic from the VNET the bit is representing, cannot be granted. The blocking bits, i.e. *blk_reqVNET*, are collected by the VA to filter the requests from the input VCs. In

particular, each input VC requiring for the VA has to provide its VNET id. A filter module operates on each input VC request by checking if it can be arbitrated or not, depending on the blocking bits. In particular, for each request from an input VC, the following condition is used to update the corresponding $req_va_filtered$:

$$req_va_filtered = (req_va f(req_route, blk_reqVNET(req_vnet_id)))?1 : 0 \quad (1)$$

where req_va is the request for the VA stage, while $f(req_route, blk_reqVNET(req_vnet_id))$ is a mapping function which returns true if the requested output port, i.e. req_route , can allocate more output VCs for the req_vnet_id VNET id, otherwise it returns false. If the condition is valid, the $req_va_filtered$ bit is set for the specific input VC, otherwise it is cleared preventing the VA to grant it. All the req_va that successfully passes the filtering stage traverse the classic two stages VA. Once the VA grants a request, both the notification and the VNET id of such request packet is forwarded to the corresponding output unit to update its state.

3.5 Buffer Remapping

Given a router pair, the Virtual Channel Allocation (VA) stage takes place in the upstream router to select the VC that will be used to store the packet in the downstream router input buffers. However, the upstream router does not know the actual state of the downstream buffers when the first flit of a packet arrives to the downstream one, since there are few delay cycles due to the router pipeline between such an event and the VA. Moreover, *buffer reuse* takes place in the downstream router to aggressively reuse the same buffer with different packets regardless their types. To this extent, the *buffer remapping* strategy is functional to the *buffer reuse* one, since it allows a late binding between the packet and the actual used buffer in the downstream router. The *buffer remapping* strategy faces these two issues. Once the packet's head flit actually reaches the input port, the *buffer remapping* logic properly selects the input buffer to store the packet, regardless the assigned VC in the upstream router. A physical buffer is selected in the downstream router to store the flits of the packet, while the credit flow is sent to the upstream router depending on the state of the actual assigned buffer. The upstream router sees the credit flow as belonging to the original selected VC during the VA stage. In this perspective, nothing needs to be modified in the baseline router, since from the logical viewpoint the optimization is transparent to the upstream router.

Implementation - The remapping logic is organized in two main components (see blue blocks in Figure 3). First, a *remapping table* on a per input port basis is implemented to store the original VC assigned by the upstream router, and the remapped VC assigned by the *buffer remapping* module. Second, a *logic block*

is introduced to select the buffer to be actually used to store the incoming flits of the packet. It operates in two phases. Initially it checks if a reusable buffer is present by inspecting both the *REUSEABLE_BIT* and *REUSED_BIT* bits; if it is not the case, then the flit is allocated on an idle buffer. To this purpose, the following aspects must be addressed:

- Remapping occurs when a head flit reaches the input port. In particular, the remapping table is updated when the head flit arrives to the input port following a simple rule that tries to allocate first the buffers with the *REUSEABLE_BIT* set. The mapping is kept until the tail flit leaves the buffer and then the record is deleted at the same time the backward credit is sent to the upstream router.
- The remapping table is updated when a head flit is processed; thus if a buffer is marked as reusable, the physical buffer identifier is associated to the new virtual channel assigned by the upstream router, since no more flits of the packet in the front of such buffer are expected.
- There is always an available buffer reusable or idle to store the flit of a new packet, since in the upstream router the VA succeeded and *buffer remapping* only reallocates buffers.

3.6 Deadlock Analysis

CUTBUF could introduce deadlocks as it stores in the same queue messages from different VNs. Here we argue the deadlock-free nature of *CUTBUF*. A protocol-level deadlock occurs when messages of different classes (e.g., request-response messages) are mapped in the same queues and messages of one type block messages of other types: introducing message type dependencies leads to deadlock [20]. Indeed, VNETs are used to avoid such deadlocks, by guaranteeing that messages of one type always reach destinations, bypassing blocked messages of different types.

Both *buffer reuse* and *VNET reuse* strategies may introduce deadlocks as they reuse queues. However, consider a baseline NoC using a deadlock-free routing algorithm (e.g., XY routing). The *buffer reuse* strategy still maintains the absence of deadlock by construction. In particular, a buffer can be reused only when it is guaranteed by the SA arbiter that the stored packet will leave the buffer in a finite number of cycles. This is guaranteed as packet-level crossbar switching is performed and the arbiter checks the number of remaining flits and the reception of the whole packet (tail flit received). Thus, the packet will never block the incoming message that is reusing the buffer. Moreover, the packet that eventually reuses such buffer will have its head flit on the top of the queue in a finite number of cycles. Therefore, there is no extra blocking conditions for packets. Furthermore, both packets belong to the same VNET and thus they are compatible and can share the queue with no chances of inducing protocol-level deadlocks.

TABLE 2: Experimental setup: processor and router micro-architectures and technology parameters.

Processor core	667MHz, out-of-order core
Int-ALU	4 integer ALU functional units
Int-Mult/Div	4 integer multiply/divide functional units
FP-Mult/Div	4 floating-point multiply/divide functional units
L1 cache	64kB 2-way set assoc. split I/D, 2 cycles latency
L2 cache	512kB per bank, 8-way associative
Coherence Prot.	MESI, 3 virtual networks
Router	4-stage wormhole with virtual channels with 64bit link width 4fl/VC
Topology	2 VCs per VNET (1 VC per VNET for the <i>ctbf-full</i>)
Technology	2D-mesh, 4x4 NoC @ 667MHz, 1 or 3 core per tile 45nm at 1.1V

Similar to *buffer reuse*, *VNET reuse* optimization relies on the possibility to reuse the same buffer at the input port. However, in this case, *VNET reuse* may lead to protocol-level deadlock as it can share the same queue to messages of different types, i.e. from different VNETs. In this perspective, *VNET reuse* implemented on the top of the baseline NoC, allows to use each buffer for any type of traffic, but ensuring that at least one VC can be reserved or is already reserved for the remaining VNETs. Indeed, an incoming packet will have always a queue guaranteed at the next hop for its own VNET; note that we keep at least one queue per VNET, hence two packets from the same VNET will never block two queues in a given input port. Thus, such packet cannot experience protocol-level deadlock nor network-level deadlock, since the routing algorithm is deadlock-free.

It has to be underlined that *VNET reuse* does not improve performance but reshapes traffic on few physical buffers, when possible. Moreover, the use of *VNET reuse* and *buffer reuse* cannot lead to deadlock, since once a buffer is reused, the VNET is checked by the *VNET reuse* optimization to see if it is the packet that is reusing the buffer, and not the one in front of the queue, since the packet in front will leave the queue in a finite number of cycles. As a conclusion, protocol-level deadlock is avoided by guaranteeing that: i) buffers are reused only when the message leaving the router will not block, and ii) for blocked messages in an input port, at least one queue is reserved for each different VNET.

4 EVALUATION

This section discusses the evaluation of *CUTBUF* from different viewpoints. The experimental setup is described in Section 4.1. Results with real and synthetic traffic patterns are discussed in Sections 4.2 and 4.3, respectively. The scalability analysis using PARSEC applications up to 48-cores is detailed in Section 4.4. Section 4.5 discusses area, power and timing numbers extracted from the RTL Verilog implementations for the baseline and the *CUTBUF* routers varying different parameters, namely buffer depth, flit size and number of queues. Last, Section 4.6 presents the impact *CUTBUF* has on buffer utilization.

4.1 Experimental Setup

Although the *CUTBUF* methodology proposed in Section 3 is generally applicable, the assessment has been

TABLE 3: Evaluated NoC microarchitectures.

NameID	Details
baseline	Baseline NoC without <i>CUTBUF</i> optimizations (2VCxVNET, 3VNETS).
ctbf	<i>CUTBUF</i> with all the optimizations using 1 VC per VNET. No additional VC for performance is present. This is the most power and area saving <i>CUTBUF</i> implementation (1VCxVNET, 3VNETS).
ctbf-(3+i)	It is <i>ctbf</i> where $i \in 1..3$ identifies the number of spare VCs implemented to improve performance. Note that <i>ctbf</i> -(3+3) has the same number of VCs of the <i>NoC-base</i> (1VCxVNET, 3VNETS + (1..3) spare VCs)

performed on 16-core and 48-core 2D-mesh tiled architectures, whose detailed parameters are reported in Table 2. For the 16-core platform the tile is composed of an out-of-order core running at 1GHz, with private L1 cache and a shared L2 cache composed of 16 physically distributed banks, each connected to a different router. The 48-core has the same 4x4 2D-mesh structure of the 16-core, but each tile has three cores. The rationale behind the platform selection is threefold. First of all, 16 and 48 cores are two reasonable architectures to show a realistic scalability of the proposed methodology. Moreover, the choice to add multiple cores per tile instead of increasing the NoC size is motivated by the design choices followed by Intel in the SCC platform, where multiple (two) cores per tile are inserted [21]. Last, the variable number of cores on the same tile allows to validate *CUTBUF* considering different traffic loads, since the 48-core configuration leads to higher traffic.

Different scenarios are simulated using the *GEM5* full system simulator [7], properly enhanced with *CUTBUF*. In particular, both MIBENCH [5] and PARSEC [22] benchmark suites are exploited to provide a comprehensive validation on single- and multi-threaded scenarios. The time to completion, i.e. the time to run the benchmark up to termination, is used as the performance metric in all the analyzed scenarios with real applications.

Table 3 reports different router microarchitecture versions derived from both the baseline and the *CUTBUF* pipeline. In particular, *baseline* identifies baseline NoC router pipeline, with 3 VNETs and 2 VCs per VNET. *ctbf* represents the *CUTBUF* implementation with the minimum number of VCs required by the coherence protocol, i.e. 3 VCs for 3 VNETs in the considered MESI-based directory based protocol. Last, *ctbf*-(3+i) identifies the *CUTBUF* pipeline implementing i additional VCs used to increase performance. Since $i \in (1..3)$, the validation has been carried out considering *CUTBUF* implementing from 3 to 6 VCs, i.e. from the minimum number of VCs required by the coherence protocol up to the same number of VCs implemented in the baseline. Note that the *CUTBUF* proposal mainly aims to reduce both area and power, while keeping almost the same performance of the baseline system.

4.2 Performance Evaluation: Real Applications

This section discusses the performance of the *CUTBUF* architecture compared to the baseline NoC, considering a 16-core 2D-mesh architecture and MIBENCH applications: all the other parameters are kept constant as

TABLE 4: Performance overheads expressed in percentage of *ctbf* and *ctbf-(3+i)* $i \in \{1..3\}$ against the *baseline* for different buffer depths. Only *CUTBUF*-based microarchitectures are reported since the *baseline* NoC is the reference.

(a) Buffer depth 4 flits.					(b) Buffer depth 8 flits.				
Benchmark	ctbf	ctbf(3+1)	ctbf-(3+2)	ctbf-(3+3)	Benchmark	ctbf	ctbf(3+1)	ctbf-(3+2)	ctbf-(3+3)
susan	5.03	0.58	0.14	0.13	susan	3.05	0.11	-0.14	-0.15
qsort	20.07	4.19	2.93	1.89	qsort	11.11	2.27	-0.17	-0.11
sha	0.24	0.07	0.13	0.02	sha	0.19	0.05	0.14	0.05
fft	10.18	0.96	-0.41	-0.60	fft	7.23	0.33	0.11	-0.14
search	10.53	0.03	-0.61	-0.59	search	5.89	-1.21	-0.79	-1.64
dijkstra	4.62	-0.95	-2.17	-0.89	dijkstra	3.56	-1.49	0.41	-2.48
bitcnts	0.12	0.02	0.01	0.01	bitcnts	0.09	0.01	0.01	0.00
basicmath	2.74	0.51	0.41	0.47	basicmath	1.54	0.39	0.06	0.14
AVERAGE	6.69	0.68	0.054	0.055	AVERAGE	4.08	0.058	-0.046	-0.54

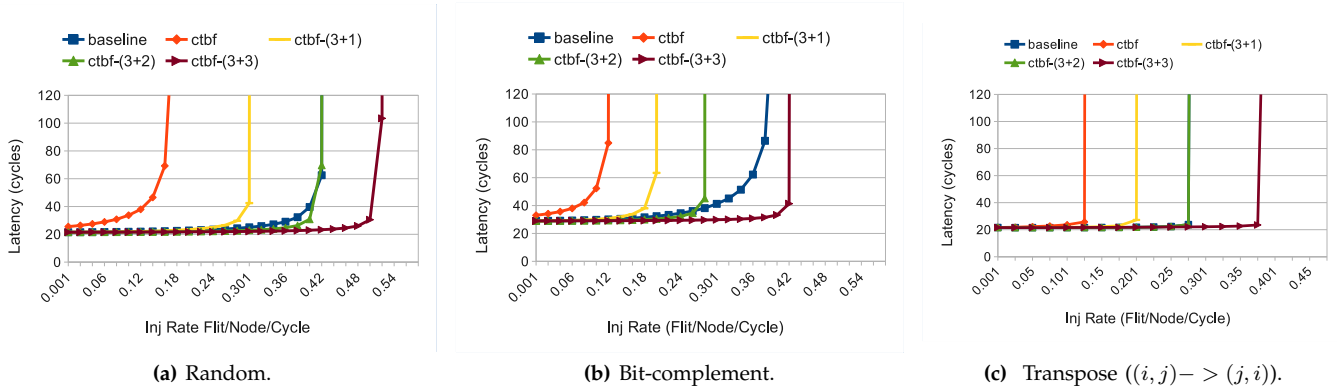


Fig. 7: Baseline and *CUTBUF* pipelines performance considering different synthetic traffic patterns. The baseline pipeline has 3 VNETs and 2 VCs per VNET, while *CUTBUF* is evaluated considering a different number of spare VCs. Moreover, we consider single flit packets, following the validation path of other research papers [4], [3].

described in Table 2. The results are displayed in Table 4a and Table 4b considering a buffer depth of 4 and 8 flits, respectively. It is evident how 4 flits is a reasonable buffer depth to reduce power and area with minimal impact on performance for the majority of the analyzed benchmarks (see Figure 1 for the analysis of a specific benchmark, namely *qsort*). On the other hand, a buffer depth of 8 guarantees no bubble insertion due to round trip time credit in the considered pipeline, thus allowing to clearly evaluate the performance of the proposed solution reducing possible constraining factors.

The two tables share the same format. Different benchmarks listed, whereas each column reports the considered *CUTBUF* architecture as detailed in Table 3. All the results are expressed in percentage with respect to the *baseline* pipeline (that is not explicitly reported) as:

$$overhead_i = (T_{uarch_i} / T_{uarch_{baseline}}) * 100 - 100 \quad (2)$$

where $uarch_i$ is the analyzed microarchitecture, i.e. one of the four microarchitectures reported in Table 3. T_{uarch_i} , $T_{uarch_{baseline}}$ represent the times to completion for the specific architecture to run the benchmark.

CUTBUF without additional VCs (i.e., *ctbf* in the tables) performs worse than the baseline in all the benchmarks considering both 4 and 8 flit buffer depth. For example, *ctbf* shows a performance penalty of 5.0% and 3.0% considering the *susan* benchmark with 4 and 8 flit

buffer depth. However, it is an expected result, since *ctbf* only implements 3 VCs, one VC per VNET.

In general, the increase of the buffer depth for the *CUTBUF* router reduces the gap with the *baseline*, due to a better usage of the additional slots. For example, considering the *qsort* application and *ctbf*, the performance penalty drops from 20.1% to 11.1% moving from 4 to 8 flit buffer depth. This behavior is justified by the *SAF* technique implemented in *CUTBUF* that reduces the contention on VC allocation, since once a packet is granted all the packet traverses the router unless it blocks. If the block is due to buffer fullness, increasing the buffer depth reduces the chances for blocking and then improves the performance of *CUTBUF*-based pipelines.

Although *ctbf* behaves worse than the *baseline*, the performance penalties can be quickly restored by adding spare VCs. In particular, most of the applications show *ctbf-(3+1)* behaving almost as good as the baseline, thus reducing the performance penalty to less than 1%. For example, *fft* shows a performance penalty of 10.18 and 0.96 considering *ctbf* and *ctbf-(3+1)* with 4-flit buffers (Table 4a). This behavior is mainly due to *CUTBUF* ability to dynamically allocate spare VCs to the most loaded VNET.

A last consideration concerns the variability on the application's load. For example, *qsort* suffers the scarcity

of VCs in the *ctbf*, highlighting a 20.0% performance penalty considering 4-flit slot buffers, that is reduced to 11.1% if the buffer depth is 8. At the same time *qsort* greatly benefits from an additional VC, i.e. *ctbf*-(3+1), obtaining a 4.2% and 2.3% degradation for 4 and 8 buffer depth. In this perspective, it is of paramount importance to provide a flexible architecture capable to dynamically allocate spare VCs to the most demanding type of traffic, balancing also power and performance.

4.3 Performance Evaluation: Synthetic Traffic

This section explores the performance of the *CUTBUF* architecture (described in Table 3) considering three synthetic traffic patterns [19].

Considering synthetic traffic, we provide the worst case analysis for the *CUTBUF* proposal from the performance viewpoint. The traffic unbalance at the VNET injection point is totally removed, thus *CUTBUF* can not take advantage over it anymore. Moreover, the use of different injection rates, namely from low traffic up to network saturation, demonstrates the *CUTBUF* effectiveness with different network loads.

Figure 7 reports results for random, bit-complement and transpose $((i, j) - > (j, i))$ synthetic traffics, considering the baseline pipeline and *CUTBUF* with a variable number of spare VCs. As expected, the *ctbf*, i.e. the *CUTBUF* with 3 VCs one per VNET, behaves worse in all the scenarios due to higher contention on the virtual channels. In particular, the saturation point is roughly one third of the saturation point of the baseline pipeline that implements 6 VCs. However, the *ctbf* is intended for power and area savings, thus fitting low traffic scenarios.

Conversely, *CUTBUF* greatly improves its saturation limit by adding spare queues. For example, Figure 7a shows how *ctbf*-(3+2) provides roughly the same saturation threshold of the baseline pipeline even using 5 VCs instead of 6. Note that such results are not only due to the possibility to reuse the VCs for different traffic types, but it is the result of the whole *CUTBUF* methodology.

Moreover, *ctbf*-(3+3) implements the same number of VCs of the baseline pipeline, i.e. 6, while it provides a saturation improvement in between 5% and 12% considering bit-complement and random traffic, respectively. Note that the possibility to dynamically allocate the additional VCs to the most loaded VNET allows quite unbalanced configurations. For example it is possible to have configurations where a single VC is allocated to VNET-0 and VNET-1 while VNET-3 has four allocated VCs. This allows for greater throughput improvement especially when the traffic distribution is not known a priori. Moreover, all the simulations consider equally distributed traffic in all the three VNETs. This is the worst scenario for the presented methodology, nevertheless we are still better than the baseline NoC.

4.4 Scalability Analysis

This section explores the scalability of *ctbf*-(3+1) from the performance viewpoint against the baseline NoC,

TABLE 5: Performance with 16- and 48-core using PARSEC with *sim-small* input set. Results normalized against baseline.

Benchmark	16-core 4x4 2D-mesh (1 core per tile) overhead	48-core 4x4 2D-mesh (3 cores per tile) overhead
	$[(ctbf_t/baseline_t)*100-100]$	$[(ctbf_t/baseline_t)*100-100]$
blackscholes	+0.43	-1.19
dedup	+0.98	-1.16
canneal	-1.30	-2.16
ferret	+2.74	-1.59
rtview	-0.15	+0.31
x264	-0.55	-2.13
bodytrack	-0.18	-1.45
fluidanimate	-2.19	+0.63
streamcluster	+1.12	+1.27
AVERAGE	0.10	-0.83

considering the 16- and 48-core platforms described in Section 4.1. Table 5 details for each simulated PARSEC benchmark the timing overhead of the two considered architectures defined as:

$$overhead_t = (ctbf_t/baseline_t) * 100 - 100 \quad (3)$$

where $ctbf_t$ and $baseline_t$ are the times to complete the benchmark execution considering the *CUTBUF* NoC and the baseline NoC, respectively. Equation (3) measures the performance degradation of *ctbf*-(3+1) with respect to the baseline NoC. In particular, a positive *overhead* means that *ctbf*-(3+1) introduces a performance degradation, while a negative *overhead* means that *ctbf*-(3+1) improves performance with respect to the baseline NoC.

Starting from the described scenarios, two different aspects can be underlined. First, the *ctbf*-(3+1) methodology shows performance overheads within 2% both considering 16- and 48-core architectures even in full system simulation using multi-threaded applications. This is achieved even if *ctbf*-(3+1) implements less buffers with respect to the baseline NoC. Second, *ctbf*-(3+1) provides the same performance degradation within few percentage points, while increasing the number of cores.

4.5 Area, Power and Timing Analysis

Both the *CUTBUF* and the *baseline* NoC routers have been implemented and simulated in RTL Verilog, and then synthesized using Synopsys Design Compiler using the standard 45nm NAND Gate cell library. Synthesis results are collected considering an operating voltage of 1.1 V and a clock frequency of 667 MHz, choosing a router with 5 inputs and 3 VNETs to resemble the architecture simulated using *GEM5*. Area and power results explore three parameters: the buffer depth, the total number of VCs and the flit width. Moreover, Table 6 reports the critical path in picoseconds for each pipeline stage for both the baseline and the *CUTBUF* routers. The *CUTBUF* architecture critical path is in the VA stage due to the additional complexity to arbitrate the implemented buffers considering the VNET reuse optimization, i.e. considering the buffers as a shared resource pool. Moreover, the critical path of the baseline router is in the SA stage. It is worth noticing the baseline router has three VA arbiters, one per VNET, each of them managing a subset of VCs dedicated to

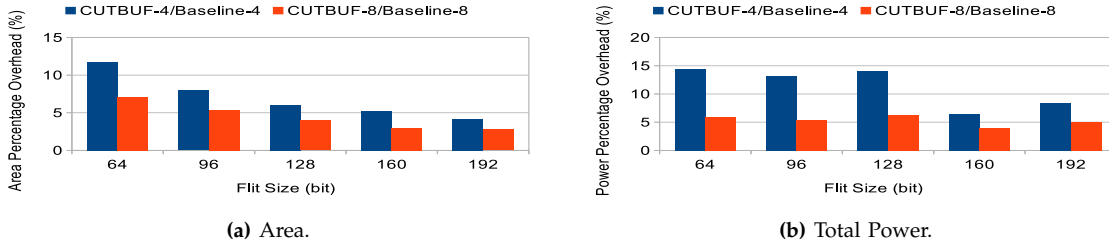


Fig. 8: Area and power exploration for *baseline* and *ctbf-(3+1)* pipelines with 3 VCs and 3 VNETs. Buffer depth of 4 and 8.

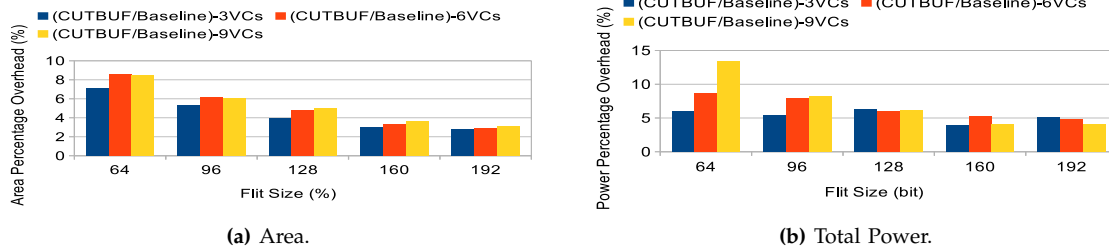


Fig. 9: Area and power exploration for *baseline* and *ctbf-(3+1)* pipelines with 3-6-9 VCs, 3 VNETs. Buffer depth of 8.

TABLE 6: Timing of the *CUTBUF* architecture against the baseline one for each stage in picoseconds. *CUTBUF* and baseline implement 4 and 6 buffers, respectively.

	BW	VA	SA	XBAR
baseline	1054	1117	1333	238
<i>CUTBUF</i>	1022	1486	1194	228

the specific VNET. On the other hand, the SA stage has to arbitrate all the buffers for each input port. The SA in the *CUTBUF* architecture is faster since only 4 buffers are implemented out of 6 while keeping the same performance level as demonstrated in Section 4.4. In this perspective, Table 6 highlights how *CUTBUF* introduces a timing overhead of 11.4% with respect to the baseline ($\frac{CUTBUF_{VA}}{baseline_{SA}} = \frac{1486}{1333}$). However, *CUTBUF* reduces the timing overhead up to 5% exploiting the time stealing technique discussed in [23] for a NoC router. The baseline critical path for the baseline with time stealing becomes $935.5ps (= \frac{1054+1117+1333+238}{4})$ while *CUTBUF* obtains a $982.5ps (= \frac{1022+1486+1194+228}{4})$. Moreover, [23] demonstrated as time stealing introduced a negligible power and area overheads.

Figure 8 reports power and area for both *CUTBUF* and *baseline* considering 3 VCs and 3 VNETs, i.e. 1 per VNET. This is the most constrained configuration, since the MESI coherence protocol uses three VNETs. Results are provided considering 4 and 8 buffer depth, where the X axis reports the flit size. Area and power overheads are reported for *CUTBUF* with respect to the *baseline*. *CUTBUF* introduces both area and power overheads, but they decrease by increasing the buffer flit size, since all the changes that *CUTBUF* introduces are not influenced by the flit width. Moreover, using a deeper buffer, i.e. 8 flits instead of 4, contributes to reduce the area and power overheads. For example, *CUTBUF* has a power overhead of 14.4% considering 64-bit flit size and 4

slots per buffer. However, such overhead decreases to 2.7% when 192-bit flit size and 8 slots per buffer are considered.

Figure 9 depicts the power and area overheads for *CUTBUF* with respect to the *baseline* pipeline considering different amounts of VCs, while keeping fixed the number of VNETs to 3 and a buffer depth of 8 flits per buffer. All the results are obtained comparing *CUTBUF* and the *baseline* with the same number of VCs. Section 4.3 demonstrates that *CUTBUF* can provide the same performance with less VCs. In particular 3, 6 and 9 VCs per input port are evaluated, organized in 1, 2 and 3 VCs per VNET for the *baseline* pipeline, respectively. Moreover, *CUTBUF* has one VCs per VNET for all the cases whereas 3 and 6 spare VCs are used in 6 VCs and 9 VCs configurations, respectively.

Since the major changes for the *CUTBUF* pipeline are in the input buffer, i.e. more complex VCs, increasing the number of VCs rises both power and area overheads. Considering as an example a flit size of 64 bits, Figure 9b reports a power overhead of 5.9%, 8.7% and 13.4% when 3, 6 and 9 VCs are implemented. However, the overhead decreases with the flit size (as well as with buffer depth, not shown in the figure). In particular, the power overhead is lower than 5% with 3, 6 and 9 VCs using 192-bit flit size, with an area overhead which is less than 2.5%. The changes introduced by *CUTBUF* negatively affect both area and power estimates. However, they highly depend on the buffer depth and flit width, thus considering a buffer depth of 8 flits per VC and a flit width of at least 64 bits, the overheads are almost lower than 10%. Moreover, the overheads are loosely coupled with the number of implemented VCs, allowing to introduce multiple spare VCs to increase the perfor-

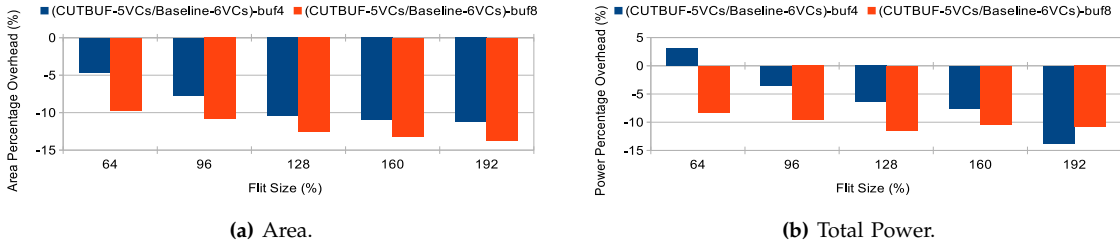


Fig. 10: Area and power for *baseline* with 6 VCs and 3 VNET, i.e. 2 VCs per VNET, and *CUTBUF* with 5 VCs and 3 VNETs.

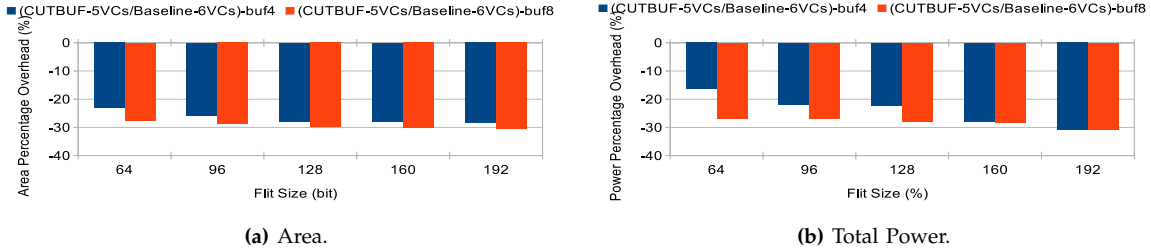


Fig. 11: Area and power for *baseline* with 6 VCs and 3 VNET, i.e. 2 VCs per VNET, and *CUTBUF* with 4 VCs and 3 VNETs.

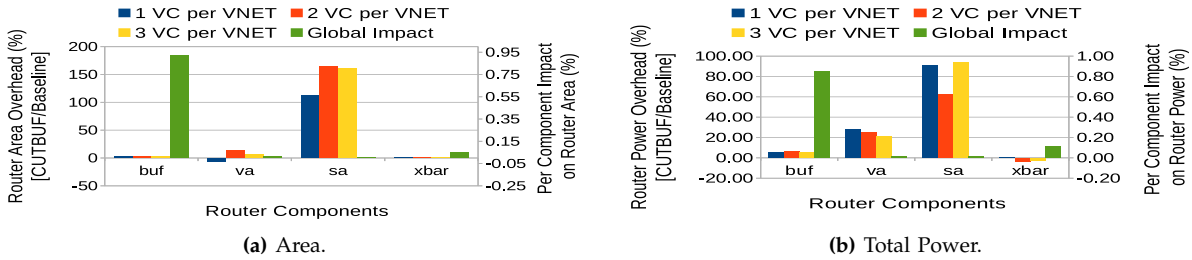


Fig. 12: *CUTBUF* area and power overhead normalized to the *baseline* considering four main router blocks: VC allocation, switch allocator, input buffers and crossbar. 6 VCs, 3 VNET, i.e. 2 VCs per VNET, and the link width is 128 bits with 8 slots per buffer. The impact of each component with respect to the router is considered as the last column of each bar group.

mance without affecting area and power overheads or to support different coherence protocols.

Until now, the analysis compared *CUTBUF* with the *baseline* pipeline considering always the same amount of VCs for each microarchitecture. However, *CUTBUF* focuses on buffer reduction still ensuring limited performance degradation. In this perspective, Figure 10 details area and power results comparing *CUTBUF* with 5 VCs and the *baseline* router with 6 VCs. First, the additional logic to support the *CUTBUF* methodology dominates if the flit size is very small, i.e. 64 bits, thus highlighting a small power overhead for *CUTBUF*, while the reduction to 5 buffers provides a limited area saving. However, starting from a flit size of 96 bits *CUTBUF* always provides lower area and better power figures with respect to the *baseline*. Note that by removing 1 buffer out of 6, the best theoretical saving is $1/6$ (16.67%). In the light of such a bound, Figure 10 shows good scalability for the proposed methodology, since it obtains up to 13.70% and 13.78% respectively, considering a 192-bit flit size.

Figure 11 compares *CUTBUF* with 4 VCs against the

baseline router using 6 VCs. The theoretical throughput obtainable by *CUTBUF* using 4 VCs is limited with respect to the *baseline*, as shown in Section 4.3. However, in almost all the experiments with real benchmarks, the performance overhead of *CUTBUF* using 4 VCs is within 5% (see Section 4.2 and Section 4.4). In this scenario, the reduction of 2 out of 6 buffers allows to save up to 33% of both power and area, provided that buffers dominate the router design. Synthesis data results show an area and power savings of 30.6% and 30.7%, respectively, compared to a *baseline* with 6 VCs.

Figure 12 depicts the area and the power impact introduced by *CUTBUF* for each main block in the router. We consider a router with 128-bit flit width and 8 slots per buffer. Synthesis results with 1, 2 and 3 VCs per VNET are reported. Four main components are considered in the analysis: the crossbar, the input buffers and the virtual channel and switch allocation stages. Additionally, the *Global Impact* bar reports the contribution each component has with respect to the entire router. As expected, the buffers dominate the router area, whereas the area overhead due to *CUTBUF*

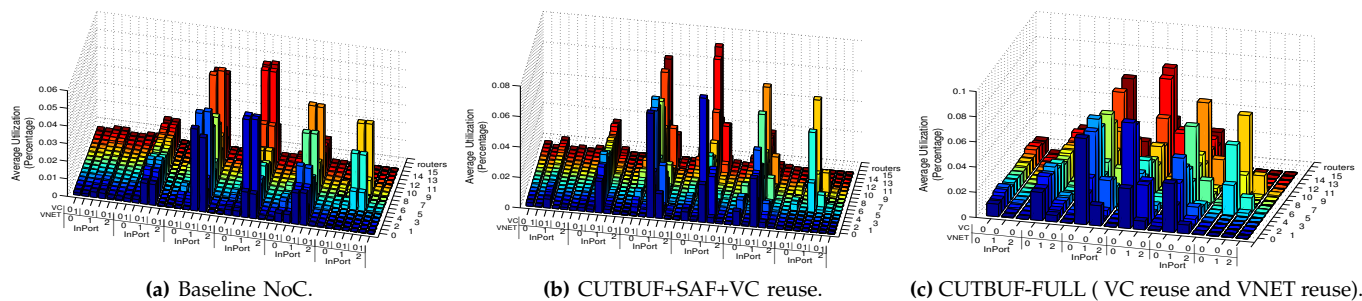


Fig. 13: Averaged traffic distribution across the NoC, i.e. each vc of each router, considering the baseline NoC and the CUTBUF methodology. The CUTBUF is presented in two different versions where the CUTBUF-full is the final proposal of this paper and uses half the number of buffers. Note that VC 2 is seldom used because of the traffic, while we need to implement it to being deadlock free.

for the buffers is limited to 3.3%. Conversely, *CUTBUF* mainly affects the SA stage imposing an area overhead of 112%, 164% and 161% using 1 2 and 3 VCs per VNET. This overhead is due to the size of the SA which only accounts for the 1.4% of the entire router area. Thus, even if the overhead is not negligible on the SA component the total router area is only marginally affected by the *CUTBUF* overhead.

The same considerations are still valid for the power analysis. Again, buffers dominate the power consumption of the router, but *CUTBUF* only adds a modest overhead within 6% for all the examined configurations. Nevertheless, *CUTBUF* greatly impacts the SA power consumption, but a negligible overhead is observed in the whole router, since the SA accounts only affects 2% of the router power.

4.6 Buffer Utilization

This section details how *CUTBUF* reshapes the buffer utilization, thus allowing to physically reduce the number of implemented buffers.

Figure 13 shows the average buffer load for a 16-core 2D-mesh topology, running the *susan* MIBENCH and considering the three different microarchitectures described in Table 3: buffer depth is 4-flit and all the other parameters are kept constant as reported in Table 2. All the results share the same format. The average load for each VC in the NoC is reported on the z axis. All the 16 routers, i.e. from 0 to 15, are reported on the y axis. The x axis reports the virtual channels organized per virtual networks and input ports with an extended label on such x axis. Each VC is reported as well as the corresponding VNET and an input port groups multiple VNETs. For example, Figure 13a reports 2 VCs per VNET, i.e. vc0 and vc1, where each input port groups 3 VNETs, i.e. 0, 1 and 2, thus 6 virtual channels are present for each input port. Note that, input ports are not numbered, since the input port numbering depends on each specific router and its position in the NoC topology. This result provides a qualitative idea on the *CUTBUF* impact. Moreover, Figure 13c reports *ctbf* buffer utilization implementing 1 VC per VNET, thus each input port groups 3 VNETs, but 3 VCs only instead of 6 VCs.

NoC-base, i.e. the results in Figure 13a, shows the different utilization of the VCs in different VNETs. Moreover, due to the flit-switching, all the VCs in the same VNET are almost equally used, thus reducing the chances to remove some buffers. Figure 13b details buffer utilization when both *buffer reuse* and *SAF* are implemented. The utilization of the buffers in the same VNET is now unbalanced, while the traffic of different types is still isolated in a VC subset. Moreover, it is still visible the unbalanced utilization of resources across different VNETs. Figure 13c shows results using the complete *CUTBUF* and implementing 3 VCs per input port instead of 6. All the traffic can use each VC, while the ability to compress the traffic in few VCs opens up the possibility for further power savings, namely using power gating actuators.

5 CONCLUSIONS

Buffers dominate the NoC router area and power consumption and, due to their complex relationship with performance, a design concurrently optimizing these three metrics is not trivial. In addition, the need of VNETs to support coherence protocols leads to an unbalanced buffer utilization.

CUTBUF is a novel router architecture for virtual channeled NoCs that allows to dynamically allocate VCs to VNETs depending on their actual loads. The same buffer can be used to store messages of different types at different points in time, thus allowing resource reuse. The final goal is to reduce the NoC router area and power consumption, while minimizing the negative impact on the performance. *CUTBUF* is validated against synthetic and real applications and a complete RTL Verilog router model has been developed to collect area and power estimates. In a nutshell, *CUTBUF* uses less buffers providing up to 30.6% and 30.7% area and power reduction with an operating frequency decrease of 5% and no more than 5% of performance penalty considering real benchmarks, though most of the time the performance penalty is confined below 2%. On a different perspective, *CUTBUF* is flexible enough to pay less than 5% of area and power overheads to improve the

throughput up to 23.8%, compared to a baseline router with the same number of buffers.

REFERENCES

[1] T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 524–529.

[2] G. Varatkar and R. Marculescu, "Traffic analysis for on-chip networks design of multimedia applications," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 795–800.

[3] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 333–346.

[4] S. Ma, Z. Wang, N. E. Jerger, L. Shen, and N. Xiao, "Novel flow control for fully adaptive routing in cache-coherent nocs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, p. 1, 2013.

[5] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, ser. WWC '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–14.

[6] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09*, April 2009, pp. 423–428.

[7] 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," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[8] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *IEEE/ACM ICCAD*, Nov 2004, pp. 354–361.

[9] 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*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 196–207.

[10] M. Hayenga, N. Jerger, and M. Lipasti, "Scarab: A single cycle adaptive routing and bufferless network," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 244–254.

[11] G. Michelogiannakis, J. Balfour, and W. Dally, "Elastic-buffer flow control for on-chip networks," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, Feb 2009, pp. 151–162.

[12] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 12, pp. 1320–1331, Dec 1993.

[13] L.-S. Peh and W. Dally, "Flit-reservation flow control," in *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on*, 2000, pp. 73–84.

[14] D. Azimi, D. Dai, A. Kumar, A. Mejia, D. Park, S. Saharoy, and A. Vaidya, "Flexible and adaptive on-chip interconnect for tera-scale architectures," in *Intel Techn. J.* 13(4), 2009, pp. 62–79.

[15] C. Concatto, A. Kolgeski, L. Carro, F. Kastensmidt, G. Palermo, and C. Silvano, "Two-levels of adaptive buffer for virtual channel router in nocs," in *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, Oct 2011, pp. 302–307.

[16] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.

[17] I. Seitanidis, A. Psarras, G. Dimitrakopoulos, and C. Nicopoulos, "Elastistore: An elastic buffer architecture for network-on-chip routers," in *Proceedings of the Conference on Design, Automation & Test in Europe*, 2014, pp. 240:1–240:6.

[18] A. Kumar, P. Kundu, A. Singhx, L.-S. Peh, and N. Jha, "A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, Oct 2007, pp. 63–70.

[19] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[20] A. Hansson, K. Goossens, and A. Radulescu, "Avoiding message-dependent deadlock in network-based systems on chip." *VLSI Design*, 2007.

[21] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Eraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 173–183, Jan 2011.

[22] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," ser. PACT. New York, NY, USA: ACM, 2008, pp. 72–81.

[23] A. K. Mishra, A. Yanamandra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "Raft: A router architecture with frequency tuning for on-chip networks," *Journal of Parallel and Distributed Computing*, vol. 71, no. 5, pp. 625 – 640, 2011, networks-on-Chip.



Davide Zoni received the Master in Computer Engineering in 2010 and the Ph.D. in Information Technology in 2014, both from Politecnico di Milano where he holds a Post-Doc position at DEIB - Dipartimento di Elettronica Informazione e Bioingegneria. His research interests include networks-on-chip, computer architecture as well as the application of control theory methodologies for power, performance and reliability optimizations in multi-cores. He received a best paper award in 2012, two HiPEAC Collaboration Grants in 2013 and 2014 and an HiPEAC Industrial Grant in 2015.



José Flich is Associate Professor at UPV where he leads the research activities on NoCs. He published over 100 papers and served in many committees (ISCA, PACT, HPCA, NOCS, ICPP, IPDPS, HiPC, CAC, CASS, ICPADS, ISCC), as program chair (INA-OCMC, CAC) and track co-chair (EUROPAR). He collaborated with several Institutions (Ferrara, Naples, Catania, Jonkoping, USC) and companies (AMD, Intel, Sun). His research focuses on routing, coherency protocols and congestion management within NoCs, with high recognition (RECN and LBDR for on-chip networks). He is member of the Hipec-2 NoE. He is co-editor of the book "Designing Network-on-Chip Architectures in the Nanoscale Era", he coordinated the FP7 NaNoC project and leads the H2020 MANGO project.



William Fornaciari is Associate Professor at Politecnico di Milano - DEIB. He published six books and over 200 papers, collecting 5 best paper awards, one certification of appreciation from IEEE and holds 3 international patents on low power design. Since 1997 he has been involved in 16 EU-funded international projects. In FP7 he has been WP leader for the COMPLEX and CONTREX IP projects, Project Technical Manager of 2PARMA (ranked as success story by the EU) and Project Coordinator of the HARPA project. He cooperated for around 20 years with the Technology Transfer Center of POLIMI and in 2013 he created a startup company. His main research interests cover multi-many core architectures, NoCs, low power design, software power estimation, run time resource management, wireless sensor networks, thermal management, and EDA-based design methodologies. He is member of the HiPEAC NoE.