ElastiStore: Flexible Elastic Buffering for Virtual-Channel-Based Networks on Chip

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

***Abstract*— As multicore systems transition to the many-core realm, the pressure on the interconnection network is substantially elevated. The network on chip (NoC) is expected to undertake the expanding demands of the ever-increasing numbers of processing elements, while its area/power footprint remains severely constrained. Hence, low-cost NoC designs that achieve high-throughput and low-latency operation are imperative for future scalability. While the buffers of the NoC routers are key enablers of high performance, they are also major consumers of area and power. In this paper, we extend elastic buffer (EB) architectures to support multiple virtual channels (VCs), and we derive *ElastiStore*, a novel lightweight EB architecture that minimizes buffering requirements without sacrificing performance. ElastiStore uses just one register per VC and a shared buffer sized large enough to merely cover the round-trip time that appears either on the NoC links or due to the internal pipeline of the NoC routers. The integration of the proposed EB scheme in the NoC router enables the design of efficient architectures, which offer the same performance as baseline VC-based routers, albeit at a significantly lower cost. Cycle-accurate network simulations including both synthetic traffic patterns and real application workloads running in a full-system simulation framework verify the efficacy of the proposed architecture. Moreover, the hardware implementation results using a 45-nm standard-cell library demonstrate ElastiStore's efficiency.**

***Index Terms*— Buffer sharing, elastic buffering (EB), network on chip (NoC), virtual channels (VCs), VLSI.**

I. Introduction

THE network-on-chip (NoC) paradigm is already being adopted in the majority of large systems on chip (SoCs) for simplifying system integration at the IP-assembly functional verification level—all the way down to physical integration—by alleviating physical routing congestion and simplifying timing closure [1]. On-chip networks also improve performance by parallelizing communication, by providing quality-of-service guarantees, and by enabling flexible system partitioning. The NoC designed to support these characteristics needs to implement a resource separation mechanism, typically facilitated by virtual channels (VCs). A VC-based architecture allows a physical channel to be used in a time-multiplexed manner by different traffic flows (i.e., VCs), provided that each flow (VC) owns a separate buffer space [2]. Consequently, VC-based networks enable traffic separation and isolation by assigning different traffic classes to different VCs, and they reduce on-chip physical routing congestion by trading off physical channel width with the number of supported VCs, thereby creating a more layout-flexible SoC architecture [3].

VCs are also instrumental for the correct operation of higher level mechanisms. For instance, protocol-level restrictions in chip multiprocessors (CMPs) employing directory-based cache coherence necessitate the use of VCs. Coherence protocols require isolation between the various message classes to avoid protocol-level deadlocks. Isolation between the message classes is enabled by VCs; each VC (or a group of VCs) is dedicated exclusively to one message class. Therefore, each message class of the coherence protocol is, essentially, given its own dedicated virtual network with its own dedicated buffering space. For example, the MOESI directory-based cache coherence protocol requires at least three virtual networks to prevent protocol-level deadlocks [4].

The NoC needs to be both scalable, in terms of network functionality and performance, and flexible, in terms of physical implementation. This requirement motivates us to unify a VC-based architecture, which favors NoC scalability, with elastic buffering (EB), which eases physical implementation and promises area and power reduction.

Owing to its elastic operation—based on simple ready/valid handshakes—EB is a primitive and simplified form of NoC buffering, which can be easily integrated in a plug-and-play manner at the inputs and outputs of the routers (or inside them) [5]–[7], as well as on the network links to act as a buffered repeater [8]. EB assumes only one form of handshake on each network channel. The handshake cannot distinguish between different flows, thus making the EB operation serial in nature. This feature prevents the interleaving of packets and the isolation of traffic flows, while it complicates deadlock prevention. Due to this limitation, direct support for VCs is abandoned and replaced by multiple physical networks or implemented via complex and nonscalable hybrid EB/VC buffering architectures [9]–[11]. However, the latter techniques remove the basic property of the EBs to act as stitching elements that can be placed seamlessly anywhere in the NoC.

Manuscript received May 2, 2014; revised October 20, 2014; accepted December 4, 2014. The work of I. Seitanidis was supported by the Alexander S. Onassis Foundation for the Ph.D. scholarship.

I. Seitanidis, A. Psarras, and G. Dimitrakopoulos are with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi 671 00, Greece (e-mail: iseitani@ee.duth.gr; apsarra@ee.duth.gr; dimitrak@ee.duth.gr).

K. Chrysanthou and C. Nicopoulos are with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia 1678, Cyprus (e-mail: chrysanthou.kypros@ucy.ac.cy; nicopoulos.chrysostomos@ucy.ac.cy).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2383442

In this paper, we aim to address the aforementioned deficiencies of EB-based designs, by generalizing the operation of EB to support multiple VCs, while at the same time retaining all scalability and composability attributes. The proposed architecture, which we call *ElastiStore*, minimizes the number of buffers per channel close to the absolute minimum of one buffer slot per VC, without sacrificing performance and without introducing any dependencies between VCs, thus ensuring deadlock-free operation. The operation of ElastiStore is generalized to support arbitrary round-trip latencies. The elastic operation and minimum buffering are maintained, while the extra buffering required due to the increased round-trip latency is absorbed via a low-cost shared (across VCs) buffer structure inside ElastiStore.

The scalability of the proposed scheme is demonstrated by the integration of ElastiStore in both single-cycle and pipelined NoC routers that offer the same performance as baseline VC-based routers, albeit at a significantly lower area cost. The experimental results—based on both synthetic traffic patterns and real application workloads running in a full-system simulation framework—validate the effectiveness of the proposed architecture. In addition, the hardware implementation results corroborate our claims pertaining to ElastiStore's efficiency. Overall, our evaluation demonstrates that ElastiStore enables the design of extremely low-cost and highly scalable VC-based NoC architectures that provide equal networking performance as much more expensive (in terms of area/power) state-of-the-art VC-based implementations. ElastiStore is envisioned as an archetypical primitive for future, extremely low-cost NoC router implementations, where the performance and functionality enhancements provided by VCs cannot be sacrificed.

The rest of this paper is organized as follows. Section II describes the operation of elastic flow control for the cases of single and multiple VCs and analyzes the relation between the chosen buffering architecture and the achieved throughput. Section III presents the proposed ElastiStore architecture, while Section IV describes a generalization of the ElastiStore concept that enables efficient operation under arbitrary round-trip times. The integration of ElastiStore buffers within NoC routers is discussed in Section V. The experimental results are presented in Section VI. Pertinent related work is discussed in Section VII, while the conclusion is drawn in Section VIII.

II. Elastic Channels and Buffers

A single-lane elastic channel carries—in parallel to the data wires—two extra control wires (valid and ready), which are required to implement the elastic protocol, as shown in Fig. 1(a). The EBs implement the elastic protocol by replacing any simple data connection with an elastic channel. When an EB can accept an input, it asserts its ready signal upstream; when it has an available output, it asserts the valid signal downstream. When two adjacent EBs both see that the valid and ready signals are both true, they independently know the transfer has occurred, without negotiation or acknowledgement. An example of this handshake is shown in Fig. 1(b).

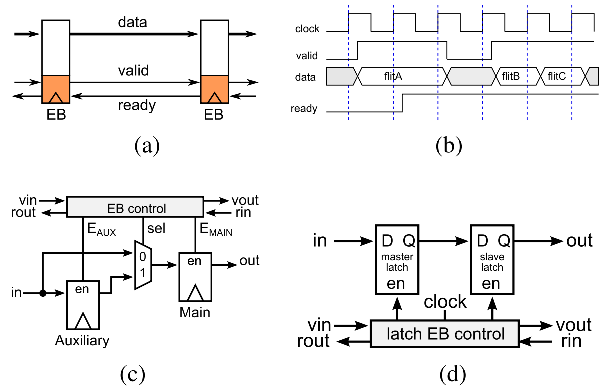


Fig. 1. Fundamentals of the EB protocol. The protocol requires two control wires (valid and ready), which typically run in parallel to the data wires. The data and control wires together comprise a single-lane elastic channel. Any EB architecture derived for edge-triggered flip-flops can also be implemented with latches. (a) Elastic channel. (b) Elastic data flow. (c) Flip-flop-based EB. (d) Latch-based EB.

When the output of a chain of EBs stalls, the stall can only propagate back one stage per cycle. To handle this, all EBs can hold two words: one for the stalled output and one caught when necessary from the previous stage. Such an implementation is shown in Fig. 1(c). By controlling the clock phases accordingly, as shown in [12], the two-slot EB can also be designed using two latches in series, instead of two flip-flops, similar to Fig. 1(d). Following the same methodology, any EB architecture derived for edge-triggered flip-flops can also be implemented with latches.

The same handshake signals can be used for deriving an inelastic flow-control policy. When elasticity is removed and the end of a pipeline of flow-controlled registers is stalled, data movement stops concurrently for all stages of the pipeline; data flow is simply frozen, and, inevitably, some pipeline stages remain empty. On the contrary, an elastic flow-control policy allows all empty stages to be filled with incoming data. In NoCs, the flits of the packet need to flow as close as possible to their final destination before being stalled for any reason. Therefore, the implementation of any flow-control policy in NoCs should be inherently elastic.

Baseline elastic flow control is serial in nature (FIFO-like). Thus, it is not possible to support different isolated flows analogous to a multilane street, or even to allow the interleaving of flits from different lanes on the same elastic channel. This can only be supported by employing individual handshaking interfaces for each supported VC, so that the various VC traffic flows are inherently logically separated and easily guided to their respective parallel buffer slots.

A. VC-Based Elastic Channels

An elastic channel supporting VCs—acting effectively as a multilane street in time-multiplexed manner [2]—consists of a set of data wires that transfer one flit per clock cycle, and as many pairs of control wires valid (i)/ready (i) as the number of VCs. Fig. 2 shows an example of a two-VC elastic channel. Since multiple VCs may be active at the sender, arbitration is employed to select which VC will use the channel. As a result, only one valid (i) signal is asserted per cycle. At the

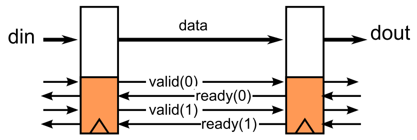


Fig. 2. Example of a two-VC elastic channel. A generic elastic channel supporting VCs consists of a data bus that transfers one flit per clock cycle, and as many pairs of control wires valid (i)/ready (i) as the number of VCs.

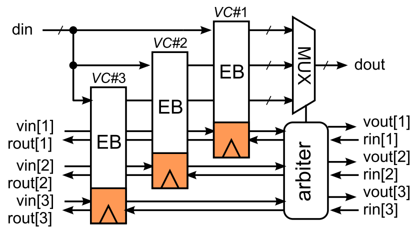


Fig. 3. Baseline EB architecture for three VCs. Such a buffer primitive for VC-based elastic channels can be built by replicating one two-slot EB per VC and by including an arbiter and a multiplexer.

same time, the receiver may be ready to accept flits that can potentially belong to any VC. Therefore, there is no limitation on how many ready (j) signals can be asserted per cycle. The arbiter at the sender should grant only a VC that is ready at the receiver. Therefore, the requests of the active VCs are first qualified by the incoming ready signals.

A buffer primitive for such VC-based elastic channels can be built by replicating one two-slot EB per VC, and by including an arbiter and a multiplexer, following the connections shown in Fig. 3 for the case of three VCs. The arbiter selects which VC will drive the output by checking if it has valid data and if the corresponding VC is ready downstream.

The buffer primitive shown in Fig. 3 is an expensive solution, since the available resources (EBs, in this case) are replicated per VC. When M VCs are active per channel, with $2 \leq M \leq V$ and V representing the maximum number of supported VCs, each VC will receive a throughput of $1/M$. In this case of uniform utilization, each VC will use only one buffer out of the two available per VC, since it will be accessed once every M cycles. The second buffer is used only when a VC stalls. This behavior exhibits a high degree of buffer under utilization, since, most of the time, only one buffer per VC is utilized. Only under extreme congestion, one will see the majority of the second buffers of each VC occupied. However, even under this condition, a single active VC can enjoy full throughput independent of the rest.

B. Pipelined Elastic Links

When the delay of the link exceeds the preferred clock cycle, one needs to segment the link into smaller parts by inserting an appropriate number of pipeline stages. In the case of single-lane channels, the role of the pipeline stages is covered by EBs, which isolate the timing paths (all output signals—data, valid, and ready—are first registered before being propagated in the forward or in the backward direction), while still maintaining link-level flow control, as shown in Fig. 4(a). In the case of multilane channels that support VCs, we can achieve the same result by replacing the

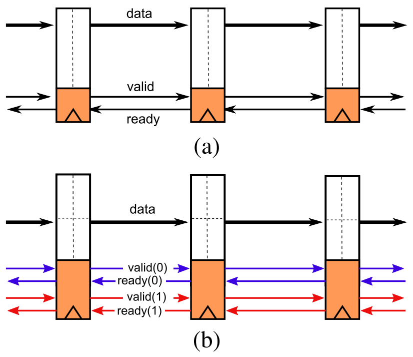


Fig. 4. Pipelined links with (a) EBs that support single-lane operation and (b) EBs for a link that supports multiple VCs (lanes).

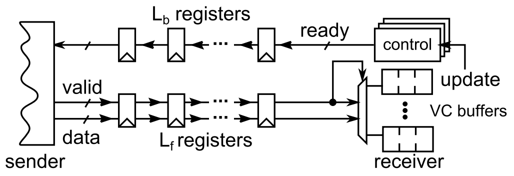


Fig. 5. Abstract model of a pipelined link with multiple VCs and independent ready/valid handshake signals per VC.

EBs with the VC-based EBs of Fig. 3, as shown in Fig. 4(b). Although this approach works correctly and allows for distributed buffer placement, even in the case of VC-based elastic flow control, it is not easily handled in complex SoCs, since the addition of $2V$ registers in arbitrary positions may create layout and physical integration problems.

The scenario of using VC-based EBs at the ends of the link and simple EBs on the link will work, but only after introducing dependencies across VCs, since the flow control information per VC needs to be serialized under a common ready/valid handshake; if one VC stops being ready, all the words on the link should stop, irrespective of the VC they belong to, as done in [11]. Such dependencies ruin the isolation and deadlock-freedom properties of the VCs and require *ad hoc* modifications to the flow control mechanism, even if sufficient private buffer space is allocated per VC.

Alternatively, in the case of VC-based elastic channels, we can rely on simple registers for pipelining the data and the ready/valid handshake signals on the link, as shown in Fig. 5. In this case, the flits cannot stop in the middle of the link, since the pipeline registers do not employ any flow control. Many words may be in flight, since it takes L_f cycles for the signals to propagate in the forward direction and L_b cycles in the backward direction. Therefore, the buffers at the receiver need to be sized appropriately to guarantee lossless and full throughput operation, i.e., more buffers per VC are needed than the two slots per VC allocated in the case of a single-cycle channel (Fig. 3).

First of all, assume that only one VC, i.e., the ith one, is active for a period of time and the remaining VCs do not send or receive any data. When the buffer of the ith VC is empty, it asserts the ready (i) signal. The sender will observe that ready (i) is asserted after L_b cycles and immediately starts to send new data to that VC. The first flit will arrive at the receiver after $L_f + L_b$ cycles. This is the first time that the receiver can react by possibly deasserting the ready (i) signal. If this is done, i.e., ready (i) $= 0$, then under the worst case assumption, the receiver should be able to accept the $L_f - 1$ flits that are already on the link and the L_b flits that may arrive in the next cycles (the sender will be notified to stop with a delay of L_b cycles). Thus, when the ith VC stalls, it should have at least $L_f + L_b$ empty buffers to ensure lossless operation. Actually, the minimum number of buffers for the ith VC reduces to $L_f + L_b - 1$, if we assume that the sender stops transmission in the same cycle it observes that ready (i) $= 0$.

Thus, a channel with V VCs and a round-trip time of $L_f + L_b$ needs at least $V(L_f + L_b - 1)$ slots. When many VCs are active on the channel, their flits would be interleaved and the probability that all $L_f + L_b - 1$ flits belong to the same VC is small. However, the worst case condition calls for providing as much buffer space to each VC as needed to prevent the dropping of any flit, independent of the traffic conditions on the remaining VCs.

Unfortunately, giving the minimum number of buffers to each VC has some throughput limitations. Assume that the ith VC has occupied all its buffer slots at the receiver and starts draining the stored flits downstream at a rate of one flit per cycle. After $L_f + L_b - 1$ cycles, the buffer will be empty (no more flits to drain) and the ready (i) signal will be asserted, causing the fist new flit to arrive $L_f + L_b - 1$ cycles later [the ready (i) signal is asserted in the same cycle that the last flit is drained]. Therefore, in a time frame of $2(L_f + L_b - 1)$ cycles, the receiver was able to drain only $L_f + L_b - 1$ flits, which translates to 50% throughput. Thus, a single active VC can enjoy 100% throughput when it has $2(L_f + L_b - 1)$ buffers and is ready when the number of empty slots is at least $L_f + L_b - 1$. The baseline VC-based EB of Fig. 3 employed in single-cycle links ($L_f = L_b = 1$) is a subcase of the general pipelined link and achieves 100% throughput of lossless operation using two buffers per VC.

III. ElastiStore Architecture

In the case of single-cycle links with $L_f = L_b = 1$, the baseline VC-based EB—which allocates two slots per VC—allows each VC to stop and resume transmission at a full rate independently from the rest. This feature is indeed useful in the case of traffic originating only from a single VC, where any extra cycles spent per link will severely increase the overall latency of the packet. However, in the case of multiple active VCs, whereby each one receives only a portion of the overall throughput ($1/M$ in the case of M active VCs), allocating more than one buffer slot per VC is an overkill.

A. Operational Details

In this paper, we take advantage of the abovementioned realization and provide a very cost-efficient design. The proposed buffering architecture, called *ElastiStore*, utilizes only $V + 1$ buffers for V VCs. Each VC owns a single buffer, which is enough in the case of uniform utilization, where each VC receives a throughput of $1/M$, with $2 \leq M \leq V$. Furthermore, when a single VC uses the channel without any other VC being active, i.e., $M = 1$, it receives 100% throughput, and, in the case of a stall, it may use the additional buffer available in ElastiStore. This additional buffer is shared dynamically by all VCs, although only one VC can have it in each clock cycle. However, when all VCs, except one, are blocked, and the shared buffer is utilized by a blocked VC, then the only active VC will get 50% of the throughput, since it effectively sees only one buffer available per channel. The baseline and expensive VC-based EB of Fig. 3, which allocates two buffers to each VC, would allow this active VC to enjoy full channel utilization.

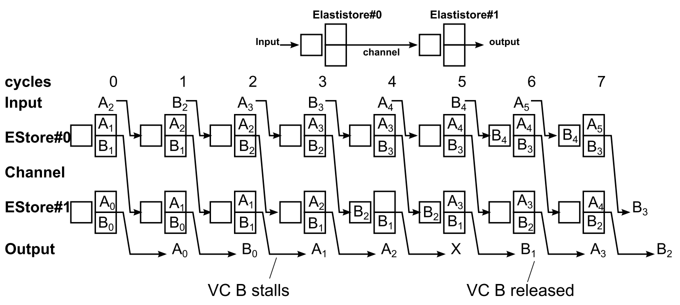


Fig. 6. Example of flit flow on an elastic channel that supports two VCs and utilizes ElastiStores.

Fig. 6 shows an example of flit flow in a pipeline of two ElastiStores connected in series, with each ElastiStore hosting two VCs. In cycle 0, both ElastiStores contain flits from the previous cycles; A_0 and B_0 reside in ElastiStore#1, and A_1 and B_1 reside in ElastiStore#0. In cycle 1, A_0 is moved to the output and its position is filled with flit A_1. In parallel, the flit A_2 waiting at the input of the pipeline in cycle 0 is written into ElastiStore#0 in cycle 1. In cycle 2, the output of the pipeline is driven by flit B_0, and, concurrently, the empty positions are filled with flits moving between ElastiStores or coming from the input. In this way, VCs A and B receive 1/2 of the throughput per channel ($M = 2$), and, at each step, they utilize one buffer slot. In those cycles, the shared auxiliary registers of the two ElastiStores are not utilized. The shared buffers are used between cycles 4 and 7 to accommodate the stalled words of VC B. In those cycles, VC A—which is not blocked—continues to deliver its words to the output of the channel. The stall of VC B is hidden from the input of the pipeline, since any incoming flit of VC B is accommodated in the shared buffer slots of both ElastiStore units. If the stall lasted longer, flits from VC B would not enter the pipeline after cycle 7.

ElastiStore offers a reasonable tradeoff, since it saves $V - 1$ buffer slots per elastic VC buffer, as compared with the baseline VC-based EB of Fig. 3, and limits throughput only under heavy congestion that blocks all the VCs except one. In the case of light traffic, a single active VC receives full throughput without any limitation. In addition, the static allocation of a single buffer to each VC and the completely independent flow control per VC guarantees forward progress for all VCs and avoids possible protocol-level deadlocks.

B. Hardware Implementation

ElastiStore can be designed using the datapath shown in Fig. 7, which consists of a single register

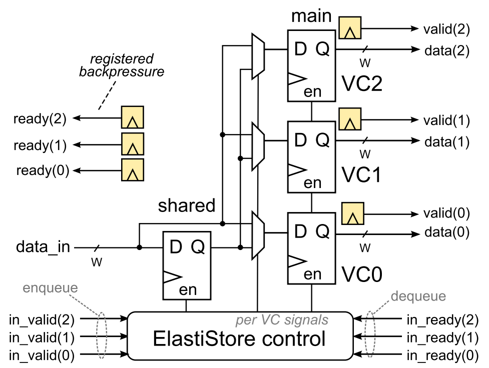


Fig. 7. Datapath and organization of an ElastiStore primitive for three VCs. ElastiStore consists of a single register per VC (main registers) along with a shared register that is dynamically shared by all VCs.

per VC (main registers) along with a shared register that is dynamically shared by all VCs. The select signals of the bypass multiplexers, the load enable signals of the registers, and the interface ready/valid signals are all connected to ElastiStore control.

When a new flit that belongs to the ith VC arrives at the input of ElastiStore, it may be placed either in the main register of the corresponding VC or in the shared register. If the main register of the ith VC is empty or becomes empty in the same cycle as the main register of the baseline EB of Fig. 1(c), the flit will occupy this position. If the main register is full, then the incoming flit will move to the shared buffer. Concurrently, once the shared buffer is utilized, all the VCs that have their main register full will stop being ready to accept new data, while those with an empty main register remain unaffected. In ElastiStore, any VC is ready to accept a new flit if at least one of the two registers is empty: either the main register corresponding to said VC or the shared one.

On the read side of ElastiStore, data are only dequeued from the main registers. The shared register acts only as an auxiliary storage and does not participate in any arbitration or allocation decisions that select which VC should be dequeued. According to this design principle, when the main register of a VC dequeues a new flit and the shared buffer is occupied by the same VC, the main register of this VC should be refilled by the data stored in the shared buffer in the same cycle. The shared buffer cannot receive a new word in the same cycle, since its readiness—which releases all VCs that have their main register full—will appear on the upstream channel in the next clock cycle. The automatic data movement from the shared to main buffer avoids any bubbles in the flow of flits of the same VC and achieves a maximum throughput.

IV. Generalized ElastiStore Architecture: Supporting Arbitrary Round-Trip Times

ElastiStore represents the simplest form of a VC-based buffer structure that can be used either as a distributed buffering alternative or at the inputs of VC-based routers, as will be shown in Section V. When the flow-control signals between two ElastiStore units take more cycles to propagate in the forward or in the backward direction, the baseline ElastiStore architecture needs to be augmented with more shared buffer positions to absorb the in-flight flits.

As in the case of single-cycle links, in pipelined links with L_f and L_b of forward and backward latencies, respectively, we minimize buffering by employing sharing and by exploiting the fact that only a single VC (dynamically selected) can enjoy 100% throughput when it is the only active VC in the system.

Instead of having $2(L_f + L_b - 1)$ buffer slots for each VC, we dedicate $L_f + L_b - 1$ slots per VC needed for safe operation (called the main buffers) and $L_f + L_b - 1$ more, which can be dynamically shared by all VCs (called the shared buffer). Any VC is ready, as long as there are $L_f + L_b - 1$ empty slots either in its main register alone or accounting for the free space in the shared buffer as well. Therefore, a single active VC can enjoy 100% throughput, while, in the case where the shared buffer is full, every active VC cannot get more than 50% of throughput (it can receive/send $L_f + L_b - 1$ flits at most every $2(L_f + L_b - 1)$ cycles); when many VCs are active, the throughput per VC is always much lower than 50%.

Under high utilization, the channel is already shared by many VCs, and achieving high throughput per independent VC does not give much benefit, unless it is the only active VC. Therefore, our optimal goal is to offer full throughput to a single active VC using just one register per VC of private buffering, as well as $L_f + L_b - 1$ more positions shared by all VCs. If we try to achieve this goal with the current flow-control semantics, dependencies across VCs may appear that can possibly lead to a deadlock. Assume, for example, that the ith VC uses its main buffer (one register) and at least one slot from the shared buffer, leaving less than $L_f + L_b - 1$ free slots in the shared buffer. Then, every other VC must deassert its ready signal, even if its main register is empty, since the available free slots for each VC are less than $L_f + L_b - 1$, which are needed to guarantee safe operation per VC. Under this scenario, the traffic on one VC is allowed to block the traffic on another VC, which removes the needed isolation property across VCs.

To satisfy our minimum buffering requirements, we need to adopt a different flow control mechanism, which does not have the limitations of independent ready/valid handshake per VC. Such transformation is only needed when simple pipeline registers that do not participate in the flow control mechanism are added between any two ElastiStores. This is most often done inside routers, as we will show in Section V.

A. Credit-Based Operation

To achieve our goal of minimum buffering, we should change the flow control mechanism and allow the sender to explicitly store the condition of each downstream VC, as done by credit-based flow control. The sender keeps a free-slot counter, or else called a credit counter, for each downstream VC and a counter for the shared buffer that counts the available shared buffer slots. A VC is eligible to send a new flit when there is at least one free position (either at the main register or the shared buffer). Once the flit is sent from the ith VC, it decrements the credit counter of the ith VC. If the credit counter of the ith VC was already equal to or smaller than zero, this means that the flit consumed a free slot of the shared buffer and the counter of the shared buffer is also decremented.

Since the state of each VC is kept at the sender, the receiver only needs to send backward a credit-update signal, including a VC ID, which indexes the VC that has one more available credit for the next cycle. On a credit update that refers to the jth VC, the corresponding credit counter is increased. If the credit counter is still smaller than zero, this means that this update refers to the shared buffer. Thus, the credit counter of the shared buffer is also increased. Please note that even if there is a separate credit counter for the shared buffer, the forward valid signals and the credit updates refer only to the VCs of the channel and no separate flow control information is needed for the shared buffer.

In this case, safe operation is guaranteed even if there is only one empty slot per VC (main register), but with a very limited throughput due to the increased round-trip time; no flit can be in flight if it has not consumed a credit beforehand. A single active VC can utilize both its main register and all the positions of the shared buffer and achieve 100% throughput, by effectively allowing this VC to use $L_f + L_b$ buffers in total. With credits, once a credit update is sent backward for a VC, it means that a new flit will arrive for this VC after $L_f + L_b - 1$ cycles. Therefore, offering to a single VC $L_f + L_b - 1$ buffer means that at the time the last flit is drained from the VC, the first new flit will arrive, thus leaving no gaps in the transmission and offering full throughput. Note again that any VC is still eligible to accept a new flit in its private (main) buffer, irrespective of the state of the shared buffer, thus completely avoiding deadlock conditions.

By adopting credit-based flow control, the generalized ElastiStore minimizes the private buffer space per VC, thus achieving its main goal of minimizing the buffer space to just one register per VC, and some extra shared space to fully cover the round-trip time for one VC.

The use of credits, or ready (i)/valid (i) handshake signals, for the flow control of different VCs is similar, in the sense that they both count either implicitly or explicitly the available number of buffer slots for each VC. For the ready (i)/valid (i) interfaces, this counting is done at the receiver, and the sender is notified via the delayed ready signals. On the contrary, in credit-based flow control, buffer availability is checked locally at the sender, without any notification delay. This difference in the notification delay causes the two flow-control policies to behave differently, in terms of minimum buffering requirements for achieving full throughput operation.

B. Hardware Implementation

The implementation of the generalized ElastiStore (which is able to support arbitrary round-trip times) is based on the three basic operational principles/rules that characterize all ElastiStore designs and differentiate them from other state-of-the-art buffer implementations.

1) Each VC has only one slot of private buffer space implemented via a main register per VC, while the remaining buffer space is shared and used together with the one main register per VC to cover the round-trip time of the channel that ElastiStore is connected to (this translates to one shared buffer for single-cycle links and $L_f + L_b - 1$ buffer slots for a link with L_f forward and L_b backward latencies, respectively). This configuration offers the minimum possible buffering, with the constraint that at most one VC can receive full throughput. This is not a restrictive choice, since a VC will get full throughput only when it is the only active VC. In all other cases, each VC will receive an equal share of the available throughput.
2) Any allocation decision regarding which VC should dequeue a flit from the ElastiStore structure is taken based only on the status of the main registers. In this way, request generation can begin without any overhead associated with checking the head-of-line flits of many VC queues hosted in a shared buffer space.
3) When a main register sends a flit downstream and gets empty, it is refilled in the same cycle, either with a flit possibly present in the shared buffer or directly from the input, assuming the new flit belongs to the same VC. This design principle completes the previous one and avoids idle cycles in the data flow. With this type of refill, ElastiStore actually mimics the simple EB of Fig. 1(c), which refills (in the same cycle) the main register using the data of the auxiliary register or the data of the input, when the main register gets empty, to offer full throughput of data transfer.

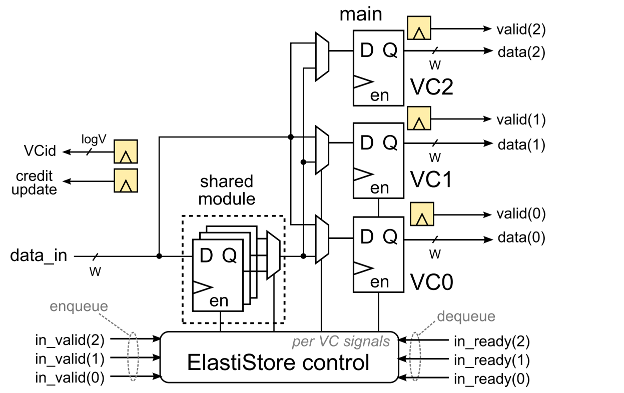


Fig. 8. Organization of the generalized ElastiStore. The shared buffer consists of as many buffer slots as required to cover the round-trip time of the flow-control signals.

The introduced design principles and the use of negative credits—that simplify the process of credit identification, i.e., which credits belong to the main registers and which ones belong to the shared buffer space—offer an overall simplified buffering architecture.

The datapath that implements the generalized ElastiStore is shown in Fig. 8. The multiplexers at the input of the main registers allow new data to come directly from the ElastiStore's input or the shared buffer. If the main registers cannot accommodate an incoming flit, a shared slot is allocated, where the flit is stored. As soon as the main register becomes available again, the flit is retrieved from the shared buffer and moves to the corresponding main register.

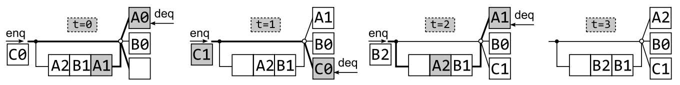


Fig. 9. Example of the interaction between the shared buffer and the main VC registers in the generalized ElastiStore architecture. Every time a VC dequeues a flit from its main register, it should check the shared buffer for another flit that belongs to the same VC.

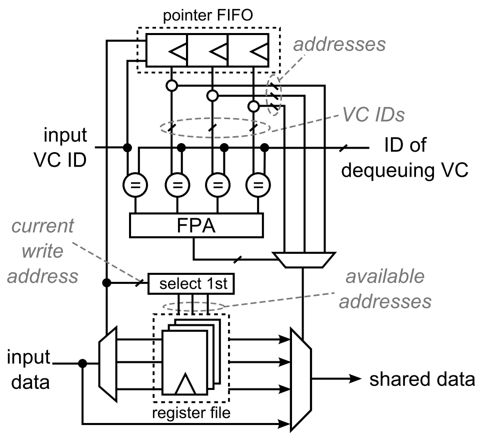


Fig. 10. Block diagram of the *shared buffer* of the ElastiStore architecture. The shared buffer consists of two main storage modules. The first one (top part) is a shift-register-based FIFO that stores (in each slot) the VC ID of a flit and a pointer. The second storage module (bottom part) is the register file that stores the actual flit contents.

Every time a VC dequeues a flit from its main register, it should check the shared buffer for another flit that belongs to the same VC. Fig. 9 demonstrates the interaction between the shared buffer and the main VC registers through a simple example. In cycle 0, VC A owns two shared slots and dequeues a flit from its main register. The empty main register should be refilled in the same cycle to avoid any bubbles in the flit flow control. Therefore, VC A initiates a search on the shared contents to find the flits that match VC A and locate the oldest (the one that came first). The refill of the main register of VC A is completed in cycle 1. Then, in cycle 2, the same procedure is followed, effectively loading the main register of VC A with a new flit. The main register of a VC does not necessarily get new data from the shared buffer, but it can be loaded directly from the input, as done for VC C in cycle 1.

The shared buffer within ElastiStore should preserve an FIFO order, not in terms of the whole buffer, but separately for the flits of each VC. In the example of Fig. 9, the flits' order of arrival is preserved by shifting them forward every time a slot in the shared buffer is emptied. However, this is simply an abstract representation of what is really happening: flits are not physically shifted, but, instead, their pointers change. In other words, the FIFO order is maintained through the pointer logic. The block diagram of the shared buffer of the ElastiStore architecture is shown in Fig. 10. It consists of two main storage modules. The first one (top part of Fig. 10) is a shift-register-based FIFO that stores (in each slot) the VC ID of a flit and a pointer. The second storage module (bottom part of Fig. 10) is the register file that stores the actual flit contents. When a flit is pushed to the shared buffer, the actual contents are written in the register file, while the write address and the VC ID of the incoming flit are pushed into the first empty slot of the shift register (the pointer FIFO in Fig. 10).

The proposed shared buffer does not participate in router allocation, as imposed by the second design rule, and thus, it is redundant to be able to see the head-of-line flits of all hosted VC queues (this option would need at least one read pointer for each active VC queue). On a read (dequeue operation), the shared buffer should be able to read out the first appearance of the dequeuing VC ID, to refill its main register. This is done by comparing the IDs stored in the pointer FIFO or the input, with the ID of the dequeuing VC, and selecting—using a fixed-priority arbiter—the first one that matches. Then, the pointer stored in the pointer FIFO is used to retrieve the actual flit from the register file, while the information of the dequeued flit is removed from the pointer FIFO, thereby causing all subsequent pointers to shift forward. At the same time, the address of the dequeued flit is marked as available in the register file and can be reallocated to any VC.

V. Integration of ElastiStore in NoC Routers

ElastiStore can be considered as the first proposal of either: 1) a primitive EB structure that supports VCs for single-cycle channels (Fig. 7) or 2) as a distributed EB architecture that supports VCs in pipelined links [Fig. 4(b)] utilizing a low-cost shared buffer structure built on top of the design principles introduced in Section IV-B. In both cases, ElastiStores can be placed seamlessly and in a plug-and-play manner everywhere within the NoC.

When a packet arrives in a router, it needs to find its output destination port via routing computation (RC). The output port can be precomputed in the previous router, using look-ahead RC (LRC) [13]. Each packet then has to choose a VC at the input of the next router, before leaving the current router (known as an output VC). Matching input VCs to output VCs is performed by the VC allocator (VA). Allowing packets to change VC in flight can be employed when the routing algorithm does not impose any VC restrictions (e.g., *XY* routing does not even require the presence of VCs). However, if the routing algorithm and/or the upper layer protocol (e.g., cache coherence) place specific restrictions on the use of VCs, then arbitrary in-flight VC changes are prohibited, because they may lead to deadlocks. In the presence of VC restrictions, the VA will enforce all rules during VC allocation to ensure deadlock freedom. Such VC restrictions are orthogonal to the operation of ElastiStore.

The flits that own an output VC arbitrate for accessing their output port. If a flit wins this stage—called switch allocation (SA) and organized in local and global arbitration steps, SA1 and SA2—it will traverse the crossbar [switch traversal (ST)], and then it will move to the output link [link traversal (LT)] toward the next router [14], [15].

ElastiStore can be integrated at the inputs and at the outputs of a router, as shown in Fig. 11. When ElastiStores are used in both inputs and outputs, the output VCs refer to the buffers of the output ElastiStore and not to the VC buffers at the input of the next router. The same also holds for flow control information that reflects the buffer status of the output

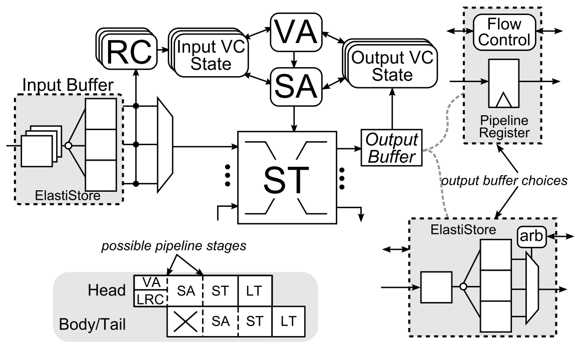


Fig. 11. Integration of ElastiStore in NoC routers. ElastiStore modules can be integrated at the inputs and at the outputs of a router. In general, ElastiStores can be placed seamlessly and in a plug-and-play manner everywhere within the NoC.

ElastiStore of the same router. A flit is ready to move to the output ElastiStore when the corresponding VC of the output ElastiStore is ready. The placement of an output ElastiStore actually breaks the flow-control cycle (which determines the round-trip time) between two neighboring routers in two parts: 1) the intrarouter part, which involves the forward and backward latencies inside the router and 2) the interrouter part, which only involves the flow-control latencies of the link. The flow control on the links does not allow packets to change VC, and its operation needs only an arbiter and a multiplexer for selecting a flit to send to the next router.

In single-cycle routers, where one cycle is spent inside the router and one on the link, the intrarouter and interrouter round-trip times are equal to two cycles. This configuration enables the use of baseline ElastiStores (shown in Fig. 7) at the inputs and the outputs, which have only one register per VC and one shared slot among all VCs. Single-cycle implementations, to avoid the serial operation of VA and SA, are often implemented with more aggressive allocation strategies, such as combined allocation [16] or speculative allocation [17], [18], which enable faster implementations and offer almost the same network performance.

High clock frequencies can be equivalently achieved by employing router pipelining. For example, a two-stage pipelined router would separate (L)RC-VA from SA-ST, thus isolating the critical path to the second pipeline stage. In this configuration, however, the intrarouter round-trip time increases to three cycles, following the inevitable increase of the forward latency by one cycle. In this case, assuming again that ElastiStores are placed both at the inputs and at the outputs, the input ElastiStore should be transformed to a generalized ElastiStore unit that consists of one register per VC plus a two-slot shared buffer (Fig. 8). When a VC is selected by SA, its main register is dequeued and a refill possibly occurs from the shared buffer. The refill data can be prepared beforehand, just after SA1, thus overlapping the search in the shared buffer with SA2, while actual dequeue (pointer movement) happens only if SA2 is also successful for the same VC. Since the round-trip time across the link remains the same, the output ElastiStore can still be the simplest one.

Baseline routers utilize a simple pipeline register—instead of the output ElastiStores—that isolates the interrouter timing paths from LT. Therefore, the round-trip times expand inevitably between the inputs of two neighboring routers, which are the only flow-controlled buffering points. In single-cycle baseline routers, this translates to three buffers per VC to cover the round-trip time, while a pipelined router with two stages increments the credit round-trip latency by one more cycle, thus needing a minimum of four buffers per VC. This analysis assumes that credit updates across routers need at least one cycle to propagate. This extra cycle can be removed if flow-control information is transferred across routers via direct combinational paths. However, this actually limits the benefits of pipelining, and the increased link delay directly affects the speed of the router.

Similarly, when ElastiStores are used only as replacements of the input buffers (the output of the router has only a pipeline register), the router should be designed using generalized ElastiStore units that support the increased round-trip times in a cost-effective manner. A single-cycle implementation with three cycles of round-trip delay would need a generalized ElastiStore with one register per VC and two buffer slots in the shared module, while a two-stage pipelined implementation would just add one more position to the shared buffer, to absorb the one cycle increase in the round-trip time.

The buffering requirements of a baseline router with N input and output ports, a router with ElastiStores at both inputs and outputs (ES-IO), and a router that employs ElastiStore only at the inputs (ES-I) for one-stage (single-cycle) and two-stage pipelined organizations are summarized in Table I.

TABLE I
MINIMUM BUFFERING REQUIREMENTS FOR BASELINE AND ELASTISTORE-BASED ROUTER ORGANIZATIONS

	1-stage	2-stage
Baseline	$(3V+1)N$	$(4V+1)N$
ES-IO	$(2V+2)N$	$(2V+3)N$
ES-I	$(V+3)N$	$(V+4)N$

In every case, the proposed designs save considerable amount of buffers, which directly translate to significant area savings without any network performance loss, as it will be shown in the next section. When using ElastiStore only at the inputs, one may reach the absolute minimum of VC buffering of one register per VC (needed also for deadlock freedom), plus any additional buffer slots needed for covering the round-trip time and offering the privilege to a single active VC to achieve 100% throughput. The use of ElastiStores at the output of the routers steals some time from LT, due to the arbitration and multiplexing operation. However, the extra delay added to the delay of the link will affect the NoC clock frequency only in the case of very long wires. In such cases, simple pipeline registers can still be used at the outputs, as shown in Fig. 11.

VI. EVALUATION

In this section, we compare ElastiStore-based routers with conventional VC-based routers, both in terms of hardware complexity and network performance, which includes synthetic traffic patterns (Section VI-B), as well as real application workloads (Section VI-C).

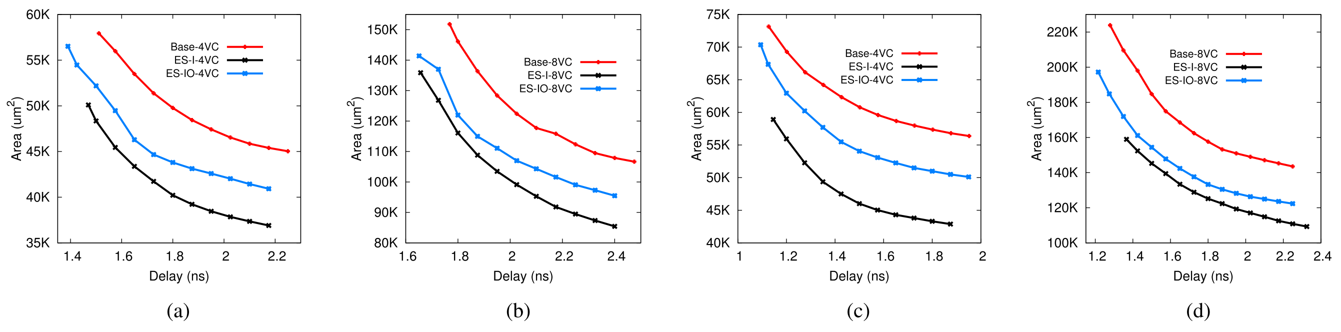


Fig. 12. Hardware implementation results of various designs with (a) one-stage four VCs, (b) one-stage eight VCs, (c) two-stage four VCs, and (d) two-stage eight VCs router designs, using an industrial low-power 45-nm standard-cell library.

A. Hardware Implementation

The routers under comparison (using LRC) were implemented in VHDL, mapped (synthesized) to an industrial low-power 45-nm standard-cell library under worst case conditions (0.8 V, 125 °C), and placed and routed using the Cadence digital implementation flow. The generic router models have been configured to five input–output ports, as needed by a 2-D mesh network, and to four and eight VCs per port, while the flit width was set to 64 bits. The area/delay curves, shown in Fig. 12, were obtained for all designs after constraining appropriately the logic-synthesis and back-end tools and assuming that each output is loaded with a wire of 2 mm.

The routers under comparison include baseline routers with one-stage and two-stage pipelined organizations, as well as ElastiStore-based routers that include buffers both at the inputs and the outputs (ES-IO) and only at the inputs (ES-I). Baseline VC routers can be built with shallow or deep buffers per VC. It is critical, however, for each VC to contain as many buffers as needed to cover the credit round-trip latency. Therefore, for all routers, we assume the buffers shown in Table I. For all the single-cycle routers, we employed the combined allocation approach presented in [16], which offers the same network performance as traditional allocation organizations, but with significantly better achievable clock frequency. On the contrary, the two-stage pipelined routers follow the normal allocation strategy, where SA begins only after VA has been completed for an arriving packet.

In all cases, the ElastiStore-based routers offer significant area savings, up to 18% and 24% for four and eight VCs, respectively, without any delay overhead. This behavior is the result of the reduced number of buffer slots required by ElastiStore and the overall simplicity of its control logic k (<10% of the total ElastiStore area). The latter is a consequence of the three newly introduced design principles/rules and the simplified credit-handling protocol. The ES-I configuration, as expected, is the most area-efficient solution. The ES-IO setup, which completely isolates the interrouter flow control mechanism from the intrarouter one, achieves even faster designs, since the readiness of each VC is directly provided by the ready/valid handshake signals, while still saving area relative to the baseline design. Note that the delay numbers reported correspond to operation at 0.8 V. At this low voltage, the clock frequency of even ultrafast three-stage commercial routers is below or marginally pass 1 GHz [19], [20].

TABLE II
ENERGY PER CYCLE (IN PICOJOULES) REQUIRED FOR BASELINE AND ELASTISTORE-BASED ROUTERS HAVING FOUR AND EIGHT VCS AND OPERATING IN SINGLE-CYCLE OR TWO-STAGE PIPELINED CONFIGURATIONS

Buffers	4 VCs		8 VCs	
	1-stage	2-stage	1-stage	2-stage
ES-I	67	69	108	122
Base	87	91	153	167

The hardware complexity analysis is completed by reporting the energy behavior of the routers under comparison. Energy (or area) comparisons are meaningful when the compared circuits are optimized for the same delay target. Therefore, based on the delay profile reported in Fig. 12 for single-cycle and two-stage pipelined solutions, we select the designs that correspond to a delay of 1.5 and 1.8 ns for the case of four and eight VCs, respectively. The energy consumed for each case is reported in Table II. The energy analysis is reported after considering all layout parasitics, while the switching activity has been computed using delay-accurate simulation of the derived logic-level netlists. The evaluated routers are all driven by the same arriving packet sequence, which mimics uniform random (UR) traffic of one-flit and five-flit packets at an injection rate of 0.2 flits/cycle. The traffic characteristics determine the header contents of each packet, while the data contents—i.e., the payload—of each packet (mostly for body and tail flits) are produced using a UR number generator. In all cases, the energy required to drive the output links is also included.

ElastiStore-based routers require the least energy per cycle, due to the significant energy cost reduction of the buffers, which are responsible—together with the network links—for the majority of the energy required in each data transfer. The energy reductions due to ElastiStore surpass its area savings and lead to 23% and 28% more energy-efficient routers for four and eight VCs, respectively.

B. Network Performance

Network performance comparisons were performed using a cycle-accurate SystemC network simulator that models all microarchitectural components of an NoC router, assuming an 8×8 2-D mesh network with XY dimension-ordered routing.

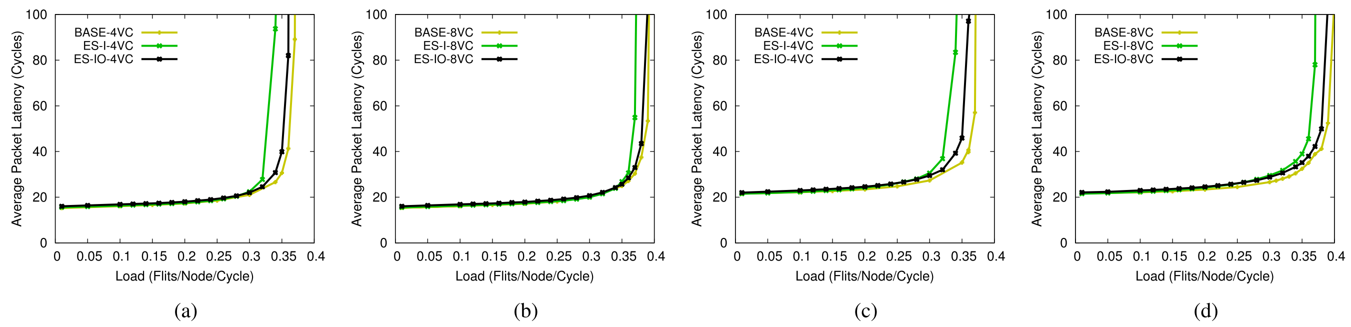


Fig. 13. Latency versus load curves for single-stage and two-stage baseline and ElastiStore-based pipelined routers with four VCs and eight VCs under UR traffic. Configurations with four and eight VCs per port are evaluated. (a) One-stage four VCs. (b) One-stage eight VCs. (c) Two-stage four VCs. (d) Two-stage eight VCs.

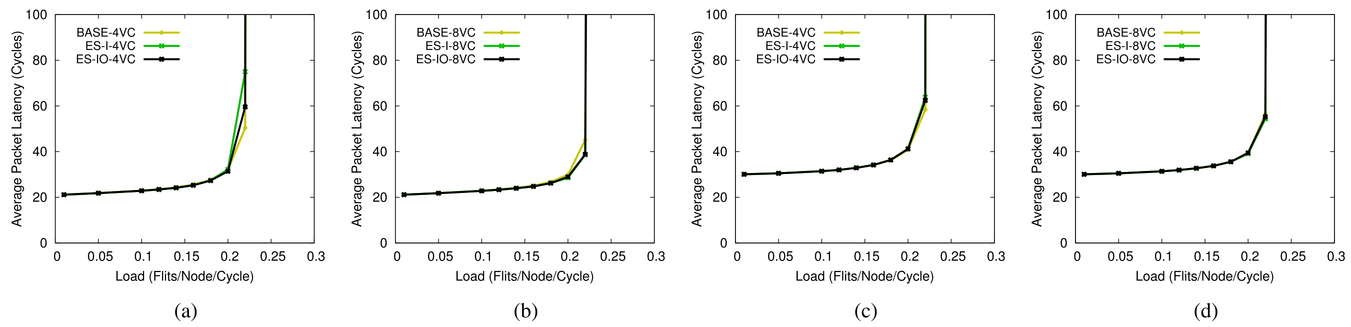


Fig. 14. Latency versus load curves for single-stage and two-stage baseline and ElastiStore-based pipelined routers with four VCs and eight VCs under BC traffic. Configurations with four and eight VCs per port are evaluated. (a) One-stage four VCs. (b) One-stage eight VCs. (c) Two-stage four VCs. (d) Two-stage eight VCs.

The evaluation involves two synthetic traffic patterns: 1) UR and 2) bit-complement (BC) traffic. Other permutation traffic patterns follow very similar trends to BC traffic. The injected traffic consists of two types of packets to mimic realistic system scenarios: one-flit short packets (just like request packets in a CMP) and longer five-flit packets (just like response packets carrying a cache line). For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, one-flit packets, and the rest being long, five-flit packets, in accordance to recent studies [21].

Even with the lower amount of buffering—which translates directly to area/power savings—the ElastiStore-based routers achieve a similar network performance when compared with single- and two-cycle VC-based routers. Fig. 13(a) and (b) shows the load-latency curves of all single-cycle routers under comparison using four VCs and eight VCs under UR traffic. In all cases, the performance of the routers is virtually indistinguishable, both at low and at high loads, while the ES-IO configuration achieves slightly less delay at high loads, when compared with ES-I. The same conclusion is drawn by the results shown in Fig. 13(c) and (d), for the case of two-stage routers. Due to its directed nature, BC traffic eliminates the small differences in the performance of baseline and ElastiStore-based designs at high loads, as shown in Fig. 14. Therefore, the savings of ElastiStore are offered to the NoC designer for free, without trading off performance.

C. Full-System Simulation Results

1) Experimental Setup: To assess the impact of ElastiStore on the overall system performance, we simulate a 64-core tiled CMP system running real application workloads on a commodity operating system. The execution-driven full-system simulation framework employs Wind River's Simics [22]—which handles the functional simulation tasks—extended with the Wisconsin Multifacet GEMS simulator [4]. The latter provides a detailed timing model of the memory hierarchy, and it includes the GARNET [23] cycle-accurate NoC simulator.

TABLE III
SYSTEM PARAMETERS FOR THE EXECUTION-DRIVEN FULL-SYSTEM SIMULATIONS

Processor	64 in-order UltraSparcIII+ cores in a tiled CMP architecture
OS	Solaris 10
L1 caches	Private, separate 32 KB I & D, 4-way set associative, 2-cycle latency, 64 B cache-line
L2 cache	Shared NUCA LLC, 4-way set associative, 16 MB total (64 cores×256 KB slice/core), 10-cycle latency, 64 B cache-line
Coherence	MOESI directory-based cache coherence protocol
Main memory	4 GB, 300-cycle latency
Network	8×8 2D Mesh, 4-stage router pipeline (+1 cycle link delay), XY Routing, 3 VCs per input port
VC size	Baseline: 6 flits per VC ElastiStore: 1-flit register per VC, and a 5-flit shared

Table III shows the full-system simulation parameters. Each CMP tile consists of an in-order UltraSparc III+ processor core with private and separate 32-KB L1 I and D caches. The CMP has a total of 16 MB shared L2 cache (each tile has a 256-KB L2 slice, i.e., 64×256 KB $= 16$ MB total) and 4 GB of off-chip main memory (dynamic RAM). The system uses the MOESI directory-based cache

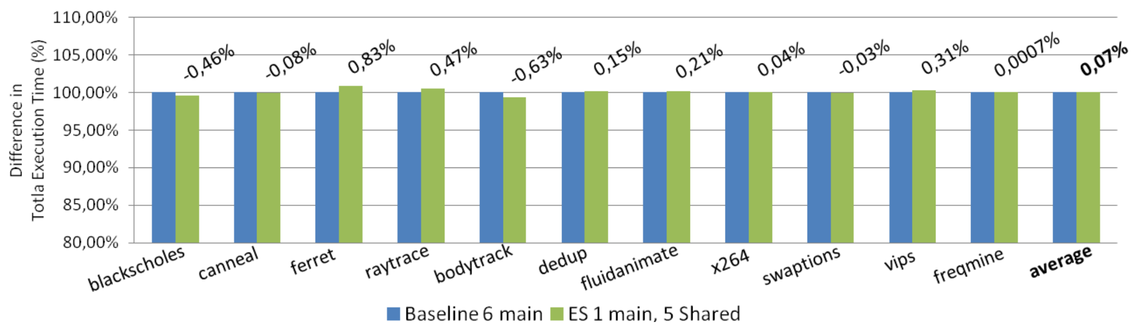


Fig. 15. Total execution times of the various PARSEC benchmark applications, normalized to the baseline router. The baseline router setup uses three VCs per input port, with each VC buffer storing six flits. The ElastiStore setup uses one single-flit register per VC, plus one multiflit buffer of size five, which is shared among all three VCs.

coherence protocol. The NoC is an 8 × 8 2-D mesh (i.e., one router per CMP tile) employing a dimension-ordered *XY* routing. Each router is implemented as a conventional four-stage pipelined router (RC, VA, SA, and ST) with one cycle interrouter link delay. As previously mentioned, cache coherence protocols require isolation between the various message classes to avoid protocol-level deadlocks. Specifically, the MOESI protocol requires at least three virtual networks to prevent protocol-level deadlocks. Consequently, in our simulations, each router input port has three VCs, each handling a specific message class of the coherence protocol.

Two different router architectures were considered. The *baseline* router design uses three VCs per input port, with each VC buffer having a six-flit depth. This setup represents a traditional NoC input port architecture, where the buffer space is statically allocated to each VC. On the contrary, the proposed ElastiStore architecture uses only one single-flit register per VC, plus a five-flit buffer shared among all three VCs, which aims to provide a direct comparison with the baseline setup, since each VC can hold a maximum of six flits (one in the main VC register + all five flits in the shared buffer). The six-flit buffers are necessary to cover the six-cycle round-trip time of a four-stage pipelined router (plus one interrouter link stage).

Both the baseline and ElastiStore-based router architectures were implemented within GARNET. The GARNET NoC simulator cycle accurately models the packet-switched routers, their pipelines, VC buffers, allocators/arbiters, crossbars, and all interrouter links.

The executed applications are part of the PARSEC benchmark suite [24], which contains multithreaded workloads from various emerging applications. All benchmarks were executed with 64 threads (one thread per processing core). The execution times reported are those of the regions of interest (ROIs), as identified in the PARSEC benchmarks. The ROI of each benchmark starts right after the initialization of the input data and ends when the computation is complete.

2) Results With PARSEC Applications: We ran the PARSEC benchmarks [24] using the setup described in Section VI-C1 to evaluate the two different NoC configurations. Fig. 15 shows the total execution times of the various applications, normalized to the baseline router.

The important insight that can be extracted from Fig. 15 is that a lightweight ElastiStore design with one single-flit register per VC and a five-flit shared buffer can yield near-identical performance as a baseline design with a six-flit buffer per input VC. In fact, the performances are indistinguishable. Both router architectures can provide a maximum space of six flits per VC, but the ElastiStore setup shares this maximum depth among all VCs (through the five-flit shared buffer). This sharing results in much more efficient resource utilization, with no impact on performance. The reason why such a dramatic decrease in buffer space is not accompanied by a decrease in the overall system performance is due to the very low average NoC traffic injection rates observed when running real single- and multithreaded applications in CMPs [25]. Hence, the baseline router architecture is, in fact, significantly over-provisioned for the needs of real application workloads, such as the PARSEC benchmarks. The fact that the ElastiStore architecture provides the same performance as the baseline with only 44% of the buffer space (a total of eight flit slots per input port versus 18 in the baseline) results in substantial savings.

D. Virtual Channels Versus Multiple Physical Networks

The separation of resources offered by VCs can also be achieved by multiple physical networks (built with wormhole routers), where each physical network serves a certain VC or, more accurately, a virtual network, since moving from one VC to another one is impossible in the case of multiple networks, due to the physical separation of the networks. Low-cost wormhole routers can be built with simple EBs at the inputs and the outputs of the router, as proposed in [5], using the two-slot EBs of Fig. 1(c) or (d).

Comparing VC-based architectures with multiple physical VC-less networks is a multidimensional problem, which has been the focus of other independent research efforts, such as [26]. However, in this paper, we repeat part of this study using ElastiStore-based VC routers.

In our comparisons, we consider four cases of an 8 × 8 2-D mesh network, which offers eight-way separation of resources, assuming a channel width of 64 bits. The first examined case involves a network built with eight-VC ElastiStore-based routers, where ElastiStores are used only at the inputs of the router (ESI-8VC-64). The second and

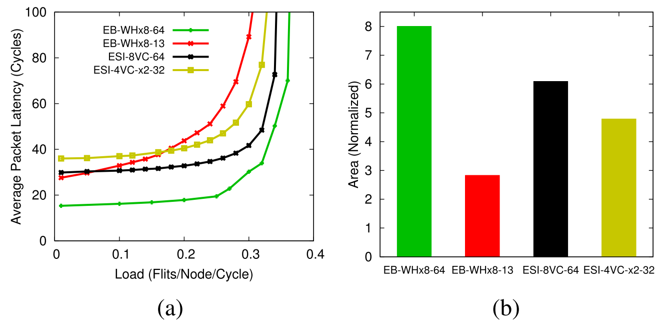


Fig. 16. (a) Load-latency curves and (b) normalized area cost of all examined configurations for comparing *virtual* (using ElastiStore buffers) or *physical* separation of resources. The area results are normalized to the area of a single EB-based wormhole network.

the third case involve eight physical networks of EB-based wormhole networks. The second case uses 64-bit channels per network, thus having a total of 64 × 8 bits per channel (EB-WHx8-64), while the third case assumes equal bisection bandwidth with the VC-based networks and uses 64/8 bits per channel (EB-WHx8-13). However, since we would like to keep the packet's header in one flit, we need at least 13 bits per network channel (2 bits for the flit's ID, 6 bits for the network addressing, and 5 bits for encoding the output port request, as needed by the LRC employed by all routers under comparison). The last case involves a hybrid of both worlds. It consists of two physical networks of four-VC ElastiStore-based routers, which—under equal bisection bandwidth—operate on 32-bit channels (ESI-4VC-x2-32).

While the first two cases can send and receive directly the one-flit and five-flit packets used in the previous experimental setup, the third and the fourth cases impose a significant serialization latency, since the number of flits per packet should be increased by 5× and 2×, to fit into the 13-bit and 32-bit channels, respectively. Please note that EB-WH×8-13 gets 1.6× more bisection bandwidth that ElastiStore-based architectures due to the 13-bit channels.

For a fair comparison, we assume a static VC allocation policy for the VC-based routers. In this case, packets are not allowed to change VC in flight and are forced to keep the VC given to them at the source (i.e., similar to what happens when a packet enters a physical network of VC-less routers). This feature simplifies significantly the VA logic of VC-based NoC routers [27], with a slight reduction in the overall network throughput, and can be used with ElastiStore, too.

First, we compare the four examined cases in terms of network performance, using the same configuration of Section VI-B for UR traffic. The results are shown in Fig. 16(a). EB-based wormhole routers are small and fast. Thus, in the load-latency curves of Fig. 16(a), we assume that the routers can switch incoming flits in one cycle. On the contrary, VC-based routers operate in a three-stage pipelined configuration to achieve the same clock frequency. As expected, the EB-WHx8-64 configuration has the best performance, both in terms of latency and throughput. The EB-WHx8-13 setup is the worst. ElastiStore-based VC-based architectures enjoy high throughput of operation, despite having 8× less bisection bandwidth than EB-WHx8-64, and, as it will be shown later, they achieve this goal with significantly less cost. The only drawback of the VC-based architectures is the high zero-load latency, due to the increased number of pipeline stages required to achieve high clock frequencies.

The cost of the implementation of each case (in terms of area) is shown in Fig. 16(b), where—for the ElastiStore-based routers—three-stage-pipelined alternatives are examined with static VC allocation, which have almost equal delay to that of simpler single-cycle EB-based wormhole routers. ElastiStore with eight VCs allows the design of NoCs offering eight-way separated resources, with less area cost than eight physical EB-based WH networks. The 24% area savings are the result of the efficient buffer sharing mechanism of ElastiStore, derived using the three design principles presented in Section IV-B and the newly introduced credit-based flow control that employs negative credits. As expected, the eight physical networks of EB-based WH routers operating under equal bisection bandwidth have lower area cost, but, as shown in Fig. 16(a), they also have the worst performance, both in terms of latency and saturation throughput. The hybrid solutions that employ multiple physical networks of ElastiStore-based routers, each one supporting a smaller number of VCs, keep the network cost low with acceptable performance.

The final outcome of this analysis is that ElastiStore significantly reduced the cost of VC-based routers, without sacrificing the throughput of the network, thereby allowing the design of NoCs with a high degree of resource separation with lower cost than multiple physical networks. The remaining challenge for VC-based architectures, which is orthogonal to ElastiStore, is the simplification of their allocation logic to decrease the number of pipeline stages required for achieving high clock frequency. This would also lower the zero-load latency of each flit.

VII. Related Work

In this section, we focus on the two main thrusts of prior research that are most relevant to the proposed ElistiStore design: 1) elastic channels and buffering in NoCs and 2) shared-buffering schemes in NoC routers.

Kodi *et al.* [11] explored the integration of elastic storage elements into the links of NoCs, in conjunction with traditional dynamically assigned VC input buffers. The introduced ideal router employs a hybrid flow control, where multiple VCs are multiplexed on a single-lane link with a ready/valid handshake. This hybrid flow control obligates the first VC that is full to stop the remaining VCs—with possibly incoming flits on the link—thus creating dependencies across VCs that may lead to higher level protocol deadlocks if not handled appropriately, irrespective of the available buffer space.

The notion of elastic channels in NoCs was further developed in [5] and [9], which reused the latch-based EBs presented in [12] for minimizing the buffering cost in NoC routers. To alleviate the problems caused by the serializing nature of elastic links, which do not provide any isolation of traffic flows and prevent the interleaving of packets, the authors resort to multiple physical channels, to create multiple subnetworks, instead of relying on a hybrid flow control mechanism.

A modified version of elastic operation, which also enables traffic-flow separation without multiple physical resources, has been employed in [10] and [28]. In [10], elasticity is only preserved across the multidrop busses of the MECS topology, and when it comes to the inputs of the routers, traditional VC-based buffers are used. On the contrary, in [28], elasticity is preserved on single-lane channels, while separation of resources is enabled by packet-level bubble flow control.

In contrast, ElastiStore uses lightweight EB primitives, and it can support any number of VCs, without the need to have multiple physical subnetworks, or to rely on any hybrid flow control mechanism. Moreover, ElastiStore can be used either as a distributed buffer primitive or at the inputs and output of routers. The semantics of the VC flow control mechanism are preserved and implemented either with independent ready/valid handshakes per VC or using credits. The presence of a shared buffer in ElastiStore is instrumental in optimizing the use of the available buffer space and covering the arbitrary round-trip times imposed by router pipelining. In essence, the proposed mechanism achieves the same objective as the significantly more expensive per-input shared-buffering techniques [29]–[34]. Other buffering architectures that extend sharing across multiple inputs [35]–[37] require multiporting, and even though they provide significant performance in terms of throughput, they suffer in terms of router delay (or they increase the required pipeline stages) and they do not scale well to higher numbers of VCs per input port.

As opposed to the extremely lightweight shared-buffering structure of ElastiStore, the aforementioned shared-buffering architectures rely on fairly complex logic to keep track of the location of flits within the unified buffer, which needs significant modifications to handle variable packet sizes. Specifically, in shared-buffer schemes, designers predominantly use linked lists [29]–[31] and table-based approaches [32] that may possibly need multiported memory accesses in their pointer-tracking logic or self-compacting buffers [33], [34] to coordinate traffic flow through the buffers. Each VC maintains its own set of pointers to identify where its flits are located in the buffer. For example, in the case of ViChaR [32], two control logic modules are needed in each input port (the arriving/departing flit pointer logic and the slot availability tracker) for the correct operation of the shared buffer, regardless of the size of the buffer. While the cost of this logic is amortized in routers with large buffer space, the overhead becomes significant in routers with minimal buffering.

Furthermore, state-of-the-art buffering mechanisms assume that all flits of the VCs that reside in the shared buffer space should be visible in the allocation logic, and the throughput per VC is a result of the VC utilization and the status of the buffer space (available credits) in each cycle. This behavior is avoided in ElastiStore, which allows only the main registers of each VC (just one per VC) and not the shared buffer to participate in allocation, and the throughput received per VC follows a more strict distribution, allowing only one VC to receive full throughput when it is the only active one (this is the only case that full throughput matters). The shared buffer is isolated from the router's operation; it just refills the main registers of the VC, when needed. In this way, every dependency across VCs is eliminated—which also enables deadlock-free operation—while variable packet sizes are supported for free as in any baseline VC buffer with no sharing. The shared buffer of the generalized ElastiStore partially follows a self-compacting approach, similar in concept to [33] and [34], although much simpler to implement. Flits are written at the end of an FIFO irrespective of the VC they belong to, and no data movement is needed to find an empty place for an incoming flit.

VIII. Conclusion

The NoC router's buffer architecture is a critical design aspect that affects both network-wide performance and implementation characteristics. In this paper, we efficiently merge elastic operation and buffering with VC flow control. The derived buffer architecture, called ElastiStore, can take many forms, based on application demands. ElastiStore can be used as the simplest form of VC buffering, which uses only one register per VC, plus one more dynamically shared register that enables a single active VC to achieve full throughput. ElastiStore can also be applied on the links, as a distributed elastic buffering architecture, or at the inputs and the outputs of NoC routers. In addition, when NoC routers follow a pipelined organization, ElastiStore can be adapted to its most generic form, which utilizes a larger shared buffer to cover the increased round-trip time arising from the pipelined operation. The new design principles governing the design of ElastiStore-based routers enable the design of low-cost routers with significant area savings and no delay penalty, as compared with current state-of-the art VC-based routers. More importantly, the resulting ElastiStore-based routers offer the same network performance as the aforementioned VC-based implementations, as verified using extensive simulations with both synthetic traffic and real application workloads.

References

[1] J. Handy, "NoC interconnect improves SoC economics," in *Semiconductor Market Research*. Los Gatos, CA, USA: Objective Analysis, 2011.

[2] W. J. Dally, "Virtual-channel flow control," in *Proc. Int. Symp. Comput. Archit.*, May 1990, pp. 60–68.

[3] J. Browne, *On-Chip Communications Network Report*. Mountain View, CA, USA: Sonics, 2012.

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

[5] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic-buffer flow control for on-chip networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 151–162.

[6] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with latency in SOC design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, Sep./Oct. 2002.

[7] A. Roca, C. Hernández, J. Flich, F. Silla, and J. Duato, "Silicon-aware distributed switch architecture for on-chip networks," *J. Syst. Archit.*, vol. 59, no. 7, pp. 505–515, Aug. 2013.

[8] N. Concer, M. Petracca, and L. P. Carloni, "Distributed flit-buffer flow control for networks-on-chip," in *Proc. 6th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, 2008, pp. 215–220.

[9] G. Michelogiannakis and W. J. Dally, "Elastic buffer flow control for on-chip networks," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 295–309, Feb. 2013.

[10] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "A QoS-enabled on-die interconnect fabric for kilo-node chips," *IEEE Micro*, vol. 32, no. 3, pp. 17–25, May/Jun. 2012.

[11] A. K. Kodi, A. Sarathy, and A. Louri, "iDEAL: Inter-router dual-function energy and area-efficient links for network-on-chip (NoC) architectures," in *Proc. 35th Int. Symp. Comput. Archit. (ISCA)*, Jun. 2008, pp. 241–250.

[12] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. ACM/IEEE Design Autom. Conf.*, Jul. 2006, pp. 657–662.

[13] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, Jan./Feb. 1997.

[14] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA, USA: Morgan Kaufmann, 2004.

[15] M. Azimi, D. Dai, A. Mejia, D. Park, R. Saharoy, and A. S. Vaidya, "Flexible and adaptive on-chip interconnect for tera-scale architectures," *Intel Technol. J.*, vol. 13, no. 4, pp. 62–79, 2009.

[16] Y. Lu, C. Chen, J. McCanny, and S. Sezer, "Design of interlock-free combined allocators for networks-on-chip," in *Proc. IEEE Int. SOC Conf. (SOCC)*, Sep. 2012, pp. 358–363.

[17] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. 7th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Jan. 2001, pp. 255–266.

[18] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proc. 31st Annu. Int. Symp. Comput. Archit.*, 2004, pp. 188–197.

[19] P. Salihundam *et al.*, "A 2 Tb/s 6 × 4 mesh network with DVFS and 2.3 Tb/s/W router in 45nm CMOS," in *Proc. IEEE Symp. VLSI Circuits (VLSIC)*, Jun. 2010, pp. 79–80.

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

[21] S. Ma, N. E. Jerger, and Z. Wang, "Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip," in *Proc. IEEE 18th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2012, pp. 1–12.

[22] Wind River Inc. *Simics, Product Overview*. [Online]. Available: http://www.windriver.com/products/simics/, accessed Dec. 2014.

[23] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2009, pp. 33–42.

[24] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 72–81.

[25] R. Hesse, J. Nicholls, and N. E. Jerger, "Fine-grained bandwidth adaptivity in networks-on-chip using bidirectional channels," in *Proc. 6th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, May 2012, pp. 132–141.

[26] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: Two alternatives to improve NoC performance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 12, pp. 1906–1919, Dec. 2013.

[27] F. Gilabert, M. E. Gómez, S. Medardoni, and D. Bertozzi, "Improved utilization of NoC channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *Proc. NOCS*, May 2010, pp. 165–172.

[28] S. M. Hassan and S. Yalamanchili, "Centralized buffer router: A low latency, low power router for high radix NOCs," in *Proc. 7th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, Apr. 2013, pp. 1–8.

[29] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," in *Proc. 15th Annu. Int. Symp. Comput. Archit.*, 1988, pp. 343–354.

[30] W.-T. Su, J.-S. Shen, and P.-A. Hsiung, "Network-on-chip router design with buffer-stealing," in *Proc. 16th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2011, pp. 160–164.

[31] G. Kim, J. Kim, and S. Yoo, "FlexiBuffer: Reducing leakage power in on-chip network routers," in *Proc. 48th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2011, pp. 936–941.

[32] 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 *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 333–346.

[33] N. Ni, M. Pirvu, and L. N. Bhuyan, "Circular buffered switch design with wormhole routing and virtual channels," in *Proc. ICCD*, Oct. 1998, pp. 466–473.

[34] J. Park, B. W. O'Krafka, S. Vassiliadis, and J. Delgado-Frias, "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 1994, pp. 713–722.

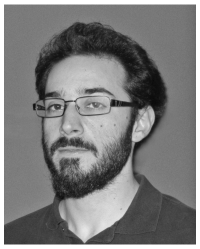
[35] A. T. Tran and B. M. Baas, "RoShaQ: High-performance on-chip router with shared queues," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 232–238.

[36] M. H. Neishaburi and Z. Zilic, "Reliability aware NoC router architecture using input channel buffer sharing," in *Proc. 19th ACM Great Lakes Symp. VLSI*, 2009, pp. 511–516.

[37] K. Latif, A.-M. Rahmani, L. Guang, T. Seceleanu, and H. Tenhunen, "PVS-NoC: Partial virtual channel sharing NoC architecture," in *Proc. 19th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Feb. 2011, pp. 470–477.

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

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



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

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

Kypros Chrysanthou received the B.Sc. degree in computer engineering from the University of Cyprus, Nicosia, Cyprus, in 2013, where he is currently pursuing the master's degree with the Department of Electrical and Computer Engineering.

His current research interests include on-chip interconnection networks, computer architecture, and fault-tolerant and reliable system design.

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

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

Giorgos Dimitrakopoulos received the Ph.D. degree from the University of Patras, Patras, Greece.

He is currently a Lecturer with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. He is interested in the design of digital integrated circuits and computer architecture. His current research interests include the design of on-chip interconnection networks and ultralow-power graphics accelerators and processors.