PhaseNoC: TDM Scheduling at the Virtual-Channel Level for Efficient Network Traffic Isolation

A. Psarras*, I. Seitanidis*, C. Nicopoulos† and G. Dimitrakopoulos*
*Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
†Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

***Abstract*—The efficiency of modern Networks-on-Chip (NoC) is no longer judged solely by their physical scalability, but also by their ability to deliver high performance, Quality-of-Service (QoS), and flow isolation at the minimum possible cost. Although traditional architectures supporting Virtual Channels (VC) 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 non-interfering VC-based architecture that adopts Time-Division Multiplexing (TDM) 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. The resulting design yields significant reductions in the area/delay cost of the network. Experimental results corroborate that – with lower cost than state-of-the-art NoC architectures, and with minimum latency overhead – we remove any flow interference and allow for efficient network traffic isolation.**

I. Introduction

The last decade has witnessed a fundamental paradigm shift in digital system design: the transition to the multi-/many-core realm. Naturally, the multi-core domain has elevated the criticality of the on-chip interconnection fabric, which is now tasked with satisfying amplified communication demands. Owing to their scalability attributes, Networks-on-Chip (NoC) have established their position as the de facto communication medium in multi-core systems. To sustain system scalability into the many-core domain (with potentially hundreds of cores), it is imperative that the NoC's hardware cost is minimized, while not sacrificing network performance [1].

This objective is non-trivial, since the functionality expected from the NoC continues to grow. For instance, multi-core systems increasingly require some form(s) of isolation – or separation – among the traffic flows of concurrently executing applications. Such segregation attributes are desired due to (a) Quality-of-Service (QoS), or real-time requirements, and/or (b) restrictions imposed by higher-level protocols, e.g., cache coherence in Chip Multi-Processors. Separation of flows is typically achieved using Virtual Channels (VC) within the NoC [2]. However, due to the widespread sharing of router resources (ports, arbiters, crossbar, channels, etc.) among all the VCs, non-interference between flows is not guaranteed. In fact, VCs are, by construction, interfering, since multiple VCs are eligible to compete for NoC resources at any given time.

In order to provide true QoS guarantees and performance isolation between concurrent flows, there is an imperative need for *truly non-interfering* VC-based router designs. Hence, the demand for a scalable NoC architecture that can support multiple VCs and provide performance isolation (non-interference) at low cost is highly relevant and pressing.

A very cost-effective approach to non-interference is Time-Division Multiplexing (TDM), which isolates the flows in time. In our context, however, TDM scheduling must be performed at the VC level (not at the time-slot level), since the aim is to provide isolation at the VC granularity. Once a static schedule exists across VCs (or groups of VCs that belong to the same application domain), the transfer of packets/flits in the network is handled using normal flow control, arbitration, and switching policies. Applying TDM-based scheduling at the VC level calls for (a) an efficient static scheduling methodology that does not incur any latency overhead, and (b) a router design that can apply said scheduling, avoid interference, and still cost substantially less – in terms of area/power – than a conventional non-TDM-based architecture. If the aforementioned two requirements are not both satisfied, the decision to use TDM scheduling would be difficult to justify from a performance-cost perspective.

Prior approaches to TDM-based scheduling in NoCs fail to simultaneously satisfy both requirements. Numerous designs perform TDM scheduling at the time-slot level [3–6]. When using such architectures, the scheduling is typically performed offline (and assumes perfect a priori knowledge of the applications expected to be running on the system), and then statically applied to the entire NoC [6]. The resulting hardware cost is quite low, but the latency overhead can be quite substantial [7]. A recently introduced architecture, called SurfNoC, employs optimized TDM scheduling – also applied at the VC level – to minimize the latency overhead [8]. However, the required hardware cost is excessively expensive. Achieving low-cost implementations with SurfNoC would increase the latency overhead of static scheduling. Other, non-TDM-based approaches provide QoS guarantees by controlling flow rates [9, 10]. However, these solutions do not guarantee non-interference among flows; they *try* to mitigate its effect.

Building on the fundamental premise of simultaneously minimizing the latency overhead and significantly decreasing the hardware cost (as compared to non-TDM designs), we propose the ***PhaseNoC*** architecture, which *ensures non-interfering operation* across *domains*. PhaseNoC's ability to provide performance isolation to multiple domains at an extremely low hardware cost emanates not only from the carefully choreographed phase propagation (i.e., the TDM scheduling), but also from a novel and complete overhaul of the routers' micro-architecture and their internal pipeline operation into a phase-amenable organization.

To the best of our knowledge, PhaseNoC is the first architecture to employ a new type of pipelining (called *explicit*

This work is supported by the European Social Fund & the Greek National Strategy Reference Framework 2007-2013 under ΠΑΒΕΤ-2013. I.Seitanidis is supported by the PhD scholarship of Alexander S. Onassis foundation.

pipelining), which is perfectly attuned to TDM-based scheduling at the VC level.

The PhaseNoC approach is shown to be extremely versatile and easily applicable to any topology and many router pipeline depths. The design can support up to 10 domains (i.e., up to 10 VCs, or 10 groups of VCs per input port) in a single physical network, while still guaranteeing perfect phase scheduling for all domains. The handling of even larger numbers of domains necessitates the adoption of multiple physical networks, which constitute a very cost-effective and scalable solution to supporting potentially any number of domains.

Hardware analysis using placed-and-routed designs verify that PhaseNoC's TDM-based scheduling at the VC level yields substantial cost savings, as compared to traditional VC-based router architectures. Moreover, the orchestrated propagation of the active phases across the entire PhaseNoC network ensures that the impact on performance (throughput and latency) – as compared to non-TDM designs – can be kept to a minimum, as demonstrated by extensive cycle-accurate network simulations.

II. The 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 virtual channels. Every input port of the PhaseNoC router hosts V virtual channels 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/arbitration steps. The head flit of a packet first calculates its output port through Routing Computation (RC). It then uses it 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 during Switch Traversal (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) [11].

Non-interference 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 non-interference 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.

When a router is pipelined, it may operate simultaneously on many application domains, by selecting the appropriate phase for each allocation/switching step. Care should be taken to assure that two domains never simultaneously participate in the same step. 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

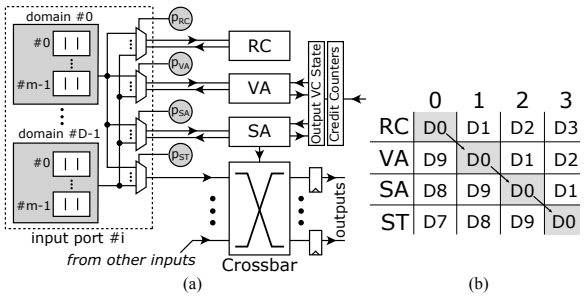


Fig. 1. (a) The PhaseNoC router architecture. Only a set of VCs (one domain) is allowed to participate from each input in each allocation stage, and it is the same set (domain) for all inputs. One multiplexer for each stage, controlled by a TDM schedule, allows different domains to be served by different stages of the router. (b) Cycle-by-cycle operation of a 4-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).

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 towards the next router. Therefore, in each cycle, the router is fully utilized, but each part of the router works on a different domain. This feature of PhaseNoC's 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 Figure 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. Care should be taken so that all inputs see the same order of phase activation.

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

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

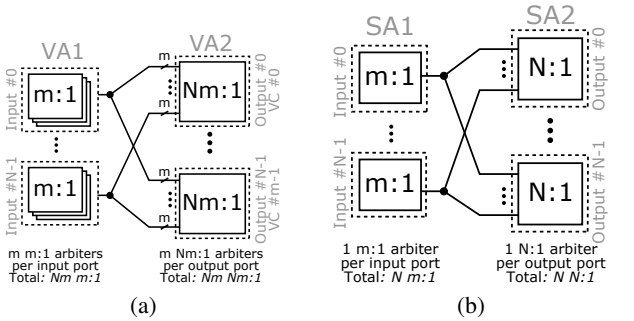


Fig. 2. PhaseNoC's reduced allocators for the (a) VA, and (b) SA router pipeline stages.

and ST are merged in the same stage, they should also share a common phase of operation (and one multiplexer), thus letting the router act on one less domain in parallel.

A. 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 Figure 2(a), VA is performed in 2 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 $N \times m : 1$ round-robin arbiter to at most one input VC of the same domain. Baseline allocators are also simplifed when serving groups of VCs of the same domain. However, multiple arbiters are still needed to serve all domains *simultaneously*.

Similar simplifications can be derived for the switch allocator as well, which, again, involves 2 steps of arbitration, as shown in Figure 2(b). The SA1 arbiter per input 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, cannot be simplified further, and it still requires an $N : 1$ arbiter per-output.

Although both 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.

B. Buffering requirements

In a router configuration using credit-based flow-control, the minimum buffer depth required for each input VC in order to operate under 100% throughput is equal to the Round-Trip Time (RTT). The RTT is the number of cycles that elapse from the instance the downstream router sends a credit until a flit (using that credit in the upstream router) arrives.

In PhaseNoC, a single input VC can only send a flit whenever the router's phase is aligned to its domain. In the simple case of a TDM schedule cycling through all domains in D cycles, the throughput of a single domain cannot exceed $1/D$. Thus, less buffering than the RTT is required, without changing the performance of the network. Other buffer cost reduction techniques that rely on buffer sharing across input VCs cannot be directly applied, unless sufficient buffering space is statically allocated per VC. Dynamic sharing would inevitably introduce interference across VCs; the buffering space allocated to one VC effectively affects the throughput seen by another VC [12].

III. Application-Domain Scheduling and Mapping

The proposed router design guarantees non-interference 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 completely independent of the other domain's load.

An efficient time-multiplexing schedule for the supported application domains should guarantee the propagation of the flits of one domain in the network in a wave-like manner, traversing many hops in consecutive cycles, without waiting for the turn of their application domain to come. In this way, non-interference is achieved with the minimum possible latency of data transfer, since only the flits of the same application domain experience contention throughout the network.

A. Zero-latency-overhead domain scheduling

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.

This property is shown in Figure 3(a), where 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 towards 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

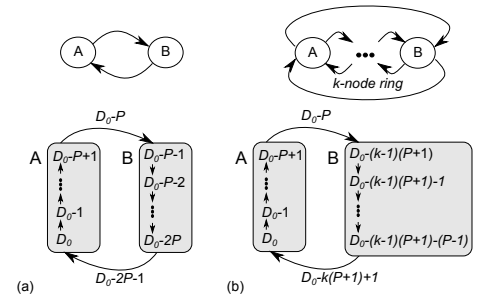


Fig. 3. The scheduling constraints set (a) by non wrap-around, and (b) by wrap-around links (used in topologies with rings) for achieving zero-latency flit traversal across routers.

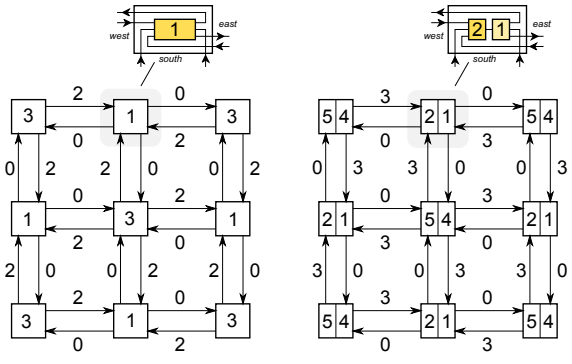


Fig. 4. Zero-latency schedules on a 3×3 2D mesh using (a) 4 domains with single-cycle routers, and (b) 6 domains in 2-stage pipelined routers. All incoming links of a router drive the domain served by the first pipeline stage.

we want router A to receive in-sync the flits coming from B, then, in the next clock cycle, it should be able to serve their domain in its first pipeline stage. In the next cycle, router A would be in domain $D_0 + 1 \bmod D$, which should be equal to $D_0 - 2P - 1 \bmod D$, where D is the number of domains.

This constraint is directly satisfied when the total number of domains is equal to $2(P+1)$. Figure 4 depicts the assignment of 4 application domains to network routers and, implicitly, to the network links, for a 3×3 mesh. Figure 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, Figure 4(b) shows the schedule applied to 2-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 a maximum of 6 application domains. All incoming links feed the first pipeline stage of the router, and all output links are driven by the last stage.

The proposed schedule satisfies the constraint that all the incoming links of each router serve the same application domain. This is a simplifying feature for the router design, and a requirement for offering isolation across domains, since (on every allocation step or pipeline stage inside the router) the same domain is active on all inputs concurrently.

The packets at the inputs of each router wait for the turn of their application domain to come. Once this happens, they follow all the allocation steps needed, contending only with the packets/flits of the same application domain in each stage. Subsequently, they move un-interrupted from router to router until their final destination, without experiencing any contention from other application domains. If a flit manages to win all other flits of the same application domain, it will reach its destination in exactly $H \times (P+1)$ cycles, independent of the path that it follows. Turning, or staying in the same dimension, does not differentiate the latency of each flit.

B. Perfect schedules on ring structures

The application of the proposed scheduling mechanism to topologies that contain wrap-around links, such as rings (hierarchical, or not) and tori, sets an additional constraint. As depicted in Figure 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 $D_0 - k(P+1) + 1 \bmod D$ domain, 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 $(D_0 - k(P+1) + 1) \bmod D = (D_0 + 1) \bmod D$. This is satisfied when $D = k(P+1)$.

Therefore, although the wrap-around connections allow the network to host more than $2(P+1)$ application domains under a perfect schedule, the maximum number cannot be increased and gets limited to $2(P+1)$, due to the zero-latency overhead required in the backward non-wrap-around connections. However, accounting also for the scheduling constraint set by the wrap-around links, then k (i.e., the number of nodes in a ring connection) is required to be an even number, else un-necessary latency is incurred in certain connections. An example of applying the proposed schedule of 4 domains in a 6-node ring is illustrated in Figure 5.

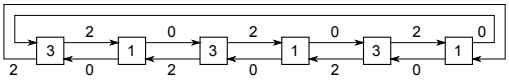


Fig. 5. Zero-latency schedule of 4 domains in a 6-node ring network.

C. Extended domain support with physical sub-networks

Such a perfect and un-interrupted schedule is offered to the flits of different application domains, irrespective of the topology used, when the total number of application domains operating in an isolated manner is $2(P+1)$ (roughly the maximum is 10 for a 4-stage pipelined router). Despite the fact that 10 domains (i.e., 10 VCs, or 10 groups of VCs in each router input port) would be more than adequate for most applications [13], a general methodology to support even larger numbers of domains is also presented here for completeness.

Supporting more application domains with zero latency overhead can be directly performed by increasing P, i.e., the pipeline stages seen between routers. Even if the NoC achieves its desired clock frequency with a certain number of pipeline stages, *dummy* pipeline stages could be added to allow PhaseNoC to host even more application domains.

Alternatively, PhaseNoC can rely on multiple physical networks for extending the supported application domains. Each physical sub-network follows a zero-overhead schedule, supporting as many as $2(P+1)$ domains.Supporting B application domains requires $B/2(P+1)$ networks, with each one serving an independent group of domains. For example, supporting a total of 16 domains is possible in two ways: either with 4 physical sub-networks, each one built with single-cycle routers and supporting a perfect 4-domain schedule, or 2 sub-networks built with 3-stage pipelined routers and supporting a perfect schedule of 8 domains. The first sub-network would serve domains 1-8, and the other domains 9-16. With this approach, every flit experiences serialization latency when entering/exiting each sub-network, but it then traverses the network un-interrupted, independent of the load in the other domains, and independent of the path that it follows.

Multiple sub-networks become cost-effective when they operate on a smaller flit width than the original network. Under equal bisection bandwidth, the links of each sub-network should be $2(P+1)$ times smaller. In this case, even if each sub-network follows a perfect schedule up to $2(P+1)$ domains, the latency per packet is increased for a larger number of application domains, due to the additional serialization imposed by each sub-network using narrower links. Nevertheless, we will demonstrate in Section IV that the approach of multiple PhaseNoC sub-networks under equal bisection bandwidth is a very cost-effective option when the number of application domains is large.

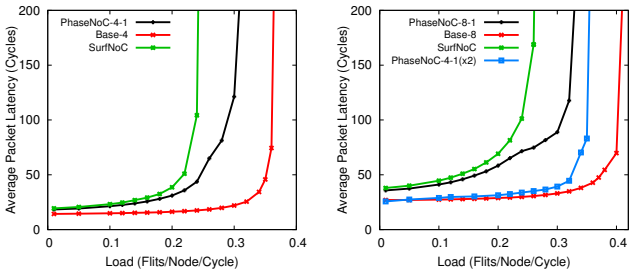
IV. Experimental results

In this section, we show the flow isolation properties of PhaseNoC and compare it with conventional VC-based NoCs and the state-of-the-art SurfNoC architecture [8], in terms of network performance and hardware complexity.

A. Network performance & traffic isolation

Network-performance comparisons were performed using a cycle-accurate SystemC network simulator that models all micro-architectural components of the router. An 8×8 2D mesh network is assumed, with XY routing. The evaluation involves uniform-random traffic. The injected traffic consists of two types of packets to mimic realistic system scenarios: 1-flit short packets (just like request packets in a CMP), and longer 5-flit packets (just like response packets carrying a cache line). Two recent studies have performed detailed profiling analysis of the network packets generated by real multi-threaded applications in CMPs [13, 14]. Their conclusion is that approximately 80% of the packets are single-flit. In accordance with these studies, we assume in our latency-throughput experiments that 80% of the packets are single-flit, with the rest being 5-flit.

Figure 6(a) depicts the load-latency curves of PhaseNoC (PhaseNoC-4-1), a baseline (Base-4), and SurfNoC [8] single-cycle routers using 4 VCs (one VC per domain), under uniform-random traffic. Baseline routers assume that each VC is a virtual network, thus in-flight VC changes are prohibited. Although PhaseNoC routers operate under a static TDM schedule at the VC level, their zero-load latency is still very close to that of the baseline designs, due to the efficient network-level schedule that 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 8% lower overall throughput, due to the non-interfering operation across domains, which reduces – in some sense – the choices for flit interleaving. The behavior of SurfNoC [8] shows slighltly higher latency at low loads and lower saturation throughput for a 4-domain operation. This is a natural effect of the wave propagation in SurfNoC, which adds additional latency when moving from a south-east direction to a north-west one. Further, in our experiments, SurfNoC was modeled without input speedup [11], in order to have a fair comparison relative to PhaseNoC and the baseline designs.



(a) 4 domains (b) 8 domains

Fig. 6. Average network latency as a function of aggregate offered load in all domains, assuming an 8×8 mesh network.

Similar conclusions can be drawn from Figure 6(b), which compares an 8-domain (8 VCs) 3-stage pipelined PhaseNoC router (PhaseNoC-8-1) with an 8-VC baseline router (Base-8), and an 8-domain pipelined SurfNoC [8] with 8 VCs per input.

Using PhaseNoC, 8 domains can alternatively be supported using two parallel sub-networks (having equal total bisection bandwidth), with each one supporting 4 domains. This setup is captured by the "PhaseNoC-4-1(x2)" curve in Figure 6(b). In this case, the latency inside each sub-network decreases, due to the single-cycle operation of the routers, but also increases at the same time, due to the increased serialization of the packets. The final latency and throughput observed at low loads is close to the baseline, and better than the single-network case, thus rendering this hybrid physical-virtual domain isolation a promising choice. Although this is not an apple-to-apple comparison, since single-cycle routers cannot reach the frequency of their pipelined counterparts, the low-complexity of PhaseNoC's allocation, in conjunction with other low-delay allocation strategies (such as combined allocation [11]) could provide efficient single-cycle implementations.

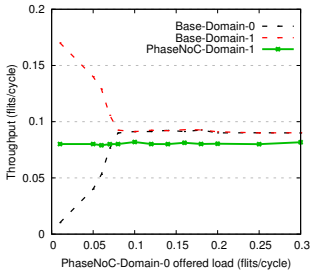


Fig. 7. Non-interfering isolation of traffic. Traffic in VC0 of PhaseNoC is constant, irrespective of the load behavior in the other domains.

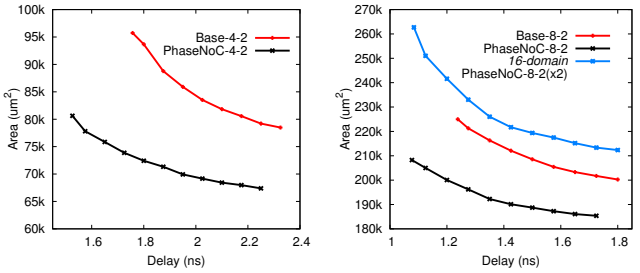
For testing the flow isolation properties of PhaseNoC, we created a specific traffic scenario involving a 4-domain network. Each domain has 1 VC, in both PhaseNoC and

a baseline network. In both cases, VC2 and VC3 serve a constant load of 0.08 flits/node/cycle. The load on VC1 is then progressively increased, and we observe the impact on VC0's throughput; the traffic on all VCs is uniform-random. As shown in Figure 7, the load in VC0 of PhaseNoC is completely immune to the changes in VC1's load (as also supported by SurfNoC). However, in the baseline case, as the load in VC1 increases, the sustained loads in VC0 and VC1 converge to an equal value of 0.09 flits/node/cycle.

B. Hardware evaluation

The routers under comparison (using look-ahead RC) were implemented in VHDL, mapped to a commercial low-power 45 nm standard-cell library under worst-case conditions (0.8 V, 125 °C), and placed-and-routed using the Cadence digital implementation flow. The generic router models have been configured to 5 input-output ports, as needed by a 2D mesh network, while the flit width was set to 64 bits. The area/delay curves were obtained for all designs, after constraining appropriately the logic-synthesis and back-end tools, and assuming that each output is loaded with a wire of 2 mm.

In the first set of experiments, we 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. Figure 8(a) depicts the area-delay curves for a single-stage PhaseNoC router supporting 4 domains with 2 VCs per domain (PhaseNoC-4-2), versus a baseline router with 8 VCs (Base-4-2). Both routers have equal buffering, necessary to cover the credit RTT independently per VC. In every design point, PhaseNoC routers require less area (up to 22%) and delay (up to 13%). The reported savings come mostly from the complexity reduction in the allocation units. PhaseNoC's allocation is always limited to the input VCs of one domain, which allows both sharing of the arbiters, as well as the reduction of their logic depth. The savings relative to SurfNoC are expected to be even larger, since *SurfNoC routers are already more costly than baseline designs* that do not isolate traffic flows [8].



(a) 4 domains (b) 8/16 domains

Fig. 8. Hardware implementation results for various router designs with (a) 4 domains, and (b) 8 domains with 2 VCs/domain/input port. Additionally, a 16-domain solution (32 VCs in total) is included, which is built from two parallel 8-domain sub-networks with half link-width.

The second set of experiments refers to pipelined router implementations, which serve more non-interfering domains in the network. The derived results are shown in Figure 8(b). PhaseNoC-8-2 corresponds to a 3-stage pipelined router that operates in phases, serving a total of 8 different domains, with each domain comprising a group of 2 VCs. The router sees in its pipeline a sliding window of 3 domains in each clock cycle. Base-8-2 is a baseline 3-stage router implementation supporting the same total number of VCs per input (i.e., 16 VCs). Again, the proposed PhaseNoC router is both faster and more area efficient than the baseline design, with up to 13% delay savings and up to 11% area savings.

Finally, we evaluated the efficiency of applying the PhaseNoC concept to parallel sub-networks. PhaseNoC-8-2(x2), depicted in Figure 8(b), corresponds to the sum of the area of two 3-stage pipelined PhaseNoC-8-2 routers, each supporting 8 domains with 2 VCs per domain, but with a flit-width of 32 bits (2× shorter than the other two designs). With only a minor 5% area overhead, compared to the baseline at its minimum delay point, PhaseNoC is able to support twice as many as domains (i.e., 16 domains with 32 VCs in total), providing a viable solution to increased domain requirements.

V. Conclusions and Future Work

In addition to providing high performance and scalability, modern NoCs are required to support additional functionality, such as QoS provisioning and performance isolation among domains. This paper introduces PhaseNoC, a cost-efficient, truly non-interfering, VC-based interconnect architecture. The new design relies on TDM scheduling at the VC-level, which optimally coordinates the multi-phase propagation across the network. PhaseNoC's router microarchitecture facilitates strict domain isolation and zero-latency flow propagation of flits. Extensive evaluation verifies that PhaseNoC constitutes a low-cost, non-interfering design, with no sacrifices in performance.

Future work will explore dynamic per-router application domain selection, which will allocate more time slots to busy domains than to ones with lighter traffic. Checking and comparing the state of all domains before deciding which one to serve inevitably introduces some (implicit) interference across domains. As a result, the ultimate scheduling decision will be tackled in a programmable manner. When complete network traffic isolation is a prerequisite, a strict TDM schedule should be followed both at the router- and the network-level. Instead, when some interference can be tolerated, dynamic scheduling could be employed to reduce latency.

References

[1] J. Handy, "NoC interconnect improves SoC economics," *Objective analysis - Semiconductor market research*, 2011.
[2] W. J. Dally, "Virtual-Channel Flow Control," in *ISCA*, 1990, pp. 60–68.
[3] K. Goossens *et al.*, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test*, vol. 22.
[4] A. Hansson *et al.*, "aelite: A flit-synchronous network on chip with composable and predictable services," in *DATE*, 2009, pp. 250–255.
[5] R. Stefan *et al.*, "A TDM noc supporting qos, multicast, and fast connection set-up," in *DATE*, 2012, pp. 1283–1288.
[6] M. Schoeberl *et al.*, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *NoCS*, 2012, pp. 152–160.
[7] E. Kasapaki and J. Sparsø, "Argo: A time-elastic time-division-multiplexed NOC using asynchronous routers," in *ASYNC*, 2014.
[8] H. Wassel *et al.*, "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip," in *ISCA*, 2013, pp. 583–594.
[9] B. Grot *et al.*, "Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees," in *ISCA*, 2011, pp. 401–412.
[10] J. W. Lee *et al.*, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *ISCA*, 2008, pp. 89–100.
[11] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, *Microarchitecture of Network-on-Chip Routers: A designer's perspective*. Spinger, 2014.
[12] D. U. Becker *et al.*, "Adaptive backpressure: Efficient buffer management for on-chip networks," in *ICCD*, 2012, pp. 419–426.
[13] S. Ma *et al.*, "Novel flow control for fully adaptive routing in cache-coherent nocs," *IEEE TPDS*, pp. 2397–2407, Sep. 2014.
[14] J. Lee *et al.*, "Centaur: a hybrid network-on-chip architecture utilizing micro-network fusion," *Springer DAEM*, pp. 1–19, 2014.