PhaseNoC: Versatile Network Traffic Isolation Through TDM-Scheduled Virtual Channels

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

***Abstract*—As multi/many-core architectures evolve, the demands on the network-on-chip (NoC) are amplified. In addition to high performance and physical scalability, the NoC is increasingly required to also provide specialized functionality, such as network virtualization, flow isolation, and quality-of-service. Although traditional architectures supporting virtual channels (VCs) offer the resources for flow partitioning and isolation, an adversarial workload can still interfere and degrade the performance of other workloads that are active in a different set of VCs. In this paper, we present PhaseNoC, a truly noninterfering VC-based architecture that adopts time-division multiplexing at the VC level. Distinct flows, or application domains, mapped to disjoint sets of VCs are isolated, both inside the router's pipeline and at the network level. Any latency overhead is minimized by appropriate scheduling of flows in separate phases of operation, irrespective of the chosen topology. When strict isolation is not required, the proposed architecture can employ opportunistic bandwidth stealing. This novel mechanism works synergistically with the baseline PhaseNoC techniques to improve the overall latency/throughput characteristics of the NoC, while still preserving performance isolation. Experimental results corroborate that—with lower cost than state-of-the-art NoC architectures, and with minimum latency overhead—PhaseNoC removes any flow interference and allows for efficient network traffic isolation.**

***Index Terms*—Network traffic isolation, network-on-chip (NoC), time-division multiplexing (TDM), virtual channels (VCs), virtual networks.**

I. Introduction

NETWORKS-ON-CHIP (NoCs) have established their position as the de facto communication medium in multicore systems, owing to their scalability attributes. To sustain system scalability into the many-core domain, it is imperative that the NoCs hardware cost is minimized, while not sacrificing network performance [1]. This objective is nontrivial, since the functionality expected from the NoC continues to grow. For instance, multicore systems increasingly require some form(s) of isolation—or separation—among the traffic flows of concurrently executing applications. In the simplest case, such segregation attributes are desired due to restrictions imposed by higher-level protocols, e.g., cache coherence in chip multiprocessors (CMPs). Constraints of this type are typically satisfied through static separation of flows (termed "message classes") using virtual channels (VCs) within the NoC [2]. Nevertheless, the requirements of flow isolation are often more elaborate than mere physical partitioning, and include advanced rules that describe how the flows are allowed to interact, or the level of service that each flow should receive.

The concept of flow isolation can manifest in two forms: 1) high-assurance, secure-grade noninterference, as required by systems demanding security guarantees [3] and 2) performance isolation with minimum-bandwidth and bounded-latency quality-of-service (QoS) guarantees [4]. Noninterference is becoming increasingly relevant within the context of virtual execution platforms, whereby the sharing of hardware resources must be impervious to cross-interference [4].

Existing solutions have only been able to satisfy one of the two above-mentioned objectives; i.e., either strict isolation at the cost of lower network throughput, or more efficient performance isolation without noninterference guarantees. There is currently no architecture that can selectively provide both properties, with minimal impact on overall network area and performance. The difficulty lies within the inherent nature of NoC architectures. Due to the widespread sharing of router resources (ports, arbiters, crossbar, channels, etc.) among all the VCs, strict noninterference between flows is, by definition, not guaranteed. In fact, VCs are interfering by construction, since multiple VCs are eligible to compete for the same NoC resources at any given time.

The main goal of this paper is to develop a NoC micro-architecture that can effectively bridge the unseemly dichotomy between strict noninterference assurance and flexible/efficient QoS provisioning. Building on this fundamental premise, we propose the PhaseNoC architecture, which provides this dual flow-isolation property. The crux of PhaseNoCs operation revolves around the notions of domains and phases. A domain is defined as an individual VC—or a group of VCs—serving one (or more) application flow(s). Each PhaseNoC router can serve any number of domains in parallel, and it can guarantee strict isolation across the supported domains. The various domains are served in phases using a fresh reinterpretation of time-division multiplexing (TDM), which is applied at the VC level.

Overall, PhaseNoC is characterized by three fundamental properties, which highlight the key contributions of this paper.

1) PhaseNoC relies on a complete overhaul of the routers' micro-architecture and their internal pipeline operation

Manuscript received April 30, 2015; revised August 9, 2015; accepted September 17, 2015. Date of publication October 7, 2015; date of current version April 19, 2016. The work of I. Seitanidis was supported by the Ph.D. scholarship of Alexander S. Onassis Foundation. This paper was recommended by Associate Editor Y. Wang.

A. Psarras, I. Seitanidis, and G. Dimitrakopoulos are with the Electrical and Computer Engineering Department, Democritus University of Thrace, Xanthi 67100, Greece (e-mail: apsarra@ee.duth.gr; iseitani@ee.duth.gr; dimitrak@ee.duth.gr).

J. Lee is with the Electrical and Computer Engineering Department, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: junghee.lee@utsa.edu).

C. Nicopoulos is with the Electrical and Computer Engineering Department, University of Cyprus, Nicosia 1678, Cyprus (e-mail: nicopoulos@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/TCAD.2015.2488490

into a domain-amenable organization called explicit pipelining, which ensures that the various domains will never "compete" (i.e., arbitrate) with each other to gain access to any network resource. In fact, the domains are completely oblivious to each other's existence, and there is no information leak whatsoever across the domains, as highlighted in Section II. Essentially, PhaseNoC constitutes a bandwidth-programmable traffic isolation mechanism that allows separate virtual networks—the supported domains—to operate concurrently and in complete isolation, despite sharing the same hardware infrastructure.

2) To address the latency overhead typically associated with TDM-based scheduling, PhaseNoC utilizes precisely choreographed phase propagation throughout the NoC, by applying VC-level TDM schedules at the granularity of individual router pipeline stages. The phases are coordinated into optimally scheduled interlocked propagating waves, which ensure that in-flight packets of all domains experience the minimum possible latency. A generalized methodology that enables the creation of optimal phase schedules in any network topology is provided in Section III, which also includes weighted phase schedules, whereby bandwidth is unevenly distributed to the domains.
3) If secure-grade isolation is not required, PhaseNoC is also able to support a more "relaxed" traffic isolation mode that sacrifices the full noninterference properties to improve overall performance. PhaseNoCs opportunistic bandwidth stealing (OBS) mechanism, introduced in Section IV, utilizes any bandwidth left unused by a domain in each cycle. The unused bandwidth is recycled among the domains that need it at any given time. The OBS technique cost-effectively facilitates the fusion of strict bandwidth guarantees and best-effort (BE) services, as encountered in mixed-criticality environments. Also, OBS allows for the seamless handling of bursty dynamic behavior, e.g., when a domain momentarily requires more bandwidth, beyond its statically allocated VC-level TDM quota. The extra traffic is "absorbed" by OBS, without affecting the other domains.

PhaseNoC is evaluated through multifaceted and extensive cycle-accurate simulations, as presented in Section V. The evaluation framework employs both synthetic traffic patterns, and execution-driven full-system simulations with real application workloads. Comparisons against SurfNoC [3]—the current state-of-the-art in secure NoC architectures—and baseline NoC architectures that do not provide any isolation guarantees indicate that PhaseNoC provides consistently higher performance. A detailed hardware analysis using a 45 nm standard-cell library verify that PhaseNoC yields substantial cost savings, as compared to state-of-the-art VC-based architectures, despite providing the extra functionalities of both strict and relaxed flow isolation.

II. PhaseNoC Router Architecture

The PhaseNoC router organizes the allocation and the switching tasks executed per-packet and per-flit in phases, ensuring that each phase deals only with a distinct set of VCs (a so called domain). Every input port of the PhaseNoC router hosts V VCs that are organized in D domains, where each domain may contain a group of m VCs ($m = V/D$).

Packets entering the input VCs of their domain must find their way to the proper output, after going through several allocation steps. The head flit of a packet first calculates its output port through routing computation (RC). It then uses the selected output port to allocate an output VC (i.e., an input VC in the downstream router) in VC allocation (VA). Once a head flit has acquired an output VC, it tries to gain access to the output port through switch allocation (SA). Winners of SA traverse the crossbar switch (ST), and are written in an output pipeline register. Finally, the flit moves to the next router through link traversal (LT), and is written in the downstream router's input buffer (BW) [5].

Noninterference is guaranteed if, at any allocation or switching step, the participating (competing) packets belong exclusively to the same domain (group of VCs). Thus, contention and interference can only arise between packets and flits of the same domain. PhaseNoC guarantees noninterference among all supported domains through its phased operation, i.e., each phase—covering all inputs of the router—deals exclusively with a single domain, and each phase is completely isolated (in terms of utilized resources) from other phases (and from other domains). The phase activation process should be the same for all inputs of the router, thus making it impossible for two different inputs to participate in a router's allocation/switching stage with packets/flits that belong to different domains.

A. Explicit Pipelining

When a router is pipelined, it may operate simultaneously on many application domains, by selecting the appropriate phase for each pipeline stage. Care should be taken to assure that two domains never simultaneously participate in the same stage. To ascertain this behavior, we let the router's pipeline operate in a predetermined (although programmable) static schedule. For example, once the group of VCs belonging to domain $D0$ perform VA, the group of VCs of domain $D1$ perform SA, while the winning flits of domain $D2$ pass the crossbar (ST), and the flits of domain $D3$ are on the link toward the next router. Therefore, in each cycle, the router is fully utilized, but each part of the router works on a different domain. This unique feature of PhaseNoCs pipeline ensures that each stage works on a different phase, and the flits/packets served in each phase belong exclusively to a single domain. We term this type of pipeline operation as explicit pipelining.

To achieve this behavior, each input port should own a separate path to VA, SA, and ST, as shown in Fig. 1(a) (only the path to ST carries real data). Each input can send to each part of the router the requests/data of a different domain (group of VCs), provided that the select signals of the multiplexers (that coordinate the phase propagation) never point to the same domain. By setting the phase of each stage appropriately, all of them may be executed in parallel, but each stage acts on the packets/flits of a different domain. It is critical, however, that all inputs see the same order of phase activation. Additionally, secure-grade isolation can be achieved by building the multiplexers in front of each stage of the router using trusted logic [6], according to the guidelines in [3].

For zero-latency phase scheduling, a flit should always find the phase of its current pipeline stage aligned to its domain, and may move uninterrupted (unless it loses to another flit of

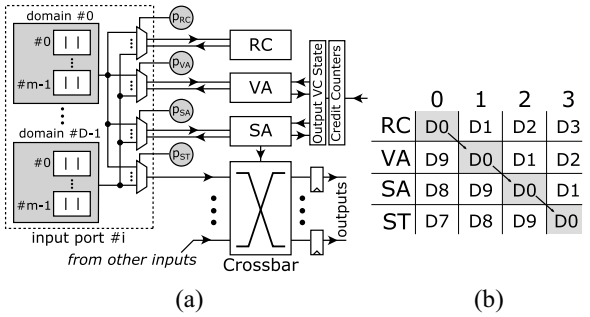

Fig. 1. (a) PhaseNoC router architecture. Only a set of VCs (one domain) is allowed to participate from each input in each allocation stage, and it is the same set (domain) for all inputs. One multiplexer for each stage, controlled by a TDM schedule, allows different domains to be served by different stages of the router. (b) Cycle-by-cycle operation of a four-stage pipelined PhaseNoC router. In each cycle, all router parts are utilized, with each pipeline stage serving (allocating or switching) a different domain (group of VCs).

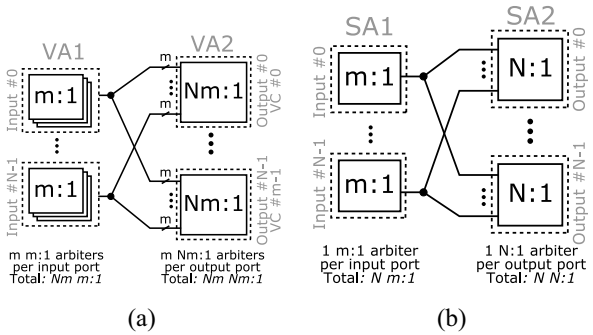

Fig. 2. PhaseNoCs reduced allocators for (a) VA and (b) SA router pipeline stages.

the same domain during arbitration). For example, if the phase of VA in cycle 1 is serving domain 1, then the phase of the SA stage in cycle 2 should be serving domain 1 as well. This behavior is captured in Fig. 1(b), which presents the cycle-by-cycle activity of an input port's pipeline stages. In each cycle (columns), all of the pipeline stages (rows) operate on a different domain, whose ID is shown by the number in the corresponding box. In the first cycle, RC operates in phase 0, so domain 0 is able to calculate its output port. In the next cycle, it finds its phase in VA, and it is able to allocate an output VC of its domain successfully, as flits of domain 1 perform RC. In cycle 2, domain 0 uses its allocated VC to participate in SA, while the head flits of domain 1 try to acquire an output VC. Whether this allocation is successful or not depends only on the contention appearing between the flits of domain 1 from all inputs; only domain 1 flits are allowed to participate in VA in this phase. In cycle 3, the flits from domain 0 that won in SA traverse the crossbar, and it is the turn of domain 2 to perform VA.

Explicit pipelining guarantees noninterference between flits/packets of different domains, irrespective of the packet length (flits per packet) and the pipeline depth of the routers. At any given time, each pipeline stage deals with the flits of a certain domain that may belong to packets of arbitrary length.

Routers with fewer pipeline stages can be built by merging stages. However, due to the phased operation of PhaseNoC routers, merging two stages means that their phase multiplexers should also be merged.

B. Structure of Allocators

In an N-port router with D domains and m VCs per domain, there exist a total of $N \times D \times m$ input and output VCs (recall that $D \times m = V$, i.e., the total number of VCs per input port of the router). The VA process between those input and output VCs in a traditional router would require an $N \times D \times m : N \times D \times m$ allocator. However, in PhaseNoC, in each clock cycle, only a single domain performs VA to a group of m VCs per input. Thus, in the whole router, at most $N \times m$ input VCs will try to allocate an output VC. Since an input VC will never request an output VC outside each domain, then at most $N \times m$ output VCs will be requested.

Thus, for completing the VA process in PhaseNoC, a simpler $N \times m : N \times m$ VC allocator suffices, which serves a different domain in each clock cycle. As illustrated in Fig. 2(a), VA is performed in two stages. In VA1, each input VC of the domain matched to the current phase of the router selects one available and possibly ready (i.e., has at least one buffer slot) output VC. The selection is made by round-robin arbiters that select one of the m active VCs of an output port of the same domain. In VA2, each of the $N \times m$ output VCs is assigned through an $N \times m : 1$ arbiter to at most one input VC of the same domain.

Similar simplifications can be derived for the switch allocator as well, which, again, involves two steps of arbitration, as shown in Fig. 2(b). The SA1 arbiter per input port is reduced from a $V : 1$ arbiter in a baseline implementation without domains to an $m : 1$ arbiter in PhaseNoC, since local arbitration involves only the input VCs of the domain currently active in the SA stage. The SA2 stage, which selects the input port that will access each output port, cannot be simplified further, and it still requires an $N : 1$ arbiter per-output.

Note that, if desired, the SA2 arbiters may be modified to implement any type of arbitration policy, and said policy would only affect the traffic within a particular domain. For example, the SA2 arbiters could implement policies such as round-robin, weighted arbitration [7], or even a static TDM schedule in the form of a TDM wheel [8]. The chosen arbitration policy will only affect the traffic within a domain, with no impact on the global TDM phase schedule. This functionality enables PhaseNoC to facilitate, if required, two-level schedules: 1) the top-level TDM schedule coordinating the domain phases and 2) second-level schedules for the traffic within each domain.

Although the VA and SA allocators are shared by different domains, sharing is performed in time and, thus, it is impossible for packets that belong to two different domains to compete for the same resource in the same cycle. Additionally, to completely eliminate any domain interference, all arbiters should use D separate priority vectors, each one corresponding to the active domain. The appropriate set is selected by the phase of the allocation stage, ensuring that arbitration decisions are completely separated across domains.

III. Application-Domain Scheduling

PhaseNoC routers guarantee noninterference between different domains by time-multiplexing the allocators, the crossbar, and the output physical channels in different domains in each clock cycle. This time-multiplexing scheme ensures that the latency and throughput of each domain is independent of the other domain's load; contention is only allowed between the packets of the same domain. To achieve zero latency

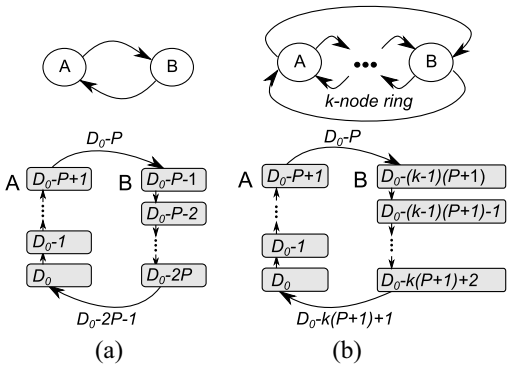

Fig. 3. Scheduling constraints set by (a) direct and (b) wrap-around links for achieving zero-latency flit traversal across routers.

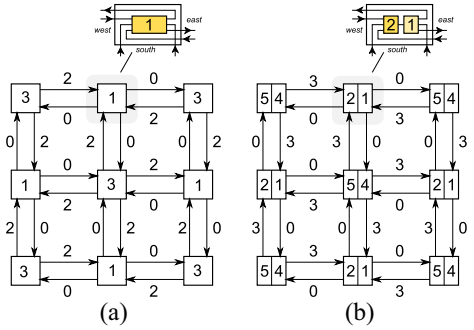

Fig. 4. Zero-latency schedules on a 3×3 2-D mesh using (a) four domains with single-cycle routers and (b) six domains in two-stage pipelined routers. All incoming links feed the first pipeline stage of the router, and all output links are driven by the last stage.

overhead in flit propagation, the inter-router phase propagation should be extended to the network level to allow flits of one domain to travel in the network in a wave-like manner. The flits of each domain would traverse many hops in consecutive cycles, without waiting for the turn of their application domain to come. In this way, noninterference is achieved with the minimum possible latency, since only the flits of the same domain experience contention throughout the network.

A. Topology Constraints

Depending on the pipeline depth of the router, each router is concurrently active on one, or many, different application domains (groups of VCs). Therefore, once an application domain performs RC in cycle t_0, the same application domain will proceed to VA in cycle t_0+1, to SA in cycle t_0+2, to ST one cycle later, and it will eventually appear on the link (LT) in cycle t_0+4. Therefore, in order for the flits of this application domain not to experience any latency overhead, the router at the other side of the link should schedule the start of the service of this particular application domain in cycle t_0+5; the first step is again RC. So, for experiencing no latency between any two routers, the domain served in the first pipeline stage of both routers connected with a forward link should differ by $P+1$ cycles; P, due to the router pipeline, plus one for the single cycle spent on the link.

To present the conditions that satisfy this property, we first study the simple case of two directly connected routers shown in Fig. 3(a). We assume that the first pipeline stage of router A is serving domain D_0 in the current cycle. Under a zero-latency schedule, the first pipeline stage of router B will reach the domain currently served by the first pipeline stage of router A after P cycles. At the same time, we should guarantee that this relationship between any two neighboring routers also holds in the backward direction, so that any traffic crossing router B toward router A does not experience any latency either. The output links of router B forward flits of domain $D_0 - 2P - 1$ when the first stage of router A is serving domain D_0. Therefore, if we want router A to receive in-sync the flits coming from B, then, in the next clock cycle, the next domain of A, which will be $D_0 + 1 \bmod D$, should match the domain that is served on the incoming links in this cycle, i.e., $D_0 - 2P - 1 \bmod D$, where D is the number of domains. This constraint is satisfied when

$$2(P+1) \bmod D = 0. \tag{1}$$

Condition (1) requires that the number of domains D is equal to any of the factors of 2, $P+1$, or $2(P+1)$.

Fig. 4 depicts the assignment of four domains to PhaseNoC routers and, implicitly, to the network links, for a 3×3 mesh. Fig. 4(a) shows a snapshot of the reset phase of the network, assuming single-cycle routers ($P = 1$, and, thus, $D = 4$), which select the same domain for all the inputs of the router. The number inside the nodes and next to the links corresponds to the domain ID. Then, each router independently increases its working domain by one in each clock cycle and wraps around when the number of domains is reached. Equivalently, Fig. 4(b) shows the schedule applied to two-stage pipelined routers (RC-VA in one cycle for one domain, and SA-ST in the next cycle for a different domain) that can serve six domains without any latency overhead.

Once the number of domains is selected appropriately, the assignment of starting phases to the internal pipeline stages of the routers and the links can begin from any node and propagate decrementally to the remaining nodes, while taking care to wrap around to domain $D-1$ when the domain 0 is reached. During the propagation of starting phases, all the incoming links of each router (and, as a result, all outgoing links, too) should serve the same application domain. This constraint is a requirement for offering isolation across domains, since on every pipeline stage inside the router the same domain is active on all inputs concurrently.

The proposed scheduling mechanism is extended to topologies that also contain wrap-around links, such as rings and tori. As depicted in Fig. 3(b), two adjacent nodes may be connected with a forward and a backward link using a wrap-around connection in a k-node 1-D ring connection. In this case, and assuming that the first pipeline stage of router A is serving application domain D_0, the output links of router B (at the other end of the ring; k hops away) should be serving the domain $D_0 - k(P+1) + 1 \bmod D$, in order for any flits from B to A not to experience any latency. If a zero-latency penalty is also required for the flits that cross the wrap-around connection, then the domain currently being served by the outputs of router B should be equal to the domain that the first pipeline stage of router A will serve in the next cycle, i.e., $D_0 + 1 \bmod D$. Thus, for allowing a perfect schedule when wrap-around connections exist in the topology, we should guarantee that $(D_0 - k(P+1) + 1) \bmod D = (D_0 + 1) \bmod D$, which translates to

$$k(P+1) \bmod D = 0. \tag{2}$$

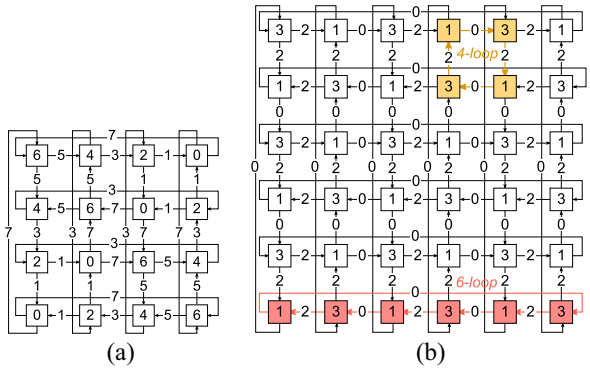

Fig. 5. Examples of zero-latency schedules in two Manhattan-grid NoCs with single-cycle routers. (a) Eight-domain schedules are allowed, constrained by the wrap-around rings. (b) Only four domains can be supported here under perfect phase scheduling, due to the co-existence of two unequal loops in the NoC.

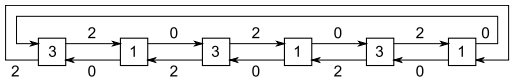

Fig. 6. Zero-latency schedule of four domains in a six-node ring network.

The loop constraint of (2) is satisfied when D is equal to any of the factors of k, $P+1$, or $k(P+1)$.

Fig. 5 depicts a snapshot of the reset state of phase propagation in two Manhattan-grid networks. In Fig. 5(a), the topology consists of four-node rings in each dimension, which limit the number of application domains that can be hosted under a perfect (zero-latency) schedule to eight domains, when routers operate in a single cycle. On the contrary, in Fig. 5(b), the topology includes two sets of loops: 1) the wrap-around rings that include six nodes and allow the scheduling of 12 domains with single-cycle routers and 2) the internal loops that span four nodes and limit the supported domains to 8. However, since the two loops co-exist, the maximum number of domains that can be supported under a zero-latency schedule is $\gcd(12, 8) = 4$.

Similar limitations arise when wrap-around ring connections spanning k nodes [Fig. 3(b)] co-exist with direct (self-loop) connections [Fig. 3(a)], and, thus, the constraints of both connection patterns should be simultaneously satisfied. In this case, the number of domains D that can be scheduled with zero-latency overhead is equal to $\gcd(2(P+1), k(P+1))$. The number of domains is $2(P+1)$ or $P+1$ for even or odd values of k, respectively. For instance, as shown in Fig. 6, a six-node ring of single-cycle routers can host at most four application domains, i.e., $\gcd(2(1+1), 6(1+1))$. When k is odd and the topology includes both direct and ring loops, the number of domains can be increased from $P+1$ to $2(P+1)$, under a perfect schedule, by adding a pipeline register on one of the links of the k-node ring—thus delaying the flits for one more cycle and making the effective number of hops in the ring to become even.

The constraints derived by the direct (self-loop) connections between two neighboring routers and the wrap-around ring connections cover the majority of network topologies. However, in some cases, another connectivity pattern exists, similar to the one shown in Fig. 7. In this case, router A can reach router R following a multihop path through router B,

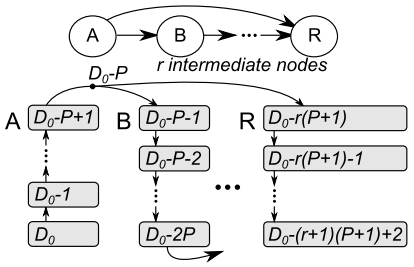

Fig. 7. Zero-latency phase propagation on a reconvergent fan-out connection between routers A and R.

or directly, using the express channel. When the first pipeline stage of router A serves domain D_0, then, at the same time, its output link is forwarding flits that belong to domain $D_0 - P$, which should be served by the first pipeline stage of B in the next cycle. Thus, node B must start by serving domain $D_0 - P - 1$ [similar to Fig. 3(a)]. In a similar manner, the first pipeline stage of router R, which is placed r nodes away, should be operating on domain $D_0 - rP - r$, in order for the flits of A (that reach R via B) to experience zero-latency overhead. However, A can also reach R directly. Therefore, in order for the two paths between A and R to be in-sync, and, thus, prohibiting any latency overhead, the equality $D_0 - P - 1 \bmod D = D_0 - rP - r \bmod D$ must hold. This equality is satisfied when

$$(r-1)(P+1) \bmod D = 0. \tag{3}$$

Equation (3) dictates that D should be equal to any of the factors of $(r-1)(P+1)$. If the condition does not hold, flits that arrive at the inputs of R from one of the two paths will have to wait for their phase to arrive.

The conditions derived suffice to describe the connectivity patterns of any network topology, and how those patterns affect the number of application domains that can be supported by PhaseNoC, in order to allow flits to propagate without any latency overhead irrespective of the path they take in the network. The reason for this powerful attribute (i.e., no dependence on the path taken within the NoC) is due to PhaseNoCs operation, which ensures that all outgoing links of a router serve the same domain in a given cycle, following a perfect schedule. Thus, any adaptive routing strategy can freely select any output of the router and still enjoy in-phase flit propagation. In topologies, either regular or irregular, where the three connectivity patterns (or a subset of them) co-exist, the number of domains that can be supported under a perfect schedule is decided by the greatest common divisor of the number of domains given by conditions (1)–(3). Once the number of domains is selected appropriately, and the phases are distributed on the links and the routers of the topology, a flit can flow uninterrupted from source to destination without experiencing any delay in any part of the network, and irrespective of the path that it follows, except: 1) the delay arising from contention with competing flits of the same domain and 2) the time spent at the network interfaces while waiting for the turn of their application domain to come.

B. Extending the Number of Application Domains

PhaseNoC can support—by construction—an arbitrary number of domains; explicit pipelining does not limit in any

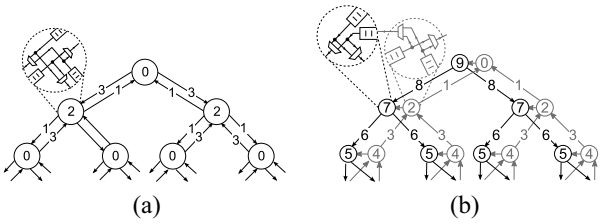

Fig. 8. Phase propagation of (a) four domains in a tree topology and (b) ten domains in a partitioned tree topology (up and down connections separated).

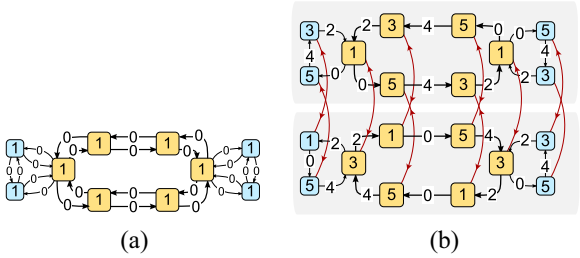

Fig. 9. Phase propagation of (a) two domains in a hierarchical ring topology and (b) six domains in each subnetwork of a partitioned version of the same topology [i.e., the hierarchical ring topology of (a) is partitioned, by separating the left and right directions]. Phases across subnetworks are not aligned.

way the number of active domains. However, at the network level, and depending on whether the number of domains are aligned to the identified topology constraints, the distribution of phases on the links and the routers of the NoC may lead to: 1) either a perfect schedule with zero-delay overhead or 2) it will inevitably favor some paths relative to others, in terms of delay. When we cannot align the number of domains to match the conditions required for perfect scheduling, we can follow another approach, which enables the scheduling of a larger number of domains with a controllable latency overhead.

More domains can be supported after topology partitioning, which aims to remove one or more of the topology constraints. For example, the tree topology shown in Fig. 8, which consists of single-cycle routers, can host at most four domains due to the self-loop constraint in each bi-directional link [condition (1)]. The self-loop constraint can be broken if we separate in different subnetworks the up and down connections. In this way, the number of domains that can be supported under a perfect schedule by PhaseNoC is unlimited in each subnetwork, since there is no loop that would limit the number of supported domains under a perfect schedule. Fig. 8(b) examines the case of ten domains.

In other cases, even though partitioning significantly increases the number of domains that can supported under a perfect schedule, said number is not unlimited. For example, the hierarchical ring built from single-cycle routers shown in Fig. 9(a) can host at most two domains (the greatest common divisor of the number of domains due to the self-loop, the three-node loop of the local rings, and the six-node loop of the global ring). After partitioning the rings, by separating the left and right directions, the constraints are more relaxed and six domains can be hosted, as shown in Fig. 9(b).

Perfect scheduling holds inside each subnetwork and does not extend across subnetworks. Once a flit finds its domain active, it will move uninterrupted until it has to change direction and turn into the other subnetwork. In order for the flit to turn, the appropriate phase should arrive. Assuming, in the general case, that the same D-slot schedule is running in both subnetworks, the flit may need to wait at most D cycles for its domain to arrive, since the schedules in each subnetwork are independent. However, this will happen at most once in the flit's path; whenever the flit moves from one subnetwork to the other.

This strategy can be followed in any topology, thus extremely simplifying the perfect scheduling requirements. For example, a 2-D mesh can be split in two subnetworks, one traversed by flits moving in the $X+/Y+$ directions, and the other one by flits moving in $X-/Y-$. In that case, flits would experience latency only when turning from $X+$ to $Y-$, or $X-$ to $Y+$, assuming XY routing. A partitioned 2-D mesh is the only topology applicable to SurfNoCs schedules [3]. On the contrary, in this paper, we have identified the constraints for zero-latency phase propagation in arbitrary topologies, showing also how the aforementioned constraints can be relaxed by selectively breaking the loops of the topology. Finally, although TDM NoCs that apply TDM-based contention-free routing (per-output SA2 arbiters are replaced by TDM wheels)—such as [8]–[10]—do not face any topology restrictions, the actual topology constraints appear implicitly as design-time scheduling inefficiencies that lead to low TDM slot usage.

C. Weighted VC-Level TDM Schedules

The bandwidth of the network may be unevenly distributed across application domains—in a programmable manner—by just changing the TDM schedule of phase activation in each PhaseNoC router. When each domain receives an equal share of the network's bandwidth, its phase is activated once every D cycles (the period of the TDM schedule is exactly D cycles), while, at the same time, it enjoys zero-latency traversal, as facilitated by the phase propagation in the network. When the bandwidth is not equally shared across domains, TDM scheduling expands to more than D cycles.

One time-frame of bandwidth allocation (i.e., VC-level time slots) consists of S TDM subperiods of D cycles each. The minimum bandwidth that can be allocated is one slot every $S \times D$ cycles. The value of S determines the granularity of bandwidth allocation. The minimum value of S is equal to $S_{\min} = \lceil (1/B_{\min} D) \rceil$, where $B_{\min}$ represents the smallest bandwidth given to any of the application domains, and it is normalized to the maximum bandwidth of 1 (i.e., 100%). On the other hand, the maximum value of S can be arbitrarily large to enable very fine-grained bandwidth allocation. For practical purposes, we can compute S assuming that $B_{\min}$ is the smallest nonzero difference between the bandwidth allocations to the various domains. This ensures sufficiently fine granularity in the allocation.

For example, assume the case of four application domains ($D = 4$), which are programmed to receive 0.29, 0.15, 0.36, and 0.20 of the total bandwidth, as illustrated in Fig. 10. In this case, $B_{\min} = 0.15$ (the minimum allocated bandwidth quota), so the value of $S_{\min} = 2$ can be used. The corresponding application domain should receive one time slot every eight cycles, which is less than the bandwidth required. Also, by using only two TDM periods, we will not be able to approximate closely the rest user-defined bandwidth ratios. On the contrary, and as previously mentioned, we can use the

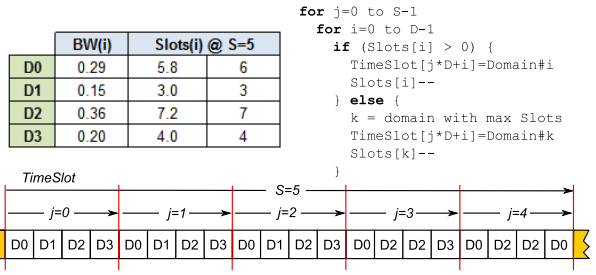

Fig. 10. Example of the construction of a weighted TDM phase schedule, which enables uneven allocation of bandwidth to the supported domains.

minimum nonzero bandwidth difference across the domains to improve the granularity of bandwidth allocation. This minimum difference is equal to $0.20-0.15=0.05$, which requires $S=\lceil(1/(0.05\times 4))\rceil=5$ TDM periods to achieve finer granularity. In this case, $S\times D=20$ time slots must be periodically allocated to four application domains (see Fig. 10).

Each domain should receive $\mathrm{BW}(i)\times S\times D$ VC-level time slots, where $\mathrm{BW}(i)$ represents the user-defined bandwidth share of the ith domain. In the example of Fig. 10, domains should receive 5.8, 3, 7.2, and 4 time slots, according to their required bandwidth share. When the number of VC-level time slots is not an integer, then—inevitably—one domain will receive one additional time slot, while another domain will receive one less time slot. Many rules can be applied to guide the distribution of the left-over time slots to the various application domains. In our example (Fig. 10), we select to round the time slots to the closest integer. Consequently, domain $D0$ will receive 0.30 of the bandwidth (instead of the requested 0.29), while domain $D3$ will receive 0.35 (rather than the requested 0.36). If we want the actual bandwidth allocation to approximate more closely the requested bandwidth allocation, we must increase the value of S (thereby increasing the granularity of bandwidth allocation).

The phase activation of the application domains should be programmed to be repeated every $S\times D$ cycles, while still respecting the following two rules of PhaseNoC scheduling.

1) PhaseNoC guarantees that the domains allocated at least $1/D$ of the bandwidth will experience a perfect schedule for this portion of their allocated bandwidth. The additional VC-level time slots given to any domain (on top of $1/D$) will be out-of-phase, but without any latency penalty, since the domain will be activated more than once every D cycles in some TDM periods.
2) The domains receiving a bandwidth less than $1/D$ will operate partially in-phase, since their domain will not be activated in all TDM periods; thus, more than D cycles may elapse between two consecutive activations of the low-bandwidth domain.

The allocation algorithm, illustrated in Fig. 10, maps the predetermined number of time slots (slots[i] for the ith application domain) to the $S\times D$ time slots of the bandwidth allocation time-frame. As long as a domain has not consumed all of its available time slots, it receives a time slot that is either fully, or partially, in-phase. A domain is always scheduled in-phase, as long as it requires one time slot in all TDM subperiods (i.e., its requested bandwidth is at least $1/D$). When a time slot is empty—since it belongs to a low-bandwidth domain that has used all of its slots—the slot is assigned to an out-of-phase domain (but with shorter delay) that has the maximum number of remaining slots.

IV. Boosting Performance With VA Concurrency and Opportunistic Bandwidth Stealing

PhaseNoCs scheduling approach leads to a static (albeit programmable) allocation of bandwidth to domains. Using the VC-level TDM scheduling mechanism, PhaseNoC guarantees that packets injected in one application domain cannot affect the delivery of packets in a different application domain. More specifically, due to the strict schedule, each domain will get a static share of the network's bandwidth, irrespective of the traffic flowing in other domains. Even if a "rogue" flow tries to flood the network, all other packets belonging to different domains will not be affected at all, both in terms of throughput and latency. Each domain is completely impervious to interference, and PhaseNoCs secure-grade isolation ensures that no information leaks across domains, i.e., a domain cannot infer anything pertaining to any other domain's traffic intensity. The downside of these strict guarantees that is typical for any architecture that offers strict network traffic isolation, lies in the fact that a domain cannot enjoy more bandwidth than the portion preallocated to the domain in the static schedule. This is true even if some domains use fewer of their available time slots at any given time.

When strict isolation guarantees are not necessary, it would be beneficial to reclaim unused cycles to decrease latency and improve throughput. Moreover, by facilitating some form of bandwidth "stealing" across domains, the NoC would be able to accommodate sudden traffic bursts and maintain work-conserving transmission of data. Toward this end, we present two light-weight modifications to the PhaseNoC micro-architecture that work synergistically: 1) to minimize unwarranted latency overheads and 2) to "recycle" any bandwidth left unused by the domains. The latter feature may sacrifice the secure-grade noninterference guarantees, but it still provides complete performance isolation to each domain. In other words, a domain may now be able to deduce the traffic intensity levels in other domains (i.e., information leak is now possible), but each domain is still provided with at least its preallocated performance guarantees, irrespective of the traffic intensity in other domains. Depending on the system's characteristics, PhaseNoC can be configured to support either strict noninterference, or performance-only isolation.

A. Phaseless VC Allocation

The first modification targets the latency overhead incurred due to the per-packet operations. For instance, the VA pipeline stage concerns only the head flit of each packet. Thus, if we provide time slots for VA to all flits in the network, said time slots will be left unused. This becomes especially pronounced in pipeline setups where VA is executed in different pipeline stages from SA and the nonhead flits are coerced to traverse the VA stage without doing any actual work.

Therefore, a phase-less VA would be preferable, which would not consume any valuable slots in the TDM schedule, and which would be executed only by head flits. In order to enable such optimization and achieve maximum utilization, packets from multiple domains must be allowed to concurrently execute VA, while still guaranteeing that the VA operation of any packet does not impact any other packet in a different domain. In a baseline VA unit, interference among input VCs can only occur in the VA2 stage, if multiple input VCs request the same output VC. Note that no interference

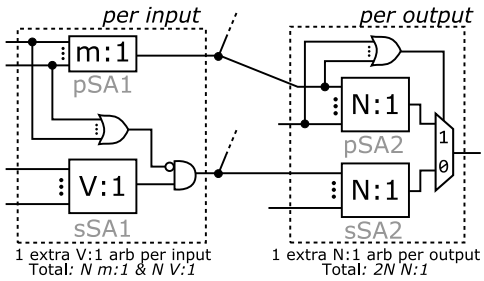

Fig. 11. Modified SA architecture employed by PhaseNoCs OBS mechanism to enable the opportunistic stealing of idle (unused) output port slots by out-of-phase flits.

occurs in VA1, since the latter is simply a selection process executed independently by each input VC. If each output VC is only allowed to be requested by input VCs belonging to a single domain, then no two domains can interfere (arbitrate against each other) in VA2. Therefore, input VCs from different domains are prohibited to request the same output VC. This restriction constitutes a sufficient condition to allow input VCs from multiple domains to perform VA in parallel, without any cross-domain interference.

To enable concurrent execution of VA by multiple domains in each cycle, the simplifications to the VA allocators outlined in Section II must be relinquished in favor of a baseline VA unit that would allow more head flits per cycle to allocate an available output VC of the same application domain.

B. Opportunistic Bandwidth Stealing

The second modification to the baseline PhaseNoC architecture is the augmentation of a mechanism called OBS. This technique optimizes the SA stage by aiming to maintain all output ports of the router busy in each cycle, whenever the active domain would otherwise leave some output ports unutilized. The OBS mechanism observes which output ports are left unused in each cycle by the active domain. All head-of-line flits belonging to the currently active domain are known as in-phase flits. The unused output ports in each cycle are thereby used by out-of-phase flits, i.e., flits belonging to other (inactive) domains. Hence, any bandwidth left unused by in-phase flits in each clock cycle is opportunistically "stolen" by out-of-phase flits. To achieve this without violating performance isolation properties, the new policy must comply with the following twofold rule, which strictly prioritizes in-phase flits over out-of-phase ones: an out-of-phase flit is allowed to participate in SA if: 1) there is no in-phase flit eligible to participate in SA in the out-of-phase flit's input port and 2) there is no in-phase winner heading to the out-of-phase flit's destined output port.

In order to implement this double-faceted rule with a reasonable implementation overhead, we introduce a secondary SA (sSA) unit, which handles switch requests from out-of-phase flits independently. The sSA unit operates in parallel with the primary SA (pSA) unit, which still operates exactly as described in Section II and serves requests from flits of the active domain (i.e., in-phase flits).

Similar to a separable input-first SA implementation, the sSA module consists of two stages of arbitration, as illustrated in Fig. 11. At the input side, sSA1 serves the requests from VCs that do not belong to the active domain, but have already been allocated an output VC (with at least one free buffer slot). The winner of sSA1 is allowed to proceed to the second stage, only if pSA1 has not declared any in-phase winner for that input port. As shown in Fig. 11, the grants of sSA1 are masked, when at least one request was made to the local pSA1 unit. Output port requests from the winners of sSA1 reach the second arbitration stage, sSA2, which declares an out-of-phase input port winner. However, the final winner for every output port is decided after checking the outcome of the corresponding pSA2 unit. If pSA2 has produced a grant, an in-phase flit is eligible to access the specific output port, and it is given priority through the multiplexer on the right-hand side of Fig. 11. In the absence of an in-phase winner, the output port is used by the out-of-phase sSA2 winner.

With OBSs sSA policy, all domains are still guaranteed their assigned TDM schedule slots. However, if these slots are unused in any given cycle, another domain can opportunistically steal them. Essentially, OBS complements strict provisioning of bandwidth guarantees with BE services, in a very efficient and cost-effective manner that allows seamless handling of bursty traffic; when a flow momentarily exceeds its statically allocated bandwidth quota (e.g., due to dynamics in an application's behavior), OBS can absorb the excess traffic without affecting the other domains.

By allowing all out-of-phase flits to always compete for the unused bandwidth of any in-phase domain, OBS inevitably sacrifices the strict two-way isolation of PhaseNoC (information leak can happen across any two domains). However, OBS could easily be modified to forbid out-of-phase flits of particular domains from stealing unused bandwidth from particular in-phase domains, by selectively masking certain out-of-phase requests. Such controllability would enable configurable and directional (one-way) isolation under OBS, if desired.

V. Experimental Evaluation

A. Hardware Evaluation

The first step in the evaluation of PhaseNoC is the quantitative assessment of the hardware complexity of the proposed designs relative to the state-of-the-art. PhaseNoC can be applied to any topology and any flow-control mechanism. The only part of the NoC that is affected by PhaseNoC is the router architecture, which now operates under explicit pipelining. Therefore, any comparison at the router level directly reflects the overall benefits, or possible overheads, of the PhaseNoC concept at the NoC level. The two-stage pipelined routers under comparison (using look-ahead RC) were synthesized and placed-and-routed using Cadence digital design flow driven by a low-power 45 nm standard-cell library (0.8 V, 125 °C). The router models have been configured to five input–output ports, as needed by a 2-D mesh network, while the flit width was set to 64 bits. The area/delay curves were obtained for all designs, under the same constraints and assuming that each output is loaded with a wire of 2 mm.

Five different architectures are considered: 1) a baseline NoC design with no traffic-isolation, or QoS-provisioning capabilities; 2) the SurfNoC architecture [3] with no input speedup; 3) SurfNoC with input speedup equal to the number of supported domains (SurfNoC-S); 4) the proposed PhaseNoC design providing, secure-grade domain isolation; and 5) the PhaseNoC architecture augmented with the OBS mechanism (PhaseNoC-OBS), which optimizes network performance

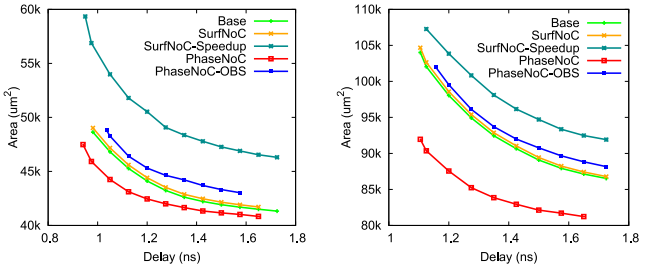

Fig. 12. Area-delay curves for two-stage pipelined routers of baseline, SurfNoC, SurfNoC with input speedup, PhaseNoC, and PhaseNoC-OBS NoC architectures. Two configurations are investigated. (a) Four VCs and (b) eight VCs organized in four domains.

while still guaranteeing complete performance isolation. The SurfNoC-S design follows the original architecture proposed in [3], which requires crossbar input speedup to achieve acceptable performance.

Our goal is to evaluate the benefits arising from the explicit operation of the router in phases, which limits the allocation and switching operations to the input/output VCs of a certain domain. Fig. 12 depicts the area-delay curves for the two-stage pipelined routers under comparison. All routers are tested under two different configurations, shown in Fig. 12(a) and (b), respectively: 1) four VCs with one VC per domain and 2) eight VCs organized as two VCs per domain. For the baseline routers, we assume that the four and eight VCs are grouped in four virtual networks. All routers have four buffers per VC as needed to cover the credits' RTT in two-stage pipelined routers.

At every design point, PhaseNoC routers require the lowest area, while being as fast as the fastest state-of-the art design. More specifically, PhaseNoC occupies up to 20% smaller area than the state-of-the-art SurfNoC design—which employs input speedup—without any delay overhead. Moreover, PhaseNoC requires 13% less area and achieves 5% lower delay than a baseline NoC. The reported savings are mostly a result of the complexity reduction in the allocation units. PhaseNoCs allocation is always limited to the input VCs of one domain, which allows both sharing of the arbiters, and a reduction in their logic depth. Increasing the number of domains will increase the savings accordingly.

PhaseNoC-OBS adds a minimal (less than 6%) overhead in area and delay, as compared to the baseline, while still being more area efficient than SurfNoC with input speedup. PhaseNoC-OBS uses the same VC allocation logic as baseline designs, while it employs a secondary switch allocator that operates in parallel to the main one (to opportunistically "steal" the unused time slots of the VC-level TDM schedule). SurfNoC without input speedup behaves almost identically to the baseline. However, it offers the worst network performance, as will be shown in the following section.

B. Network Performance Evaluation

The goal of this section is to evaluate the effectiveness of PhaseNoC, in terms of network performance, as compared to conventional VC-based NoCs and SurfNoC architecture [3]. Moreover, the traffic isolation properties of the various architectures are also investigated.

TABLE I
SYSTEM PARAMETERS FOR THE FULL-SYSTEM SIMULATIONS

Processor	64 in-order x86 cores @ 1 GHz in a tiled CMP architecture
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, 1-cycle routers (+1 cycle on the link) @ 1 GHz, XY Routing, 6 VCs per input port (3 flits/VC)

In the first set of experiments, we simulate a 64-core tiled CMP system running real application workloads on a Linux operating system. The simulation framework employs Simics [11]—which handles the functional simulation tasks—extended with the GEMS [12] that provides a detailed timing model of the memory hierarchy and it includes the GARNET [13] cycle-accurate NoC simulator.

Table I shows the full-system simulation parameters. We assume that the system operates on two domains. The first domain executes a multithreaded application from PARSEC benchmarks [14], while the other domain receives synthetic GPU traffic. Three VCs are assigned to the domain serving the PARSEC application, and three other VCs to the domain serving the synthetic GPU traffic. This VC setup was dictated by the MOESI cache-coherence protocol, which requires at least three virtual networks to prevent protocol-level deadlocks. Each VC has a depth of three flits, which is adequate to cover the credits' RTT in single-cycle routers. The synthetic GPU traffic is modeled based on real GPU application behavior, as described in [15], and exhibits the many-to-few-to-many traffic patterns observed in GPU applications. This environment, which simultaneously handles both CPU and GPU traffic, abstractly mimics heterogeneous multicore processors, which integrate both CPU and GPU cores on the same die.

The five architectures under comparison are configured to support two domains: 1) for the PARSEC applications and 2) for the synthetic GPU traffic. Each domain receives 50% of the offered bandwidth. The injection rate of the synthetic GPU traffic is 0.054 flits/node/cycle, which is, in fact, rather high, because the many-to-few-to-many traffic patterns create severe hot spots in the network [15].

Fig. 13(a) shows the average network latency of the various PARSEC applications, normalized to the baseline NoC. Recall that GPU traffic also traverses the NoC, concurrently with the PARSEC application traffic. As expected, the baseline router provides the best performance, albeit with no isolation guarantees. SurfNoC provides complete isolation (noninterference) to the PARSEC applications, but at a significant performance cost (45% latency increase, on average). When SurfNoC employs input speedup, this cost decreases to 19% on average, but with a substantial area overhead, as shown in Section V-A. On the contrary, PhaseNoCs performance is substantially better than SurfNoC and close to SurfNoC with input speedup, while still providing complete noninterference. This performance improvement is attributed to the efficiency of PhaseNoCs zero-overhead phase scheduling. On average, PhaseNoCs latency overhead over a baseline NoC is 16%. Finally, PhaseNoC-OBS, which sacrifices secure-grade isolation for performance-only isolation, provides near-identical performance as the baseline NoC.

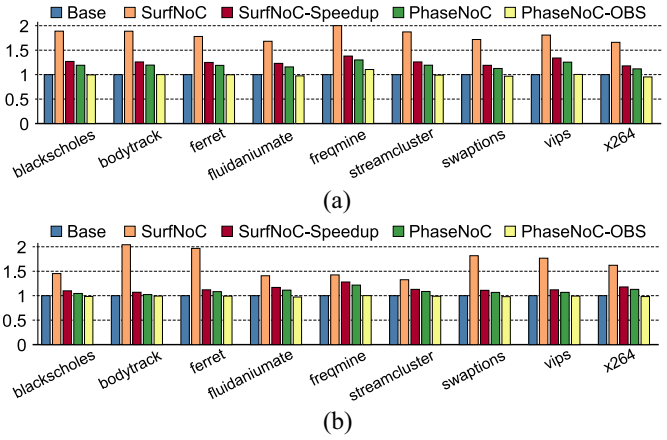

Fig. 13. Full-system simulation results using PARSEC applications [14]. (a) Average network latency. (b) Total execution time.

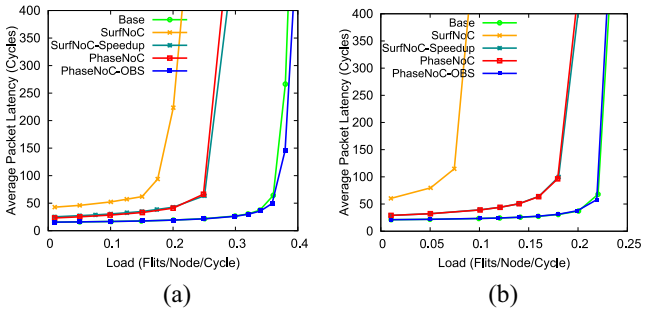

Fig. 14. Average packet latency as a function of the total load for (a) UR and (b) bit complement permutation traffic.

Similarly, Fig. 13(b) shows the total execution time of the various PARSEC applications, normalized to the baseline NoC router. The trends are similar to those of the average network latency, but PhaseNoC is now only 8% slower, on average, than the baseline NoC. This indicates that the relatively small degradation in average network latency sustained under PhaseNoC does not greatly impact the total execution time (due to the fact that a lot of the packets exhibit large slack [16]). Once again, PhaseNoC-OBS offers near-identical performance as the baseline NoC (the differences are within simulation-noise margins). Overall, both PhaseNoC incarnations yield execution times that are very close to the baseline, with significantly less area than baseline and SurfNoC designs.

Despite the authenticity provided by full-system simulations, the flexibility to stress the NoC and isolate its inherent attributes is somewhat limited, due to the fixed characteristics of the running applications. In order to differentiate more clearly the various architectures, we henceforth resort to synthetic traffic patterns (for the remaining experiments), by operating Garnet in a "network-only" mode. Various configurations are explored, while the network size, routing algorithm, and buffer depth are the same as in the full-system simulation configuration. Synthetic traffic consists of 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 request packets, and the rest being long, five-flit reply packets.

In the first set of experiments, we examine the average network performance characteristics of the architectures under comparison. Fig. 14(a) depicts the load-latency curves of PhaseNoC architectures, baseline NoC (Base), and SurfNoC variants, assuming single-cycle routers with four VCs per port, which are distributed to four domains (one VC per domain), under uniform-random (UR) traffic. The baseline NoC, even if not providing any isolation properties, assumes that each VC is a virtual network, thus in-flight VC changes are prohibited. The PhaseNoC and SurfNoC variants are driven by a static 4-phase TDM schedule that gives an equal share of the available bandwidth to each domain. The same setup holds in Fig. 14(b), which repeats the same experiment using the bit-complement permutation traffic pattern.

From both figures, we can deduce that the zero- or low-load-latency of PhaseNoC, even after the strict TDM operation at the VC level, is still very close to that of the baseline design, due to the efficient network-level schedule, which allows the domino-like propagation of flits without any additional latency (independent of the source and destination, the topology, or the routing algorithm). In terms of saturation throughput, PhaseNoC offers lower throughput than the baseline, due to the noninterfering operation across domains, which reduces—in some sense—the choices for flit interleaving. Note that this difference is not observed in the real applications of Fig. 13, because the injection rates of said applications are very low. This throughput loss is inevitable, since there is no way for the time slots allocated to one domain to be used by another domain. On the contrary, PhaseNoC-OBS manages to fully recycle all unused time slots. SurfNoC without input speedup provides the worst performance. The wave propagation in SurfNoC adds additional latency when moving from a south-east direction to a north-west one. Further, the absence of crossbar input speedup markedly increases SurfNoCs latency overhead. On the contrary, SurfNoC with input speedup improves the performance dramatically, thereby yielding results that are near-identical to PhaseNoC. However, this performance comes at a steep hardware cost: an 20% area overhead relative to PhaseNoC. The PhaseNoC-OBS design substantially outperforms both SurfNoC variants and achieves performance that is on par with the baseline NoC.

To evaluate the effect of the number of application domains on latency and hardware area, we conducted an additional experiment. Fig. 15(a) shows the average packet latency of PhaseNoC and SurfNoC networks using four-stage pipelined routers at an injection rate of 0.01 flits/node/cycle using UR traffic, when varying the number of application domains. In terms of latency, PhaseNoC and SurfNoC with speedup show equivalent performance, both experiencing a controlled latency overhead. On the other hand, PhaseNoC-OBS can "absorb" any extra delay due to phase nonalignment, thus reducing the criticality of scheduling and phase alignment. Fig. 15(a) does not capture the impact of supporting additional domains on hardware cost, which is very high for SurfNoC with input speedup (as previously analyzed). To capture the interaction between latency and area, we employ the combined metric of $(\text{Latency}_{\text{design}}/\text{Latency}_{\text{base}}) \times (\text{Area}_{\text{design}}/\text{Area}_{\text{base}})$. Under this metric (lower is better), both PhaseNoC variants exhibit the best performance, as shown in Fig. 15(b). Specifically, PhaseNoC outperforms SurfNoC with input speedup by 18%, in terms of this combined metric.

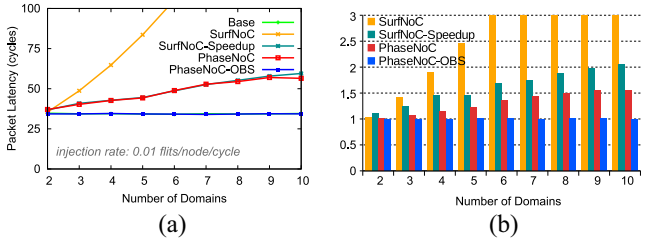

Fig. 15. Comparison of PhaseNoC and SurfNoC networks using four-stage pipelined routers when varying the number of application domains, in terms of (a) average packet latency and (b) normalized latency × normalized area (the normalization is to the baseline NoC).

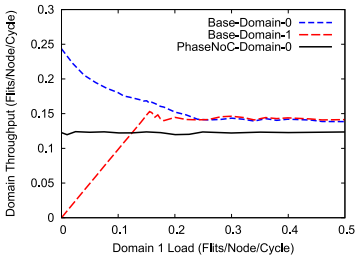

Fig. 16. Noninterfering isolation of traffic in PhaseNoC. Unlike a baseline NoC, traffic in VC0 (i.e., domain 0) of PhaseNoC is constant, irrespective of the load behavior in the other domains.

Note that the support of additional domains implies additional area cost, due to the presence of additional VC buffers. Hence, PhaseNoCs area efficiency over SurfNoC with input speedup (20% lower area) allows for the support of additional domains. Conversely, assuming equal router-area budget, PhaseNoC can support more domains than SurfNoC.

PhaseNoC variants achieve high network performance, while still offering isolation properties that are not provided by a baseline NoC. The first scenario that shows the isolation properties of PhaseNoC, involves a two-domain network, where each domain has one VC, in both PhaseNoC and a baseline NoC. The traffic on both VCs is UR, while the load in VC0 remains constant and the load in VC1 is progressively increased. As shown in Fig. 16, the load in VC0 of PhaseNoC is completely immune to the changes in VC1s load. However, in the baseline case, as the load in VC1 increases, the accepted throughput of VC0 (domain 0) drops and converges to that of VC1 (domain 1).

In the second scenario, we evaluate the bandwidth guarantees offered by PhaseNoC when enhanced with the OBS mechanism, as compared to a baseline network without any guaranteed services (GSs). We simulate a network with two domains, each one utilizing a single VC, which are used to serve two different traffic classes: 1) BE and 2) an urgent guaranteed-bandwidth (GB) service. To provide such services, PhaseNoC with OBS operates under a schedule that offers 99% of the slots to the GB domain, leaving only the remaining 1% to BE. In this way, GB traffic is always guaranteed high priority, while BE can still enjoy more than 1% (as long as GB is idle), in order to deliver the BE service. In this experiment, BE is serving a constant load of UR traffic, while GB remains idle most of the time and is only used to inject urgent burst traffic, once during the simulation time: on the 5Kth cycle, 12 random nodes initiate a 5K-cycle burst traffic, each one toward a separate destination node, using the GB domain.

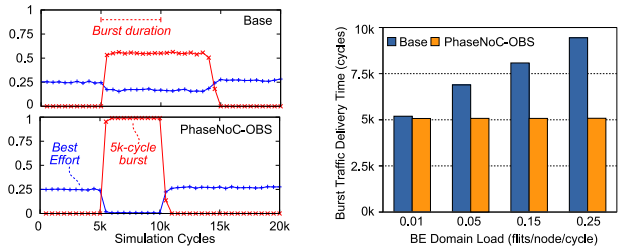

Fig. 17. Demonstration of the flow isolation properties of PhaseNoC-OBS. The GB domain in PhaseNoC-OBS is completely immune to the load of the BE domain, and always manages to deliver the GB traffic in time. (a) Real-time serving of GB traffic. (b) Sensitivity to BE load.

Fig. 17(a) illustrates both domains' accepted throughput during the simulation period, whereby the BE load is set to 0.25 for both baseline (Base) and PhaseNoC-OBS networks. Before the bursts are initiated, the BE traffic in PhaseNoC-OBS "steals" the idle domain's slots, and is able to deliver the same throughput as the baseline network. Once the burst injection starts, the urgent traffic (GB) is offered higher priority and immediately receives 99% throughput, in contrast to the baseline network, where the traffic is equally shared among the two domains, without any prioritization. Immediately after the burst stops, PhaseNoC-OBS has delivered the urgent traffic in time, and the BE domain immediately recovers. On the other hand, the recovery period in the baseline network is almost as long as the duration of the burst, since urgent traffic has not been ejected yet, and occupies network resources long after the burst is over. The total delivery time of the whole 5K-cycle burst, under various UR loads in the BE domain, is shown in Fig. 17(b). Notice that the GB domain in PhaseNoC-OBS is completely immune to the load of the BE domain, and always manages to deliver the load in time, as needed for real-time traffic. In any case, in PhaseNoC-OBS the TDM prescheduled time slots at the VC level are gracefully combined with BE output port utilization to provide the best of both worlds.

VI. Related Work

At first glance, PhaseNoC may appear to be yet another facilitator of QoS guarantees. However, the proposed design is distinctly differentiated from existing techniques by its ability to guarantee full, secure-grade isolation between the supported domains. While PhaseNoC can also provide typical QoS guarantees and differentiated services among the supported domains, its key contribution lies in the mechanism's ability to completely isolate the various domains. PhaseNoC provides a TDM schedule at the VC level governing the noninterfering service across domains. The service of communication flows within each domain could potentially be provided by any existing QoS technique. Hence, the PhaseNoC architecture is orthogonal to (and can benefit from) current mechanisms supporting differentiated services.

A. Priority-Based Scheduling

Existing NoC techniques employing priority-based scheduling as a means to provide QoS guarantees were inspired by rate-based techniques in multicomputer networks, such as weighted fair queueing [17], virtual clock [18], and rotating

combined queueing [19]. Similarly, age-based arbitration prioritizes packets based on their age (oldest first) [20], while source throttling [21] can regulate flow rates to facilitate congestion control. Rate-based schemes employ fine-grained scheduling of packets, which incurs high implementation cost that is often prohibitive for on-chip implementations.

Hence, researchers have developed on-chip-amenable derivatives of rate-based approaches. Two examples are globally synchronized frames (GSFs) [22] and preemptive virtual clock (PVC) [23], which utilize a coarse-grain rate-based philosophy based on the notion of time epochs, or "frames" [24]. By coarsening the granularity of bandwidth allocation, frame-based techniques reduce the hardware cost and scheduling complexity. To improve the efficiency of frame management, GSF [22] adopts an additional dedicated "barrier network," which operates alongside the main NoC. Nevertheless, GSF still suffers from significant throughput degradation, as compared to a baseline router with no QoS provisioning. PhaseNoC could be augmented with GSF functionality (if desired), since each "frame" would correspond to a PhaseNoC domain. The GSF mechanism would control the traffic injection at the NIs, while GSFs barrier network would monitor the global drainage of each domain. Similarly, nonlinear weighted arbitration [25] can be incorporated within a PhaseNoC domain to shape the traffic within each domain and provide equality of service among the flows of a domain.

Unlike GSFs reliance on multiple active frames, the PVC mechanism [23] employs only a single active frame at a time. However, PVC also relies on an additional network, which facilitates packet preemption (i.e., packet dropping and retransmission). In general, PVC suffers from higher implementation complexity than GSF, and it achieves markedly lower throughput than a baseline NoC. Furthermore, the PVC mechanism provides limited scalability with system size, as acknowledged in the topology-aware QoS schemes of [26].

Stall-time criticality is exploited to enable application-aware prioritization in the NoC [27]. This technique improves system throughput, with no QoS guarantees. Similarly, Aérgia [16] prioritizes in-flight packets based on how they affect the application's execution time. LOFT [28] combines a frame-based approach and flit-reservation flow control [29] to improve flow robustness and network utilization over GSF. QNoC [30] provides four QoS service levels and uses preemptive scheduling to prioritize higher service levels. Per-priority-class buffering and preemptive scheduling at the flit-level are characteristics of any priority-based scheduling approach to avoid priority inversions. CoQoS [31] enables coordinated QoS management of the cache, NoC, and memory. MANGO [32] classifies VCs into two separate classes and implements statically programmed connections and reserved VCs/paths. Centrally managed VC-level connections between any communicating pair are used in [33].

B. TDM-Based Scheduling

The vast majority of the TDM-based designs perform TDM scheduling at the time-slot level [8], [9], [34]. The scheduling is typically performed offline (assuming *a priori* knowledge of the applications expected to be running on the system), since online scheduling is normally slow [35], and then statically applied to the NoC. Argo [34] allows schedules to evolve at the granularity of a single cycle, even when the routers have more than one pipeline stage. The resulting hardware cost is quite low, but the latency overhead can be substantial.

Æthereal [8] employs pipelined TDM (at the time-slot level) and circuit-switching to guarantee performance services. Traffic is separated into two main classes: 1) GS and 2) BE. Excess bandwidth not used by GS flows is given to BE flows. Packets on a single connection are always ordered, but ordering cannot be enforced between connections. The utilization of excess bandwidth by other domains in PhaseNoC is facilitated by OBS. The SuperGT NoC [36] is an evolution of Æthereal providing three QoS classes. Aelite simplifies the router architecture by providing only GS [9], and dAElite moves one step further by including multicast traffic and fast virtual-circuit setup [10]. Weber *et al.* [37] provided run-time programmable bandwidth guarantees by regularly injecting epoch packets. Finally, Nostrum [38] and Lu and Jantsch [39] employed a form of TDM and virtual circuits to allocate bandwidth and satisfy QoS guarantees.

The overall simplicity of TDM-based NoCs, and their useful properties in providing QoS guarantees, are normally contrasted by the rather high buffering requirements. For example, TDM-based NoCs require virtual-circuit buffers (i.e., per connection) at least in the network interfaces, whereas PhaseNoC only requires buffers per application domain. In the same context, PhaseNoC-OBS provides a practical approach for mixing BE and GS traffic, without resorting to additional virtual-circuit buffers for BE connections [40]. PhaseNoC reuses VC buffers that normally refer to link-level flow control to statically define application domains as a generic mechanism for flow isolation. This reduces the need to dynamically set and tear down virtual circuits. The overall buffer savings expected from the use of VC buffers as isolated application domains—as opposed to using virtual circuits for the same purpose—is expected to be significant. However, the exact savings are hard to quantify, since they depend on the number of application domains and the maximum number of virtual-circuit connections required for sufficient performance. In this paper, we treat application domains as virtual networks, which allow the sharing of all network resources by different applications (e.g., CPU and GPU traffic). As future work, we plan to explore different mappings for the supported domains.

Additionally, PhaseNoC can be used in conjunction with TDM-based NoCs that apply TDM scheduling at the time-slot level. We believe that the TDM schedules derived for applying contention-free packet routing can be derived independently of their bandwidth guarantees, whereas the actual bandwidth allocation can be imposed at the VC-level by the PhaseNoC architecture. In this case, the algorithmic complexity of traditional TDM scheduling [41] is expected to be lower. This extension to PhaseNoC is also planned as future work.

C. Avoiding Flow/Traffic Interference in NoCs

One way to avoid interference among flows is to physically separate them in distinct networks. For example, the Tile64 iMesh separates user application traffic from OS and I/O traffic into separate physical networks [42]. Traffic isolation is also required in congestion management [43], [44]. ICARO [45] uses two virtual networks to isolate congested traffic from noncongested traffic, in order to avoid head-of-line blocking. PhaseNoC could provide similar functionality if combined with the congestion detection mechanism of [44] or [45].

Nevertheless, all approaches discussed so far may provide some QoS guarantees, but none can guarantee complete flow isolation (noninterference). Many designs provide increased robustness to interference (e.g., PVC), but no absolute guarantees can be given, i.e., QoS properties are insufficient to guarantee noninterference within the network [46].

Guaranteed noninterference has been investigated primarily within the realm of security [47]. In [46], static network partitioning in space and time is employed to provide multiway isolation among the supported domains. This multiway isolation property comes at a high performance cost, which is alleviated by the introduced reversed priority with static limits (RPSL) mechanism, which uses priority-based arbitration and static limits to guarantee one-way isolation between high-security and low-security flows. The RPSL mechanism [46] facilitates bandwidth reuse among domains, just like PhaseNoCs OBS. However, its operation gives rise to fairness/starvation issues. As a result, RPSL requires an additional mechanism to monitor and statically limit the use of each input/output port by particular domains. On the contrary, PhaseNoCs OBS scheme enables bandwidth reuse without suffering from any fairness issues whatsoever.

Efficient multiway noninterference among traffic flows is provided by SurfNoC [3] and TDM-based architectures [8], [10]. Similar to PhaseNoC, SurfNoC also revolves around the concept of propagating waves across the on-chip network and TDM scheduling at the VC level. However, PhaseNoC is distinctly different and offers significant advantages over SurfNoC.

1) PhaseNoCs novel router micro-architecture allows—for the first time—each pipeline stage of the router to deal explicitly with the flits of only one domain.
2) Even though PhaseNoC achieves full isolation, it is less expensive (in terms of hardware) than even baseline designs, due to explicit pipelining, which provides isolation across domains for free and allows for extensive sharing of the allocation logic. On the contrary, SurfNoC routers require expensive crossbar input speedup (of degree equal to the number of supported domains), in order to achieve acceptable performance.
3) Unlike SurfNoC, which always suffers a delay overhead on wave-front changes, PhaseNoC benefits from zero-latency propagation overhead, when satisfying the topology constraints derived in Section III-A. The provided methodology generalizes to any network topology; instead, SurfNoCs scheduling is only applicable to 2-D meshes/tori.
4) PhaseNoC jointly handles optimal scheduling and uneven allocation of bandwidth to the supported domains.
5) If secure-grade isolation is not required, PhaseNoCs OBS mechanism recycles any bandwidth left unused in each cycle to optimize performance.

VII. Conclusion

In addition to providing high performance and scalability, NoCs may also be required to support additional functionality, such as flow isolation, and QoS provisioning. This paper introduces PhaseNoC, a cost-efficient VC-based NoC architecture supporting truly noninterfering multidomain operation. The new design relies on TDM scheduling at the VC-level, which optimally coordinates the multiphase propagation of domains across the NoC. PhaseNoC is a bandwidth-programmable traffic isolation mechanism, which is applicable to any topology upon following the proposed methodology for creating optimal phase schedules. When secure-grade domain isolation is not mandatory, PhaseNoC can also support a more "relaxed" isolation mode, which substantially improves the latency/throughput performance. By utilizing OBS, any unused bandwidth is recycled among the domains. This performance-optimized mode still provides complete performance isolation to each domain. Extensive evaluation using both synthetic traffic and real application workloads validates the efficiency of PhaseNoC, as compared to the state-of-the-art. Finally, detailed hardware analysis verifies the cost-effectiveness of the proposed new architecture.

References

[1] Arteris, "A comparison of network-on-chip and buses," Tech. Rep., 2005.

[2] W. J. Dally, "Virtual-channel flow control," in *Proc. ISCA*, Seattle, WA, USA, 1990, pp. 60–68.

[3] H. G. W. Wassel *et al.*, "SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip," in *Proc. ISCA*, Tel Aviv, Israel, 2013, pp. 583–594.

[4] K. Goossens *et al.*, "Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow," *ACM SIGBED Rev.*, vol. 10, no. 3, pp. 23–34, Oct. 2013.

[5] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*. New York, NY, USA: Springer, 2014.

[6] M. Tiwari *et al.*, "Complete information flow tracking from the gates up," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Hamilton, ON, Canada, 2009, pp. 109–120.

[7] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Select. Areas Commun.*, vol. 9, no. 8, pp. 1265–1279, Oct. 1991.

[8] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design Test Comput.*, vol. 22, no. 5, pp. 414–421, Sep./Oct. 2005.

[9] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *Proc. DATE*, Nice, France, 2009, pp. 250–255.

[10] R. A. Stefan, A. Molnos, and K. Goossens, "dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Trans. Comput.*, vol. 63, no. 3, pp. 583–594, Mar. 2014.

[11] Wind River Inc. *Simics, Product Overview*. [Online]. Available: http://www.windriver.com/products/simics/, accessed Oct. 2015.

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

[13] 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. Int. Symp. Perform. Anal. Syst. Softw.*, Boston, MA, USA, Apr. 2009, pp. 33–42.

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

[15] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *Proc. Int. Symp. Microarchit. (MICRO)*, Atlanta, GA, USA, 2010, pp. 421–432.

[16] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Aérgia: Exploiting packet latency slack in on-chip networks," *ACM SIGARCH Comp. Archit. News*, vol. 38, no. 3, pp. 106–116, 2010.

[17] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM*, vol. 19, no. 4, pp. 1–12, 1989.

[18] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *ACM SIGCOMM*, vol. 20, no. 4, pp. 19–29, Sep. 1990.

[19] J. H. Kim and A. A. Chien, "Rotating combined queueing RCQ: Bandwidth and latency guarantees in low-cost, high-performance networks," *ACM SIGARCH Comp. Archit. News*, vol. 24, no. 2, pp. 226–236, 1996.
[20] D. Abts and D. Weisser, "Age-based packet arbitration in large-radix k-ary n-cubes," in *Proc. Int. Conf. Supercomput.*, Reno, NV, USA, 2007, pp. 1–11.
[21] M. Thottethodi, A. R. Lebeck, and S. S. Mukherjee, "Self-tuned congestion control for multiprocessor networks," in *Proc. High-Perform. Comput. Archit.*, Monterrey, Mexico, 2001, pp. 107–118.
[22] J. W. Lee, M. Cheuk Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *Proc. ISCA*, Beijing, China, 2008, pp. 89–100.
[23] B. Grot, S. W. Keckler, and O. Mutlu, "Preemptive virtual clock: A flexible, efficient, and cost-effective QoS scheme for networks-on-chip," in *Proc. Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2009, pp. 268–279.
[24] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. ACM SIGCOMM*, Zürich, Switzerland, 1991, pp. 113–121.
[25] K. Gwangsun *et al.*, "Low-overhead network-on-chip support for location-oblivious task placement," *IEEE Trans. Comput.*, vol. 63, no. 6, pp. 1487–1500, Jun. 2014.
[26] B. Grot *et al.*, "Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *Proc. ISCA*, San Jose, CA, USA, 2011, pp. 401–412.
[27] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *Proc. 42nd IEEE/ACM Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2009, pp. 280–291.
[28] J. Ouyang and Y. Xie, "LOFT: A high performance network-on-chip providing quality-of-service support," in *Proc. 43rd IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Atlanta, GA, USA, 2010, pp. 409–420.
[29] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proc. High-Perform. Comput. Archit. (HPCA)*, Toulouse, France, 2000, pp. 73–84.
[30] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *J. Syst. Archit.*, vol. 50, nos. 2–3, pp. 105–128, 2004.
[31] B. Li *et al.*, "CoQoS: Coordinating QoS-aware shared resources in NoC-based SoCs," *J. Parallel Distrib. Comput.*, vol. 71, no. 5, pp. 700–713, 2011.
[32] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. Design Autom. Test Europe (DATE)*, vol. 2. Munich, Germany, 2005, pp. 1226–1231.
[33] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *Proc. VLSI Technol. Archit.*, Karlsruhe, Germany, 2006, pp. 211–216.
[34] E. Kasapaki *et al.*, "Argo: A real-time network-on-chip architecture with an efficient gals implementation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2015.2405614
[35] R. Stefan, A. B. Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *Proc. Interconnect. Netw. Archit.: On-Chip Multi-Chip (INA-OCMC)*, Paris, France, 2012, pp. 13–16.
[36] T. Marescaux and H. Corporaal, "Introducing the SuperGT network-on-chip; SuperGT QoS: More than just GT," in *Proc. 44th ACM/IEEE Design Autom. Conf. (DAC)*, San Diego, CA, USA, 2007, pp. 116–121.
[37] W.-D. Weber, J. Chou, I. Swarbrick, and D. Wingard, "A quality-of-service mechanism for interconnection networks in system-on-chips," in *Proc. Design Autom. Test Europe (DATE)*, vol. 2. Munich, Germany, 2005, pp. 1232–1237.
[38] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proc. Design Autom. Test Europe Conf.*, vol. 2. Paris, France, 2004, pp. 890–895.
[39] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 8, pp. 1021–1034, Aug. 2008.
[40] K. Goossens and A. Hansson, "The æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2010, pp. 306–311.
[41] A. Hansson, K. Goossens, and A. Radulescu, "A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic," *VLSI Design*, vol. 2007, 2007, Art. ID 68432.
[42] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.
[43] J. Duato *et al.*, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *Proc. High-Perform. Comput. Archit.*, San Francisco, CA, USA, 2005, pp. 108–119.
[44] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *Proc. 14th IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Salt Lake City, UT, USA, 2008, pp. 203–214.
[45] J. V. Escamilla, J. Flich, and P. J. Garcia, "ICARO: Congestion isolation in networks-on-chip," in *Proc. IEEE Int. Symp. Netw.-on-Chip*, Ferrara, Italy, 2014, pp. 159–166.
[46] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *Proc. 6th IEEE/ACM Int. Symp. Netw. on Chip (NoCS)*, Copenhagen, Denmark, 2012, pp. 142–151.
[47] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for NoCs," in *Proc. Hardw./Softw. Codesign Syst. Synth.*, Atlanta, GA, USA, 2008, pp. 197–202.

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

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

Junghee Lee received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, Korea, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2013.

From 2003 to 2008, he was with Samsung Electronics, Suwon, Korea. Since 2014, he has been an Assistant Professor with the University of Texas at San Antonio, San Antonio, TX, USA. His current research interests include architectural design of microprocessors and storage systems for high-performance computing and embedded systems.

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

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

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

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

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

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. His current research interests include design of digital integrated circuits and computer architecture, with emphasis in network-on-chip design and ultralow power systems.