INTEGRATION, the VLSI journal 77 (2021) 104–112

Contents lists available at ScienceDirect

Integration

journal homepage: www.elsevier.com/locate/vlsi

Multicast-enabled network-on-chip routers leveraging partitioned allocation and switching

Dimitris Konstantinou [a], Chrysostomos Nicopoulos [b], Junghee Lee [c], Giorgos Dimitrakopoulos [a,*]

[a] Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[b] Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus
[c] School of Cyber Security, Korea University, Seoul, South Korea

ARTICLE INFO

Keywords:
Network-on-Chip
Multicast
Router
Micro-architecture

ABSTRACT

Multicast on-chip communication is encountered in various cache-coherence protocols targeting multi-core processors, and its pervasiveness is increasing due to the proliferation of machine learning accelerators. In-network handling of multicast traffic imposes additional switching-level restrictions to guarantee deadlock freedom, while it stresses the allocation efficiency of Network-on-Chip (NoC) routers. In this work, we propose a novel partitioned NoC router microarchitecture, called *SmartFork*, which employs a versatile and cost-efficient multicast packet replication scheme that allows the design of high-throughput and low-cost NoCs. The design is adapted to the average branch splitting observed in real-world multicast routing algorithms. Compared to state-of-the-art NoC multicast approaches, SmartFork is demonstrated to yield high performance in terms of latency and throughput, while still offering a cost-effective implementation.

1. Introduction

Networks-on-Chip (NoC) have been universally established as the enabling communication fabrics that can sustain the many-core era. Modern NoCs need to support both unicast (point-to-point) and multicast (one-to-many) traffic. Multicast (and often broadcast) communication is widespread in some of the most popular cache-coherence protocols targeting multi-core processors, while the multicast intensity has been shown to increase with the number of on-chip cores, thereby underscoring its performance impact on system scalability. Furthermore, multicast traffic is also widespread in the increasingly prevalent hardware accelerators targeting artificial intelligence [1].

In-network multicast support can take various forms. A naive and low-performing approach is to inject multiple unicast clones of the packet, with each one sent to a distinct recipient (i.e., *unicast-based*). To increase performance – while keeping a minimal multicast-packet footprint – one may employ *path-based* multicast routing [2–4]. Only one multicast packet is injected, which sequentially visits all recipients. The packet is delivered to a single recipient at a time, allowing for at most two multicast branches, with one of them always sinking into the local ejection port.

To further improve performance, *tree-based* multicasting relaxes the replication degree, allowing for branching to an arbitrary number of output ports within each router [5–9]. This extra flexibility enables each recipient to be served independently, and, thus, more quickly, as opposed to creating sequential dependencies among recipients.

Nevertheless, the increased branching flexibility in tree-based multicast algorithms is precisely the reason why deadlocks may, in fact, arise at the multicast-replication level, i.e., as a result of dependencies among the various multicast branches. Note that such switching-level deadlocks arise even if the routing algorithm is deadlock-free. To tackle this issue, Virtual Cut-Through (VCT) switching [10–12] is employed. Alternatively, low-performance circuit-switching approaches [13,14], or costly deadlock recovery schemes [15], have also been presented.

Multicast packets in NoCs are generally short, as they typically carry only control information [16], or single-word data; e.g., a cacheline invalidation message. Hence, multicast messages in NoCs could most likely fit within a single-flit packet. Single-flit packets make the routing of each multicast branch independent, and their switching in each router is, *by construction*, deadlock-free. This attribute has been exploited in Ref. [6,9] to build multicast NoCs. In a similar vein, the work in Ref. [17] transforms all multicast packets into independent single-flit multicast packets.

* Corresponding author.
E-mail address: dimitrak@ee.duth.gr (G. Dimitrakopoulos).

https://doi.org/10.1016/j.vlsi.2020.10.008
Received 21 March 2020; Received in revised form 22 July 2020; Accepted 25 October 2020
Available online 7 November 2020

Single-flit multicast allocation and switching inside each router can be performed in two ways. One approach is to treat multicast branch replication as a set of serially-executed unicast transmissions; i.e., sending a flit to multiple output ports in the same cycle is prohibited [11]. This branch serialization yields a low-cost multicast solution, but limits the flit replication rate to one output port per cycle. Alternatively, each multicast packet can be replicated in parallel to all required output branches in each router [9,18]. This parallel replication is readily supported by the crossbar of each NoC router. However, the increase of requests from multiple input virtual channels to multiple output virtual channels stresses the separable allocators used in state-of-the-art NoC routers [19], thereby increasing Head-of-Line (HoL) blocking.

To address the inefficiency of separable allocation when dealing with mixed multicast and unicast traffic, the use of input buffers with as many independent read ports as the number of output ports in the router (aka *buffer speedup*) has been proposed originally [10] and applied in a NoC environment in Refs. [12,20]. This approach increases throughput by providing a separate path for each input-to-output connection, but with prohibitive hardware cost – especially with respect to local wiring congestion.

Nevertheless, the key question is whether real-world multicast routing algorithms exhibit such high levels of parallel branch splitting in each router that would justify parallel branch replication combined with full-fledged buffer speedup.

Parallel branch replication per router – with or without buffer speedup – assumes that each input port can be connected in parallel to all output ports. However, measurements on traffic generated by real applications show that the amount of observed multicast branch splitting is much lower than the number of output ports. For instance, Fig. 1 depicts the average branch splitting across all PARSEC applications [21] observed in each router of an 8×8 2D Chip Multi-Processor (CMP) in a full-system simulation setup (the details pertaining to the evaluation platform are described in Section 4.2). The routing algorithm employed was XY.

In all cases, the average splitting is below 5 (the number of output ports of a 2D mesh router), while, in the periphery of the NoC, it barely surpasses 1, as needed by unicast packets. These low levels of intra-router multicast branch splitting are, in fact, orthogonal to the NoC traffic characteristics and the application workloads running on the multicore system. *The degree of multicast branch splitting is inherently an attribute of the routing algorithm itself*. Similar conclusions are reached when using other state-of-the-art multicast routing algorithms, such as Whirl [9], which constitutes a superset of the trees produced by several other multicast routing algorithms [9].

Motivated by the observation that the degree of branch replication per router, as dictated by the multicast routing algorithm, is much lower than the number of input/output ports in a router, we hereby propose *SmartFork*, a scalable multicast-enabled NoC micro-architecture that is characterized by the following novel features:

(1) The output ports are partitioned into groups; intra-group ports are serviced serially, while inter-group ports are serviced in parallel. Therefore, SmartFork can be used to target any desired point on the multicast performance spectrum.
(2) Each output partition is driven by a separate input-buffer read port, thus mitigating allocation inefficiencies without introducing prohibitive hardware cost.
(3) Virtual Channels (VC) are inherently supported without any restrictions on which VCs can support multicasting; any VC within the router can support multicast transfers.

SmartFork is evaluated through multifaceted and extensive cycle-accurate simulations, as presented in Section 4. The evaluation framework employs both synthetic traffic patterns and execution-driven, full-system simulations with real application workloads. Additionally, detailed hardware analysis of synthesized and placed-and-routed designs using commercial 45 nm standard-cell libraries corroborates the design's implementation efficiency.

1.10	1.58	1.42	1.05	1.05	1.04	1.03	1.01
1.29	2.63	2.91	2.96	2.98	2.99	2.02	1.03
1.22	2.48	2.92	2.93	2.74	2.97	2.02	1.04
1.24	2.38	2.98	2.96	2.97	2.98	2.04	1.05
1.22	2.27	3.09	2.91	2.93	2.89	2.02	1.02
1.22	2.31	3.17	2.95	2.96	2.94	2.03	1.03
1.22	1.28	3.21	2.98	2.99	2.98	2.03	1.03
1.09	1.53	1.39	1.05	1.05	1.04	1.02	1.01

Fig. 1. The average number of branch splits per router observed across all PARSEC benchmark applications [21] when executed in a full-system simulation of a 64-node x86 CMP organized as an 8×8 2D mesh of processor tiles.

2. SmartFork: partitioned multicasting

The NoC routers facilitate the transfer of packets between communicating (source/destination) nodes. Packets are typically split into smaller flow-control units, aka *flits*. The flits traverse the network in a hop-by-hop fashion through all the routers encountered on the path from the source to the destination. The micro-architecture of a typical VC-based NoC router [19] is depicted in Fig. 2.

Each router has N input and N output ports. In a 2D mesh NoC topology, $N = 5$; one input/output port for each cardinal direction, and a local injection/ejection port. Each input port has V virtual channels, with each channel being served by a different FIFO input buffer. All incoming packets are written into these input buffers, and they pass through a series of operational stages before they can exit the router. The Routing Computation (RC) logic determines the output port of each packet. The Virtual-channel Allocation (VA) stage maps input VCs to output VCs. The Switch Allocation (SA) stage – that is typically split in two stages, SA1 and SA2 – arbitrates amongst all input VCs requesting access to the crossbar, and declares one winning flit for each input and output port. The SA winners are then able to traverse the crossbar (Switch Traversal, ST), and are placed on their respective output links.

The basic organization shown in Fig. 2 can, in fact, be used to also support *multicast* transmissions, as explained in Ref. [9]. This is achieved by allowing the flits of each input VC to simultaneously send requests to *multiple* output ports. For instance, if a multicast packet must branch out to three output ports of the router (as decided by the employed multicast routing algorithm), then the packet is allowed to send concurrent

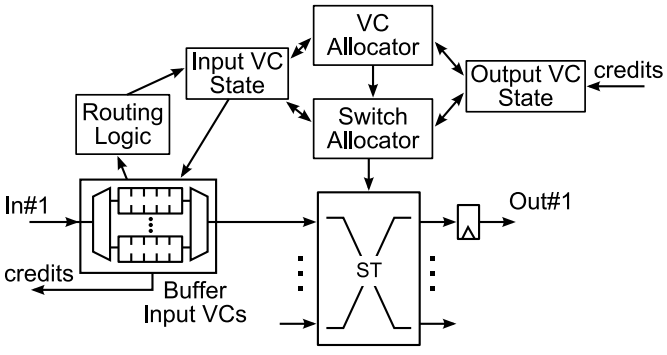


Fig. 2. The micro-architecture of a typical virtual-channel-based NoC router.

requests to all three output ports. The same multicast architecture has also been used in the Scorpio cache-coherent multicore research prototype [18]. While said approach is effective in handling multicast communication and is cost-efficient, it suffers from an often-debilitating affliction: increased susceptibility to HoL blocking. Specifically, if any of the multiple requested output ports is/are unavailable, the flit at the head of the input buffer must remain there until *all required output ports* are, eventually, served. This requirement inevitably blocks the flits stored behind the head flit in the same input VC, even though they may be heading towards available output ports.

Recently, the work in Refs. [12] revived an older architecture [10] that employs buffer speedup. In this case, each multicast packet is served by independent input read ports used for accessing the different outputs. Thus, in contrast to Ref. [9], a packet is blocked only when the requested output port is not available, and not because a flit at the head of the same input buffer cannot be replicated to other output ports. The architectures in Ref. [9,12] constitute the current state-of-the-art in multicast-enabled router architectures, and they will be used for experimental comparisons in Section 4.

2.1. Router organization and operation

SmartFork is founded on *two basic properties* that enable high multicasting efficiency with minimal changes to the baseline VC-based NoC router microarchitecture. The first property defines the number of multicast branches that can be served in parallel in any given cycle, denoted as P. The value of P aims to reflect the average branching degree per router observed in established multicast routing algorithms. To achieve this in a cost-efficient manner, the N output ports in a SmartFork router are partitioned into P groups. Since P is an architectural parameter, the whole design space is covered: the serial approach corresponds to $P = 1$, while the fully parallel one corresponds to $P = N$. The proposed multicast-enabled design generalizes the degree of parallel branch replication to any value in the range $1 \leq P \leq N$, having a double-faceted goal: matching the needed replication parallelism within the router, thereby minimizing serialization latency, while the P-way independent/partitioned buffer access allows access deeper in the buffer – past the FIFO's head position – thus mitigating the effects of HoL blocking.

To reduce allocation inefficiencies and HoL blocking when serving both multicast and unicast traffic, at a reasonable cost, the router connects each partition of output ports with a distinct and independent read port per input VC, as shown in Fig. 3. Each one of the P read ports per input VC serves a different and statically allocated partition of the router's output ports. For example, in the case of a 2D mesh ($N = 5$), and assuming $P = 2$, the first read port of each input buffer can serve 3 output ports (East, West, and the Ejection Port), while the second read port of each input buffer can serve the remaining 2 output ports (North and South). The P read ports per input buffer allow for independent (i.e., non-synchronized) and parallel branch replication across the output-port partitions. Thus, HoL blocking is allowed for traffic heading to the same output partitions and it is completely removed for outputs in different output partitions. This approach is enough, as shown by real application traffic demands, while using input buffers with multiple independent read ports equal to the number of output ports in the router (aka "buffer speedup"), as done [12] for reducing HoL blocking, incurs an unnecessary cost. Within the SmartFork concept, the approach followed in Ref. [9] and the Buffer Speedup used in Refs. [12], can be thought of as the $P = 1$ and $P = N$ cases, respectively.

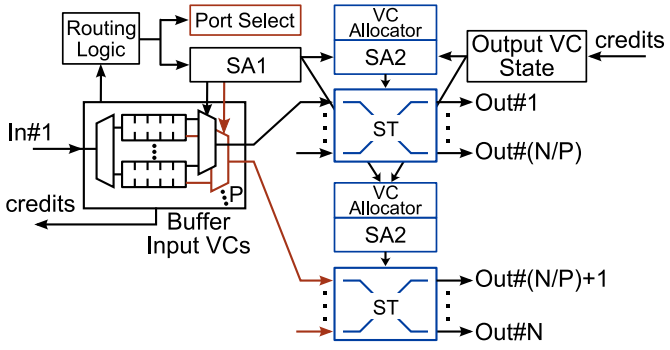


Fig. 3. The organization of a SmartFork router. Assuming the NoC router has N output ports, SmartFork employs P read ports (with $1 \leq P \leq N$) to serve a maximum of P multicast branches in parallel, per cycle.

This specific partitioning choice is also depicted in the walk-through example of Fig. 4. Said example highlights the independence among the P read ports, which has a clear benefit in reducing HoL blocking. Multicast flit A of a certain input VC needs to be replicated to the *North*, *South*, and *East* output ports of a router in a 2D mesh network. This replication occurs through the two partitions of the SmartFork router. In the first cycle, flit A needs to request one of the two (*North*, or *South*) output ports that belong to partition 0, and, concurrently, the *East* output port that belongs to partition 1. Assume that flit A gets successfully serviced for the *South* output port, but it is blocked on the *East* output port.

In the next cycle, the request of flit A for the blocked *East* output port persists, while the flit is also requesting the *North* output port (note that the flit has not yet been replicated to this particular output port). Having the *North* output port serviced, all the output ports of flit A that may be serviced through the second read port (i.e., the one dedicated to the partition of the *North* and *South* output ports) have been fully served and, thus, flit A will not use that read port again.

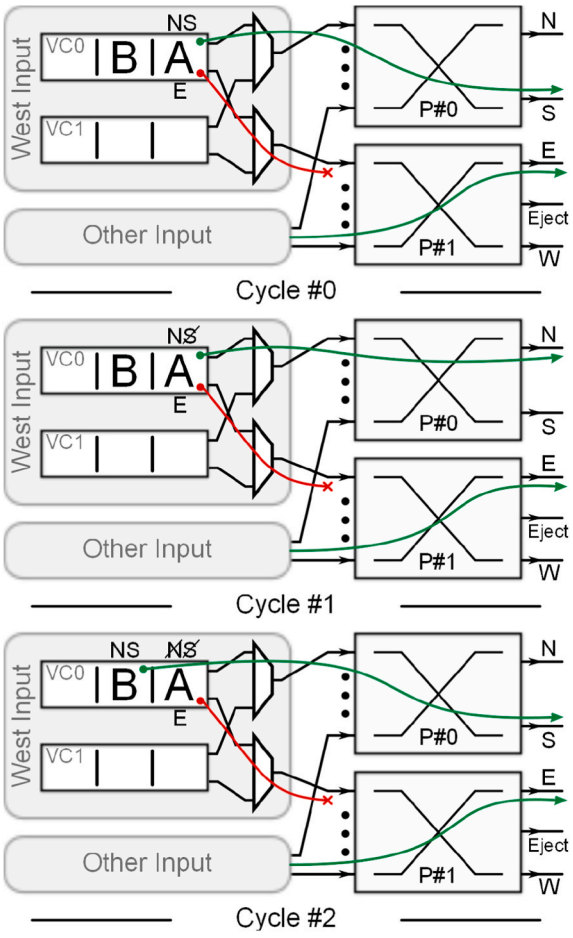


Fig. 4. Walk-through example of the operation of the SmartFork architecture. In this example, multicast flits A and B are transmitted towards their respective required output ports. Let us assume that the *East* port remains unavailable, thereby causing HoL blocking.

SmartFork's architecture allows the next flit, *B*, to request transmission through that partition's – otherwise idle – read port. Therefore, the request for the *East* output port still persists for flit *A*, since it has not yet been serviced, while, concurrently and independently, flit *B* is allowed to request the *South* output port and move to it through the second read port of the same input VC. Each read port is able to continue servicing all flits heading towards available output ports, regardless of the progress of flit *A* that is headed to a currently unavailable output port, thus effectively reducing HoL blocking per input VC.

2.2. Switch and VC allocation

Input-to-output allocation and switching in a SmartFork router involves several steps. Initially, each input VC prepares at most *P* flit requests. According to the outcome of the multicast routing computation logic, each request may refer to multiple output ports that may possibly belong to multiple output partitions. To simplify the design, the requests are grouped per partition. Then, for each partition separately, each one of those flit requests must win locally in the SA1 stage, which arbitrates among the requests from all the VCs of the same input port. In other words, the flits in each input port only contend with the flits of all other VCs belonging to the same input port and wanting to move to the same output partition. Therefore, the delay complexity of arbitration in the SA1 stage remains identical to the one encountered in VC-based routers [22,23], while, instead, in SmartFork *P* independent SA1 arbiters operate in parallel per input port.

To keep complexity under control, each read port serves its respective output ports in a serial manner within its own output-port partition. If a multicast packet must be split into two multicast branches in a SmartFork router, and both branches must exit from output ports that belong to the same output partition, then those branches will be served serially. For example, using the aforementioned partitioning of output ports in a 2D mesh (described in Section 2.1), if an incoming multicast packet must branch out to the North and South output ports of the router, those branches will be served serially (one after the other), since the North and South output ports are both served by the same read port. The packet will first be forwarded to one output port, and then to the other output port.

The decision of which output port is served first from each input read port is determined by the port selection logic, which selects one active output port from those computed by the multicast routing unit. This selection (of a single output port) needs an additional arbitration step that is performed in parallel to SA1 for all contending flits, as shown in Fig. 3.

On the other hand, if a packet must branch out to the North and West output ports, then both branches can be served in parallel, since the North and West output ports belong to different output-port partitions and are, thus, served by different read ports.

The *P* winners of the SA1 stage will move in parallel to their corresponding output partition having selected one valid request inside each output partition. Also, since SmartFork is by design non-speculative and each allocation step is always productive, the valid requests must refer to an available resource. This is achieved by masking the port request with the availability/readiness of the output port's VCs.

Inside each output partition, SA2 arbiters [24] are employed (one for each output port) which only see unicast requests. Therefore, one *N*: 1 arbiter per output is enough for the SA2 arbitration step. Essentially, multicasting is achieved by replicating the same flit to different partitions. The separate arbitration and multiplexing at each output port can be efficiently implemented using the merged arbiter-multiplexers of [25].

SmartFork follows a combined allocation [19,26] approach, thus not requiring a separate VC allocator. In combined allocation, the input VC that actually won access to its selected output is assigned to an available output VC selected in parallel to SA2. Alternatively, the assigned output VC could be selected from a pool of available VCs given to winning packets in FIFO order [27].

3. Input-to-output partition assignment

SmartFork assumes that input ports are statically assigned to the *P* output partitions. Once a partition is selected at design-time, no re-partitioning is allowed thereafter. However, the input-to-output-partition connectivity need *not* be the same for all routers of the network and may optimally be adapted to (a) the (possibly) known traffic characteristics and (b) how the routing algorithm distributes traffic across the NoC.

This principle also applies inside each router, where each input port can assume a different output port partitioning. For example, in the case of $P = 2$ for a 2D mesh router, Read Port #0 of the North input port can serve the output partition {West, East}, and Read Port #1 can serve output partition {Local Eject Port, South}. The East input of the same router is not obligated to use the same read-port-to-output assignment and can connect its Read Ports #0 and #1 to output partitions {North, West} and {South, Local Eject Port}, respectively. Note that, in both cases, the output connections used in this example satisfy the XY routing algorithm.

Our goal is to solve the assignment of the read ports of each input to the output ports optimally, using a novel Integer Linear Program (ILP) formulation that tries to *load-balance* the traffic passing through each one of the *P* available read ports per input. Without loss of generality, we assume that, for each input under consideration, we have available traffic demand weights w_j that declare how much of the incoming traffic needs to leave from output port j of the router. If normalized, the sum of the weights of all outputs for one input should not exceed one.

These weights can be derived from simulation of real or synthetic traffic, or they can be probabilistic approximations of the traffic expected to pass from each input-to-output connection, based on the characteristics of the routing algorithm. For example, the North output of a router in a 2D mesh is expected to receive more traffic than the East output, merely due to the turning restrictions of the XY routing algorithm. Similarly, the East-to-West connection of a router in the center of the NoC is more likely to have a higher weight than in a router on the periphery of the NoC.

Balancing the utilization of the *P* read ports means that each read port experiences similar traffic demand and no read port is over-utilized compared to the others. This balancing can be achieved by properly assigning read ports to outputs. To control which read port is connected to which output, we define binary variables $x_{ij} \in 0, 1$: if $x_{ij} = 1$, it means that the read port i of an input, with $0 \leq i \leq P-1$, is connected to output j of the router, with $0 \leq j \leq N-1$. Since, in SmartFork, each output cannot connect to more than one read ports of the same input, we need to guarantee that for each output port j, $\sum_{i=0}^{P-1} x_{ij} = 1$.

Based on the assignment of variables x_{ij}, the traffic load passing through the read port i is equal to

$$\varphi(i) = \sum_{j=0}^{N-1} x_{ij}\, w_j \tag{1}$$

Each read port would be perfectly load-balanced if all receive approximately the same load, i.e., $\varphi(i) \approx \varphi(k)$, $\forall k \neq i$. If this happens, each read port would see a load equal to the average load of all outputs, i.e., $w_{\text{avg}} = \sum_{j=0}^{N-1} w_j / N$.

The defined optimization problem aims at minimizing the absolute distance of the load of each read port $\varphi(i)$ from the average traffic load w_{avg} of the input port under investigation. Therefore, formally, we define the load-balancing assignment problem as follows:

$$\begin{aligned} \text{minimize:} \quad & \sum_{i=0}^{P-1} |\varphi(i) - w_{avg}| \\ \text{subject to:} \quad & \sum_{i=0}^{P-1} x_{ij} = 1 \\ & x_{ij} \in \{0,1\}, \forall i,j \end{aligned} \tag{2}$$

The absolute value function can be easily handled in the ILP formulation by introducing additional variables and constraints [28]. The ILP is solved using the Gurobi solver [29] in less than a second, since the assumed numbers of output ports N and read ports P are limited in real-world scenarios.

To see how the assignment problem works, let us describe a simple numerical example. Assume that an input port needs to connect to 5 output ports, and that the traffic percentage to each output port is distributed as follows:

$$\{w_0,w_1,w_2,w_3,w_4\} = \{0.375,0.457,0.061,0.083,0.024\}$$

In other words, this particular input port needs to send most of its traffic to outputs 1 and 0, respectively. If those two outputs are grouped in the same partition, then the read port assigned to it would be over-utilized. To avoid such scenario, the ILP gives the following assignment for the case of $P = 2$ read ports:

Read Port #0 ↔ [1, 2].

Read Port #1 ↔ [0, 3, 4]

In this way, the total load expected on Read Port #0 would be 0.518, while the total load on Read Port #1 would be 0.464.

4. Experimental evaluation

In this section, we evaluate the performance and hardware cost of SmartFork, as compared to two current state-of-the-art multicast-enabled NoC architectures: Design presented in Ref. [9] and a synchronous equivalent version of [12]. Recall that the design in Ref. [9] facilitates the simultaneous transmission of flits from one input VC to multiple output ports. This is achieved by allowing input VCs to concurrently send switch allocation requests to multiple output ports, while using the same data multiplexing network. On the contrary [12], multiplies data multiplexing paths offer per input VC as many independent read ports as the number of output ports in the router.

The performance evaluation approach is double-faceted, utilizing (1) *synthetic* traffic patterns, and (2) *real application workloads* running in an execution-driven, *full-system* simulation environment. Synthetic traffic patterns are initially used – in Section 4.1 – to stress the evaluated designs and isolate their inherent network attributes. Subsequently, in Section 4.2, real application workloads are employed to yield more authentic insights into the real-world performance of the compared architectures. Finally, a comprehensive hardware implementation analysis is presented in Section 4.3.

4.1. Network performance using synthetic traffic

Network performance comparisons were performed using a cycle-accurate SystemVerilog network simulator that accurately models all microarchitectural components of the NoC routers. Latency and throughput measurements were derived from simulations, assuming two different network topologies: (a) an 8×8 2D mesh network using 5-port VC-based routers, and (b) a 4×4 2D high-radix mesh network using 8-port VC-based routers. Each input port employs a 3-flit deep FIFO buffer. Without loss of generality, all NoC routers have single-cycle operation (even though all techniques could also be fitted to router pipelines of arbitrary length). Dimension-ordered XY routing is used for both unicast and multicast traffic. For the experiments in this subsection, two variants of each architecture under investigation are compared: one with 2 VCs per input port of the NoC router, and one with 4 VCs.

In terms of synthetic traffic patterns, we evaluate uniform-random traffic, whereby every node sends its packets to all other nodes of the network with equal probability. Other traffic patterns – such as transpose, bit-complement, and non-uniform localized – have also been tested and exhibited equivalent behavior without noteworthy changes to the observed trends. 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 3-flit packets (just like *response* packets carrying a cache line). We assume a bimodal distribution of packets with 50% of the packets being short, 1-flit packets, and the rest being long, 3-flit packets, in accordance to recent studies [16]. Both unicast and multicast packets are injected into the network. In general, the actual amount of multicast traffic (as a percentage of the total network traffic) is an application- and/or platform-specific attribute. Prior research has reported real-world multicast percentages ranging from around 5% to almost 30% of the total injected traffic [5,30]. Guided by these numbers, we investigate two different scenarios: (a) 5% multicast traffic (low multicast intensity), and (b) 30% multicast traffic (heavy multicast intensity). For both scenarios, we assume that each multicast packet is sent to 25% of the network nodes, which is in line with what has been observed in real applications [30].

The results pertaining to the synthetic uniform-random traffic are depicted in Fig. 5(a) and (b), for 5% multicast traffic intensity, and in Fig. 5(c), (d) for 30% multicast traffic. Compared to the design in Ref. [9], SmartFork with 2 VCs per input port yields a throughput increase of 11% and 13% when 5% and 30% multicast traffic intensity is applied, respectively. When 4 VCs are used, the reaped gains are amplified to 15% and 18%, respectively, while there is no latency penalty at lower loads. In the same experiments, the 'FullSpeedup' architecture of [12] – that utilizes 5 independent read ports per input buffer – achieves negligible throughput gains over SmartFork under every scenario of multicast intensity and number of VCs, giving equal latency characteristics over the entire injection load range. The reason for this is the main observation that led to the design of SmartFork: The average branch splitting per router imposed by multicasting routing algorithms is far less than the number of output ports.

To further investigate the designs' scalability, Fig. 6 depicts the saturation throughput achieved as the VCs per input port increase. The saturation throughput improves with increasing VC numbers, albeit with diminishing returns. Nevertheless, SmartFork achieves up to 20% throughput improvement over the architecture in Ref. [9] when 8 VCs are used, while it provides almost identical saturation throughput as the FullSpeedup design in Ref. [12]. The results of Figs. 5 and 6 highlight the scalability of SmartFork with the number of VCs: as the VCs increase, the design in Refs. [9] experiences increasing HoL blocking. On the contrary, SmartFork effectively mitigates HoL blocking and reaps substantial throughput improvements that are similar to those obtained by the design in Ref. [12], i.e., an architecture with a completely independent and parallel replication scheme.

A similar trend is highlighted when investigating high radix topologies. Fig. 7 depicts the latency and the saturation throughput of all architectures under comparison, assuming uniform-random traffic in a 4×4 high-radix 2D mesh. While the network still consists of 64 source-sink endpoint pairs, each router has in total 8 input and 8 output ports: 4 are connected to adjacent routers to form the mesh topology, while the other 4 are connected to the local injection-ejection endpoints.

The results in Fig. 7 summarize the comparison between the designs in Ref. [9,12] with several versions of SmartFork, each with a different value of P. The architecture in Refs. [9] enables a higher probability of parallel/concurrent replication in a higher-radix network (due to its higher-degree crossbar). Regardless, SmartFork $P = 2$ enjoys even lower HoL blocking and reaps more performance, with a saturation throughput increase of 6.5%. Increasing SmartFork's parallelism (aka P value) further enhances its performance, leading to 11.7% and 13% improvements (as compared to Refs. [9]) for $P = 3$ and $P = 4$, respectively. On the other hand, the FullSpeedup architecture of [12], which utilizes 8

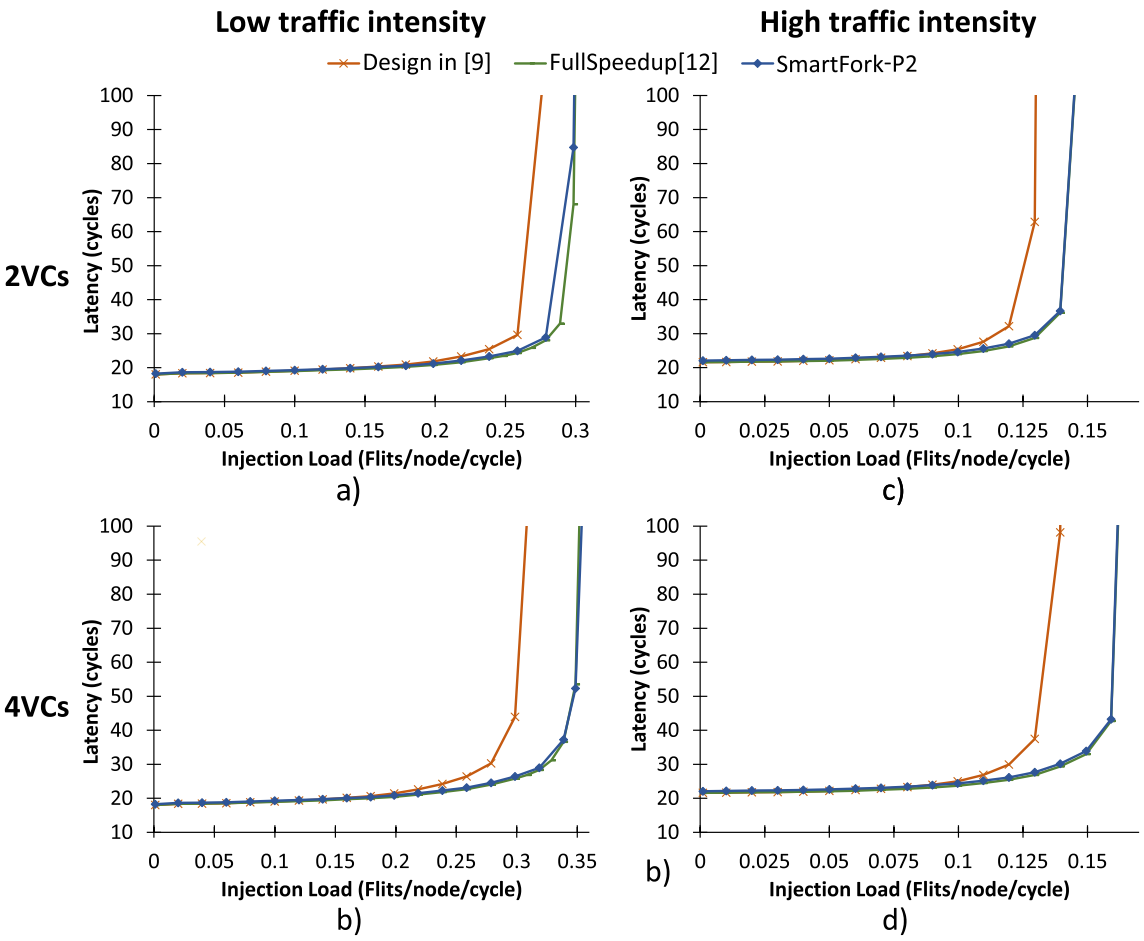


Fig. 5. Network performance results for routers with 2 and 4 VCs under synthetic uniform-random traffic with 5% multicast traffic intensity in (a) and (b) and 30% in (c) and (d) in an 8×8 2D mesh NoC.

independent read ports per input buffer, shows only a marginal throughput increase, as compared to SmartFork. Overall, SmartFork is able to always outperform the design in Ref. [9], while being flexible enough to reach any performance target by choosing a different *P* value. In fact, SmartFork can approach the performance of the FullSpeedup architecture of [12], while using fewer read ports per input buffer that translates to less hardware cost.

4.2. *Full-system performance evaluation*

In addition to evaluating the network performance under synthetic traffic (as reported in the previous sub-section), it is also imperative to evaluate the two compared architectures in a more *authentic* environment running *real application workloads*. Toward this end, we simulate a 64-core tiled CMP system running real application workloads on a commodity operating system. The execution-driven, full-system simulation framework employs Wind River's Simics [31] – which handles the functional simulation tasks – extended with the Wisconsin Multifacet GEMS simulator [32]. The latter provides a detailed timing model of the memory hierarchy and it includes the GARNET [33] cycle-accurate NoC simulator.

The architectures of SmartFork, the design in Ref. [9], and the design in Ref. [12] were implemented within GARNET. The GARNET NoC simulator cycle-accurately models the packet-switched routers, their virtual-channel buffers, allocators/arbiters, crossbars, and all inter-router links. Table 1 summarizes the salient full-system simulation parameters. Each CMP tile consists of an in-order UltraSparc III + processor core with private and separate 32 KB L1 I and D caches. The CMP has a total of 16 MB shared L2 cache (each tile has a 256 KB L2 slice; i.e., 64×256 KB = 16 MB total), and 4 GB of off-chip main memory (DRAM). The system uses a broadcast-based cache coherence protocol, which is modeled similarly to AMD's HyperTransport [34] and the Token Coherence [35] protocols. The NoC is an 8×8 2D mesh (i.e., one router per CMP tile) employing dimension-ordered XY routing for both unicast and multicast/broadcast traffic. In all architectures under comparison (i.e., SmartFork, the design in Ref. [9], and the design in Refs. [12]), each router has a single-cycle (intra-router) latency, while the inter-router link delay is also a single cycle. Each router input port has 3 VCs to accommodate the employed cache coherence protocol.

The executed applications are part of the PARSEC benchmark suite [21], which contains multi-threaded workloads from various emerging applications. All benchmarks were executed with 64 threads (one thread per processing core). The execution times reported are those of the "Regions Of Interest (ROI)", as identified in the PARSEC benchmarks. The ROI of each benchmark starts right after the initialization of the input data and ends when the computation is complete.

Fig. 8 summarizes the full-system evaluation results. To put things in perspective, a baseline reference NoC router design is also included, which has the exact same micro-architectural parameters as SmartFork and the routers in Ref. [9,12], but *no* multicast/broadcast support. Said baseline router converts multicast packets into multiple independent unicast packets.

Fig. 8(a) compares the average *network* latency of the *broadcast* packets in each benchmark application. We deliberately focus on the broadcast packets, since SmartFork and the router in Ref. [9] specifically target – and expedite the delivery of – multicast/broadcast packets. The

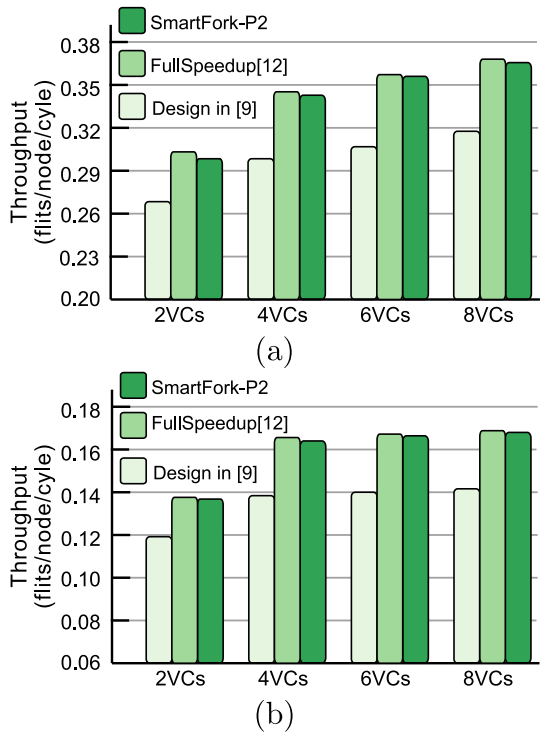
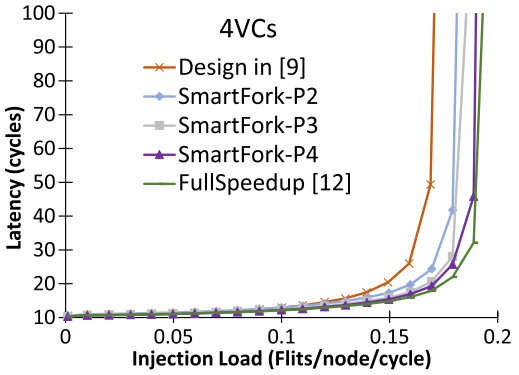


Fig. 6. Saturation throughput comparison of the router architectures utilizing up-to 8 VCs for a) 5% and b) 30% multicast traffic intensity in an 8×8 2D mesh NoC.



Saturation Throughput				
Design in [9]	SmartFork			FullSpeedup[12]
	P=2	P=3	P=4	
0.169	0.180	0.189	0.191	0.195

Fig. 7. Latency vs injection load and the corresponding saturation throughput for uniform random traffic on a high-radix 2D mesh for routers under comparison [9,12] and SmartFork with three different *P* values, assuming 4 VCs and 5% multicast traffic intensity.

percentage of packets that are broadcast ranges from 26% to 31% of all injected packets across the 9 examined benchmark applications. These broadcast percentages are an inherent attribute of the broadcast-based cache coherence protocol. The simulation results are normalized to the average network latency exhibited by broadcast packets when using the baseline reference NoC router architecture. As expected, this baseline router design offers the worst performance. As can be seen in Fig. 8 (a), SmartFork clearly outperforms the design in Ref. [9] in all benchmark applications. On average, SmartFork exhibits 30% lower network latency in the delivery of broadcast packets than the architecture in Ref. [9]. More importantly, SmartFork matches the performance of [12],

Table 1
System parameters for the execution-driven, full-system simulations.

Processor	64 in-order ×86 cores in a tiled CMP layout
OS	Linux Fedora
L1 caches	Private, separate 32 KB I&D, 4-way set associative 2-cycle latency 64B cache-line
L2 cache	Shared NUCA LLC 4-way set associative 16 MB(64cores × 256 KB slice/core) 10-cycle latency 64B cache-line
Coherence	Broadcast-based cache coherence
Main mem	4 GB, 300-cycle latency
Network	8 × 8 2D Mesh 1 cycle router delay 1 cycle link delay XY Routing 3 VCs per input port
VC size	5 flits per VC

but with a significantly lower hardware cost, as will be demonstrated in the following sub-section. By employing fewer read ports per input buffer, SmartFork is markedly more cost-effective than the design in Ref. [12], without sacrificing any real-world performance.

Network latency alone is not enough to yield insight as to overall system performance, since the interconnect is only one component within the system. Thus, it is crucial to also observe the execution times of the running applications. Fig. 8(b) compares the total execution times of the various benchmark applications. The results are normalized to the total execution times obtained when using the baseline reference NoC router architecture (that has no multicast support). Evidently, SmartFork yields faster execution times in all benchmark applications, as compared to the design in Ref. [9]; SmartFork achieves, on average, a 14% lower total execution time, which is a very significant and tangible improvement in real-world performance. Compared to the FullSpeedup design of [12], SmartFork yields near-identical execution times (less than 2% difference on average) with a much simpler hardware design.

4.3. Hardware implementation analysis

For the hardware evaluation, it is important to assess – among other pertinent metrics – the scalability of the investigated architectures with the number of VCs in each router input port. Consequently, in this subsection, we revisit the implementations evaluated in Section 4.1, i.e., for SmartFork, the design in Ref. [9], and the design in Refs. [12], we compare two variants: one with 2 VCs per input port of the NoC router, and one with 4 VCs. In all cases, router traversal has a latency of one cycle.

The 5-port NoC routers under comparison – all fully implemented in SystemVerilog – were synthesized using 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. In all NoC configurations, the flit width was set to 128 bits. To enable fair and meaningful comparisons with respect to area/power/energy, all single-cycle designs under comparison operate at 1 GHz, which is close to their maximum achievable clock frequency at the examined worst-case operating conditions.

The obtained results are summarized in Table 2. SmartFork occupies slightly more area (around 9–10%), as compared to the state-of-the-art [9]. However, SmartFork achieves much higher throughput, and, due to its efficient micro-architecture, it incurs a near-negligible power consumption overhead. In fact, SmartFork's power consumption is very similar to that of the architecture in Ref. [9]. On the other hand, the design in Ref. [12] incurs significantly higher hardware cost than SmartFork, even though it yields similar network performance. Specifically, the architecture in Refs. [12] incurs 32% and 11% higher area and power cost, respectively, than SmartFork.

To evaluate the proposed SmartFork design's area and power *efficiency*, we utilize two metrics that are derivatives of the *Kill Rule* [36]. Said rule states that the percentage gain in performance (in our case, throughput) should outweigh the percentage increase in hardware cost (i.e., in area and power). Based on this reasoning, the two metrics employed are: (a) Throughput percentage gain over area percentage

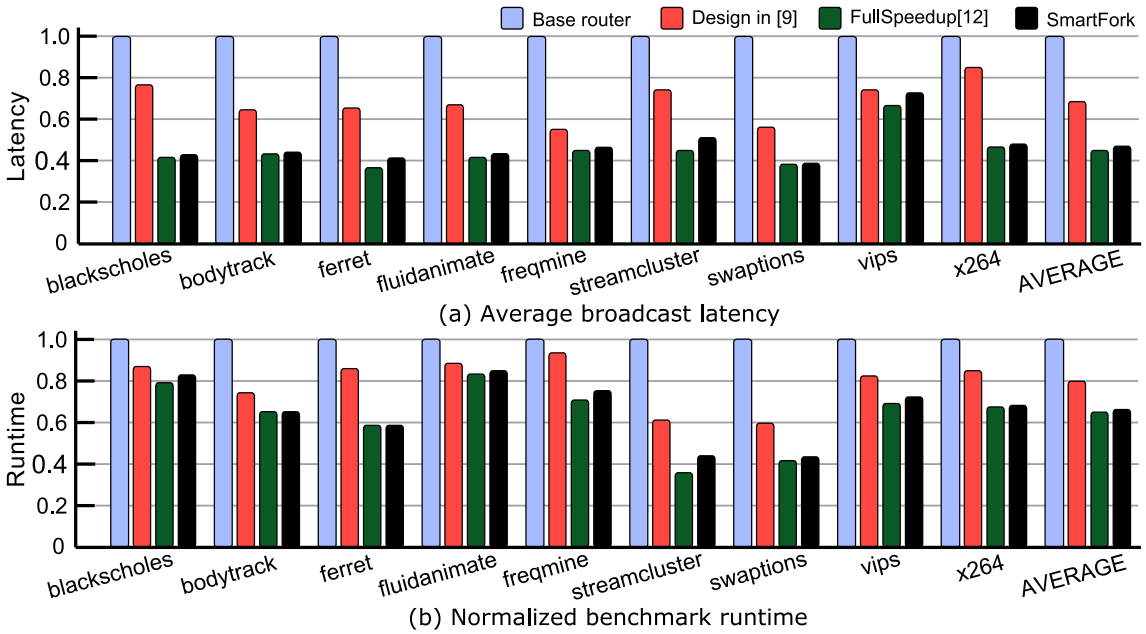


Fig. 8. Performance results using an execution-driven, full-system simulator running real applications from the PARSEC benchmark suite [21] on a 64-core CMP. For all benchmark applications, we report (a) the average network latency of the broadcast packets, and (b) the total application execution times. The results are normalized to the performance of a baseline reference NoC router with no multicast support.

Table 2
Hardware implementation results of the architectures under investigation at 45 nm technology and 0.8 V.

Design @1 GHz	VCs	Area (μm^2)	Power (mW)	Thru-put%Gain/Area%Incr.	Thru-put%Gain/Power%Incr.
Design in [9]	2	57,223	5.36		
FullyParrallel [12]		82,390	6.21	0.34	0.94
SmartFork		62,370	5.58	1.39	3.09
Design in [9]	4	118,393	10.10		
FullyParrallel [12]		171,835	11.21	0.40	1.66
SmartFork		130,442	10.25	1.73	12.37

increase, and (b) Throughput percentage gain over power percentage increase. The obtained values for these metrics are shown in the last two columns in Table 2, and they refer to the SmartFork and FullSpeedup design in Ref. [12], over the design in Ref. [9]. Obviously, in the case of SmartFork *all four values are larger than 1*, thereby indicating *higher throughput gain than the hardware cost paid.* For example, in the case of SmartFork with 4 VCs, every 1% increase in power consumption results in a 12.37% increase in reaped throughput, relative to Ref. [9].

On the contrary, despite its high performance, the FullSpeedup design of [12] is dominated by its excessive area and power requirements. In most cases, its corresponding values in the last two columns of Table 2 are less than one, signifying that more than 1% of hardware cost needs to be "paid" in exchange" for 1% of throughput gain, leading to much lower levels of efficiency, as compared to SmartFork. Overall, the two efficiency metrics clearly indicate that SmartFork is highly area- and power-efficient in increasing the achieved throughput. Hence, it constitutes a superior design choice over the FullSpeedup design in Ref. [12].

Finally, SmartFork $P = 2$ not only reduces the area of the design relative to FullSpeedup [12], but it also simplifies physical design by significantly reducing the intra-router routing congestion, as shown in Fig. 9 for 2-VC-based routers. The layout of FullSpeedup in the case of a 2D mesh reveals the severity of wiring congestion incurred by the multiple read ports. The congestion is a result of inter-connecting the 25 distinct input read ports with the 5 router output ports. On the contrary, wiring congestion in SmartFork $P = 2$ is significantly lower.

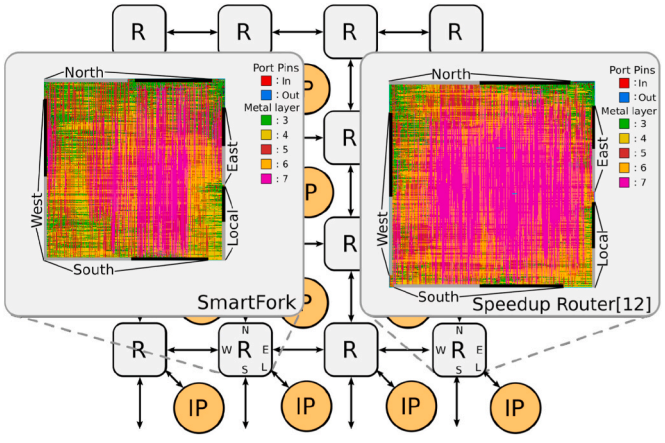


Fig. 9. Post-clock-tree-synthesis layout of a 2-VC, 5-port SmartFork implementation with $P = 2$ and with an equivalent design following FullSpeedup architecture [12].

5. Conclusions

The increasing prevalence of multicast traffic in NoCs highlights the imperative need to provide scalable multicast support in future systems. In this work, we present the novel VC-based and multicast-enabled SmartFork NoC router architecture. SmartFork relies on a flexible and

cost-efficient packet replication mechanism that can yield implementations targeting any desired point on the multicast performance spectrum. A specific variant of SmartFork – adapted to the average branch splitting observed under well-known multicast routing algorithms – is demonstrated to yield higher or almost identical throughput than the current state-of-the-art architecture, while also achieving very high area and power efficiency.

CRediT authorship contribution statement

Dimitris Konstantinou: Conceptualization, Methodology, Software, Validation, Investigation, Writing - review & editing. **Chrysostomos Nicopoulos:** Conceptualization, Methodology, Writing - review & editing. **Junghee Lee:** Methodology, Software, Validation. **Giorgos Dimitrakopoulos:** Conceptualization, Methodology, Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Arteris, Arteris IP FlexNoC AI Package, 2018.
[2] X. Lin, L.M. Ni, Deadlock-free multicast wormhole routing in multicomputer networks, Comp. Archit. News (1991) 116–125.
[3] R.V. Boppana, S. Chalasani, C.S. Raghavendra, On multicast wormhole routing in multicomputer networks, in: IEEE Symp. on Par. and Distr. Processing, Oct 1994, pp. 722–729.
[4] M. Ebrahimi, M. Daneshtalab, M.H. Neishaburi, S. Mohammadi, A. Afzali-Kusha, J. Plosila, H. Tenhunen, An efficent dynamic multicast routing protocol for distributing traffic in nocs, in: Design Automation and Test in Europe (DATE), April 2009, pp. 1064–1069.
[5] N.E. Jerger, L.-S. Peh, M. Lipasti, Virtual circuit tree multicasting: a case for on-chip hardware multicast support, Comput. Archit. News (2008) 229–240.
[6] S. Ma, N.E. Jerger, Z. Wang, Supporting efficient collective communication in nocs, in: IEEE International Symposium on High-Performance Comp Architecture (HPCA), 2012.
[7] S. Rodrigo, J. Flich, J. Duato, M. Hummel, Efficient unicast and multicast support for cmps, in: Intern. Symp. on Microarchitecture (MICRO), Nov 2008, pp. 364–375.
[8] L. Wang, Y. Jin, H. Kim, E.J. Kim, Recursive partitioning multicast: a bandwidth-efficient routing for networks-on-chip, in: Int. Symp. on Networks-on-Chip (NoCS), May 2009, pp. 64–73.
[9] T. Krishna, L.-S. Peh, B.M. Beckmann, S.K. Reinhardt, Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication, in: Int. Symp. on Microarchitecture (MICRO), 2011, pp. 71–82.
[10] R. Sivaram, D. Panda, C. Stunkel, Multicasting in irregular networks with cut-through switches using tree-based multidestination worms, in: Parallel Computer Routing and Communication, 1997, pp. 39–52.
[11] W. Hu, Z. Lu, A. Jantsch, H. Liu, Power-efficient tree-based multicast support for networks-on-chip, in: ASP-DAC, Jan 2011, pp. 363–368.
[12] K. Bhardwaj, S.M. Nowick, A continuous-time replication strategy for efficient multicast in asynchronous nocs, IEEE Trans. Very Large Scale Integr. Syst. 27 (2) (2019) 350–363.
[13] Z. Lu, B. Yin, A. Jantsch, Connection-oriented multicasting in wormhole-switched networks on chip, in: ISVLSI, March 2006.
[14] R.A. Stefan, A. Molnos, K. Goossens, daelite: a tdm noc supporting qos, multicast, and fast connection set-up, IEEE Trans. Comput. (March 2014) 583–594.
[15] M.P. Malumbres, J. Duato, J. Torrellas, An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors, in: IEEE Symp. on Par. and Dist. Processing, Oct 1996.
[16] S. Ma, N.E. Jerger, Z. Wang, Whole packet forwarding: efficient design of fully adaptive routing algorithms for networks-on-chip, in: IEEE International Symposium on High-Performance Comp Architecture (HPCA), Feb 2012, pp. 1–12.
[17] F.A. Samman, T. Hollstein, M. Glesner, Adaptive and deadlock-free tree-based multicast routing for networks-on-chip, IEEE Trans. VLSI Syst. (July 2010) 1067–1080.
[18] B. Daya, C.-H. Chen, S. Subramanian, K. Woo-Cheol, P. Sunghyun, T. Krishna, J. Holt, A.P. Chandrakasan, L. Peh, Scorpio: a 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering, in: International Symposium on Computer Architecture, June 2014, pp. 25–36.
[19] G. Dimitrakopoulos, A. Psarras, I. Seitanidis, Microarchitecture of Network-On-Chip Routers: A Designer's Perspective, 2015 plus 0.5em minus 0.4emSpringer.
[20] K. Bhardwaj, W. Jiang, S.M. Nowick, Achieving lightweight multicast in asynchronous nocs using a continuous-time multi-way read buffer, in: Int. Symp. on Networks-on-Chip (NoCS), 2017.
[21] C. Bienia, S. Kumar, J.P. Singh, K. Li, The parsec benchmark suite: characterization and architectural implications, in: International Conference on Parallel Architectures and Compilation Techniques, 2008, pp. 72–81.
[22] I. Seitanidis, A. Psarras, E. Kalligeros, C. Nicopoulos, G. Dimitrakopoulos, ElastiNoC: a self-testable distributed vc-based network-on-chip architecture, in: International Symposium on Networks-on-Chip (NOCS), 2014, pp. 135–142.
[23] A. Psarras, I. Seitanidis, C. Nicopoulos, G. Dimitrakopoulos, ShortPath: a network-on-chip router with fine-grained pipeline bypassing, IEEE Trans. Comput. 65 (10) (2016) 3136–3147.
[24] G. Dimitrakopoulos, N. Chrysos, C. Galanopoulos, Fast arbiters for on-chip network switches, in: IEEE Intern. Conf. on Computer Design (ICCD), 2008, pp. 664–670.
[25] G. Dimitrakopoulos, E. Kalligeros, K. Galanopoulos, Merged switch allocation and traversal in network-on-chip switches, IEEE Trans. Comput. 62 (10) (2013) 2001–2012.
[26] Y. Lu, C. Chen, J.V. McCanny, S. Sezer, Design of interlock-free combined allocators for networks-on-chip, in: EEE 25th International SOC Conference (SoCC), 2012, pp. 358–363.
[27] T. Krishna, J. Postman, C. Edmonds, L. Peh, P. Chiang, Swift: a swing-reduced interconnect for a token-based network-on-chip in 90nm cmos, in: 2010 IEEE International Conference on Computer Design, 2010, pp. 439–446.
[28] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004 plus 0.5em minus 0.4emUSA.
[29] L. Gurobi, Optimization, "Gurobi Optimizer Reference Manual, 2020 [Online]. Available: http://www.gurobi.com.
[30] S. Abadal, R. Martínez, J. Solé-Pareta, E. Alarcén, A. Cabellos-Aparicio, Characterization and modeling of multicast communication in cache-coherent manycore processors, Comput. Electr. Eng. 51 (2016) 168–183.
[31] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: a full system simulation platform, Computer (Feb 2002) 50–58.
[32] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset, SIGARCH Comput. Archit. News (Nov. 2005) 92–99.
[33] N. Agarwal, T. Krishna, L. Peh, N.K. Jha, Garnet: a detailed on-chip network model inside a full-system simulator, in: IEEE International Symposium on Performance Analysis of Systems and Software, April 2009, pp. 33–42.
[34] P. Conway, B. Hughes, The amd opteron northbridge architecture, IEEE Micro 27 (2007).
[35] M.M.K. Martin, M.D. Hill, D.A. Wood, Token coherence: decoupling performance and correctness, in: International Symposium on Computer Architecture (ISCA), June 2003.
[36] A. Agarwal, M. Levy, The kill rule for multicore, in: Design Automation Conference (DAC), 2007, pp. 750–753.