Achieving Lightweight Multicast in Asynchronous Networks-on-Chip Using Local Speculation

Kshitij Bhardwaj
Dept. of Computer Science
Columbia University
New York, NY 10027
kbhardwaj@cs.columbia.edu

Steven M. Nowick
Dept. of Computer Science
Columbia University
New York, NY 10027
nowick@cs.columbia.edu

ABSTRACT

We propose a lightweight parallel multicast targeting an asynchronous NoC with a variant Mesh-of-Trees topology. A novel strategy, local speculation, is introduced, where a subset of switches are speculative and always broadcast. These switches are surrounded by non-speculative switches, which throttle any redundant packets, restricting these packets to small regions. Speculative switches have simplified designs, thereby improving network performance. A hybrid network architecture is proposed to mix the speculative and non-speculative switches. For multicast benchmarks, significant performance improvements with small power savings are obtained by the new approach over a tree-based non-speculative approach. Interestingly, similar improvements are also shown for unicast. Finally, another benefit is to reduce the address field size in multicast packets.

1. INTRODUCTION

In today's many-core era, on-chip networks have major impact on system-level power and performance. Networks-on-chip (NoCs) have been an active area of research for the last decade [1], [2]. Most of the NoC research has been devoted to improving performance, power and fault-tolerance for unicast (i.e. one-to-one) traffic. However, in recent years, multicast (i.e. one-to-many) has also seen growing interest, with several optimization strategies [3].

In multicast, the same packet is sent from a source to an arbitrary subset of destinations. Multicast is widely-used in various parallel computing applications: for example, in cache coherency protocols to send write invalidates to multiple processors, in shared-operand networks for operand delivery, and in multi-threaded applications for barrier synchronization [4]. Each of these applications cause significant multicast traffic in NoCs. For example, for the *Token* cache coherence protocol, 52.4% of injected traffic is multicast [5].

There is also a growing interest in supporting multicast in NoCs using emerging technologies, such as wireless [6] and photonic [7]. Other emerging areas include large-scale neuromorphic chip multiprocessors [8] and the use of CDMA to handle multicast [9]. Both these approaches support multicast in application-specific asynchronous NoCs.

Asynchronous NoCs are at the core of designing modern globally-asynchronous locally-synchronous (GALS) systems [10]. Asynchronous NoCs eliminate the need for a global clock and are therefore free from associated overheads: clock skew, clock tree switching power and complex clock gating circuitry. Several recent examples have shown significant power and area reductions, compared to the synchronous NoCs, while achieving similar or better latency and throughput [11], [12], [13]. There is also a recent surge in industrial uptake of asynchronous NoCs: (i) IBM's TrueNorth neuromorphic chip integrates 4096 neurosynaptic cores, modeling 1 million neurons and 256 million synapses using a fully-asynchronous NoC [14], and (ii) STMicroelectronics' advanced GALS many-core system, called STHORM, that uses a fully-asynchronous NoC to connect 4 clusters, each with 16 synchronous processors [10].

Related work. There has been much research on handling multicast in synchronous NoCs [15], [16], [4], [5],[17]. These approaches are classified into two categories: *serial multicast* ("path-based") and *parallel multicast* ("tree-based"). In path-based, a multicast packet is serially routed from the source to a first destination, from where it is routed to the next sequential destination, and so on [15], [16]. This approach is simple, but not very scalable in terms of latency for large number of destinations. More widely-used is the tree-based approach, where a multicast packet is first routed from the source on a single common path towards all the destinations and is only replicated when the common path ends [4], [5], [17].

The tree-based approach achieves high-performance multicast, but can still have significant cost overheads. For example, earlier tree-based approaches required an expensive set-up phase, to configure paths taken by a multicast packet, using separate unicast packets [4], which adds to network latency, congestion and power. Other approaches avoid set-up, and simply encode all destination addresses inside the multicast packet [5]. However, this approach lowers coding efficiency and requires complex decoding logic at every switch, extra virtual channels, and turn prohibitions, thus degrading network latency. More recent work claims to achieve full chip broadcast in just 2 cycles for an 8x8 mesh [17]; multicast is performed using a combination of full chip broadcast and dropping of packets, aiming to optimize for a "dense" multicast. However, this approach can lead to considerable power overheads for multicasts to a small number of destinations.

Although almost all of the above multicast research targets a mesh topology, another viable option is to use tree-based topologies such as Mesh-of-Trees (MoT). For some applications, MoT has been shown to achieve higher performance compared to mesh topologies [18]. A variant of the traditional MoT topology has been proposed that achieves higher saturation throughput with lower contention compared to the original MoT topology [19]. Variant MoTs have been recently used for core-to-cache connections in high performance shared memory parallel processors [20].

Contributions. Given the overheads in recent synchronous multicast approaches, there is a need for more cost-effective multicast solutions. The focus of this paper is on asynchronous NoCs, driven by their potential for lightweight design, and the growing interest in their use for large-scale system integration. However, despite much recent activity in this area, there are no general-purpose asynchronous NoCs with multicast capability. The goal of this work is to achieve lightweight and power-performance efficient multicast using asynchronous NoCs. We propose multiple solutions to support efficient multicast, in asynchronous NoCs, using a new routing strategy and network architectures.

This work was partially supported by NSF Grants CCF-1219013 and CCF-1527796.

DAC'16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2897978

The first contribution is a lightweight tree-based parallel multicast using an asynchronous NoC. The target topology is MoT. *To the best of our knowledge, this is the first general-purpose asynchronous NoC to support multicast.* This solution, although simple, is not very efficient in terms of performance. Further enhancements are therefore proposed for performance, while still maintaining design simplicity.

The second contribution is a novel strategy called *local speculation* that achieves high-performance parallel multicast. In local speculation, a packet (unicast or multicast) is *always* broadcast at a fixed subset of *speculative* switches in the network. To restrict the distance traveled by any redundant packets to small "local" regions, these packets are throttled by neighboring *non-speculative* switches. This localized approach limits the penalties of speculation, by terminating redundant packets early, resulting in minimal impact on congestion and power. Speculative switches are built for speed, as they do not do any route computation or output channel allocation. Non-speculative switches perform throttling with almost no hardware overhead. Interestingly, local speculation improves network performance not only for multicast, but also for unicast. In addition, the simple, 'sub-cycle' operation for local broadcast and low-overhead throttling, not discretized to clock cycles, make the new paradigm a good match for asynchronous design.

As a third contribution, local speculation leads to a new *hybrid network architecture* that mixes speculative and non-speculative switches. This is the first time a hybrid NoC has been used to support multicast (cf. [4], [5], [17]). Moreover, unlike previous unicast work that uses speculation within a router for early VC allocation [1], this work uses speculation at the network-level to support efficient multicast.

As a fourth contribution, two more architectures are introduced besides hybrid, with *extreme degrees of speculation* for an exhaustive design space exploration. The first does not use any speculation, which is the same as our first tree-based multicast solution. In contrast, the second is an almost fully speculative architecture.

Finally, as a fifth contribution, *protocol optimizations* are introduced, to further improve the power and performance of speculative and non-speculative nodes. These optimizations are performed only for multi-flit packets, and are triggered by the header flit. For speculative switches, once the header is processed, the switch can revert to non-speculative mode for body and tail flits, thereby saving power. For non-speculative switches, once the header is processed, the channel remains allocated for the current packet, with no repeated route computation, thereby improving throughput.

The above new techniques have been incorporated into new networks and extensive experiments are performed. For multicast benchmarks, the new simple tree-based parallel multicast network achieves 39.1-74.1% lower latency, with only small power overheads, than a unicast-based serial multicast approach. The hybrid network with local speculation provides additional 17.8-21.4% latency improvements, and small power reductions, over our simple tree-based solution. We also consider an extreme case of an almost fully speculative network, which has the best performance but incurs 10.8-13.4% power overheads over the hybrid solution. Interestingly, similar results are also seen for unicast traffic.

2. BACKGROUND

This section presents background on asynchronous communication, a variant MoT topology, and a baseline asynchronous NoC, which form the foundation of the new work.

Asynchronous communication. A handshaking protocol defines the channel communication between an asynchronous sender and receiver, using request/acknowledge wires. A *two-phase (NRZ) protocol* has only 2 events per transaction (toggle on req followed by toggle on ack) [21], while a *four-phase (RZ) protocol* has 4 total events (req/ack initially zero, with rising then falling req/ack transitions) [12], [14]. This paper uses *two-phase*, as it has only a single roundtrip channel communication per transaction, whereas *four-phase* has two roundtrip communications, since the former leads to better throughput. High-performance, synchronous style *single-rail bundled data* is used for data encoding in this paper [21], [13], [11], rather than delay-insensitive encoding [12], [14], [10].

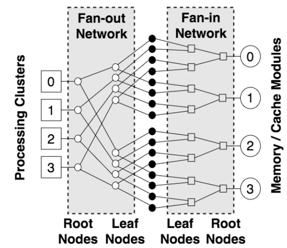


Figure 1: 4x4 MoT network architecture

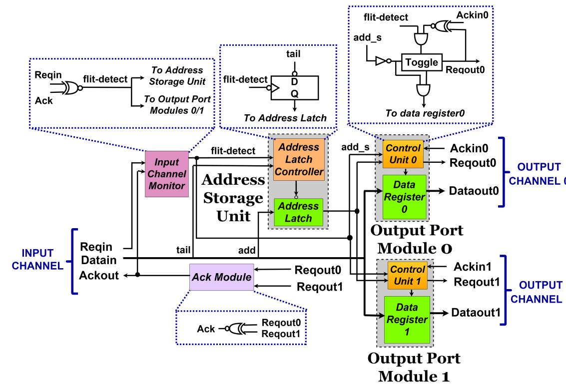


Figure 2: Baseline fanout node

Variant Mesh-of-Trees topology. The architecture of a 4x4 variant MoT network is shown in Figure 1. It contains two binary trees: a fanout tree composed of *routing (i.e. fanout) nodes,* and a fanin tree composed of *arbitration (i.e. fanin) nodes* [10], [19]. The trees are mirror copies: fanout network from a source root to leaves, and the fanin network from leaves to a destination root. This topology provides two key advantages: (a) hop count from source to destination is always a small constant, log(n), leading to low latency, and (b) each distinct source-destination pair has a unique path through the network, which can minimize network contention significantly. However, the lack of path diversity can be a potential performance bottleneck for some adversarial cases, where traffic is extremely unbalanced. Overall, though, recent results demonstrate significant benefits for saturation throughput [20], [19], [21].

The system shown in Figure 1 will require multicast support to implement cache-coherence such as using invalidation-based snoopy protocol, where multicast traffic goes from processors to caches, or caches to processors [4].

Baseline asynchronous network. An efficient asynchronous NoC implementation of this variant MoT topology was recently introduced [21]. The target is an 8x8 network. The approach can only support unicast traffic. As shown in Figure 1, a fanout node receives packets on one input channel and forwards them to one of two output channels, based on the destination address. In contrast, a fanin node receives packets on two input channels, performs arbitration, and forwards the winning packet on the single output channel. *Source routing* is used [2], where each packet header contains the address fields of every fanout node on its path. In this case, each such address field is only 1-bit wide.

In this paper, the above network will be significantly enhanced to support multicast operation. Only fanout nodes will be modified; existing fanin nodes are directly reused. In particular, fanout nodes are responsible for all routing, so must support distribution of multiple packet copies; hence they must be instrumented with new addressing and repli-

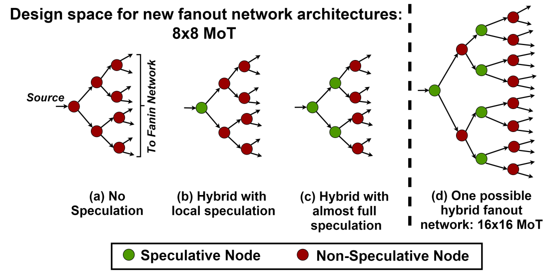


Figure 3: New fanout network architectures: (a)-(c) full range for 8x8 MoT, (d) One possible hybrid network for 16x16 MoT

cation capability. In contrast, even with multicast, the existing fanin network will still direct all packets to their destinations. Hence, this paper focuses only on the fanout nodes.

The micro-architecture of a baseline fanout node is shown in Figure 2. There are 5 main components: *Input Channel Monitor, Address Storage Unit,* two *Output Port Modules*, and *Ack Module*. The Input Channel Monitor detects the arrival of each flit of a packet. The Address Storage Unit stores the address of the header flit, which it holds until after arrival of the tail flit. The two Output Port modules, which are *normally opaque,* manage routing and flow control of each output channel. Finally, the Ack Module observes when either output channel transmits a flit, in unicast traffic, then completes handshaking on the input channel.

The fanout node has relatively simple operation [21]. When a packet arrives on the input channel, the header is directed to inputs of both Output Port Modules. The Input Channel Monitor detects the arrival of the flit, enabling storing of its address in the Address Storage Unit. The monitoring *flit-detect* also partially-enables both Output Modules; the one receiving the correct address is then activated, and the flit is sent out on that output channel along with a toggled *Reqout*. This signal enables two concurrent operations: closing of the Output Port Module (for data protection) and enabling of the Ack Module to generate the acknowledge on the input channel. Finally, the downstream node acknowledges (toggles the *Ackin* signal), completing the handshaking on the output channel. Similar operation occurs for the remaining body and tail flits.

3. PROPOSED APPROACH: OVERVIEW

This section introduces multiple solutions to achieve efficient parallel multicast, using new routing strategy and network architectures. Protocol optimizations of the new fanout nodes and five new networks based on these architectures and optimizations are also presented.

Simple tree-based multicast. The first contribution is a simple tree-based parallel multicast, applied for the first time to a general-purpose asynchronous NoC. Routing of a unicast packet is the same as in the baseline network. The fanout network architecture, in Figure 3(a), has all non-speculative nodes. New fanout nodes are designed to handle parallel replication, as described in Section 4(b).

Source routing is used to encode the address for every fanout node on each path to the destination(s). The address at each fanout node must encode 3 symbols: top route, bottom route or both. Therefore, 2-bit encoding is used for the address field of each fanout node.

This basic tree-based multicast is simple but not efficient in terms of latency and throughput. The new fanout nodes are slow due to expensive route computation and channel allocation protocols, required to handle a more complex set of transmission modes. Another limitation is that the source routing, as described above, leads to low packet coding efficiency, which does not scale with larger network sizes.

Local speculation-based multicast. A new strategy, local speculation, is introduced for high-performance parallel multicast. In local speculation, a subset of fanout nodes are speculative and always broadcast a multicast (or unicast) packet. These nodes are surrounded by non-

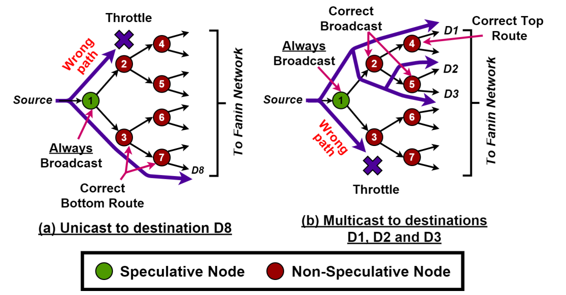


Figure 4: Unicast/multicast simulations: hybrid network

speculative nodes that always send packets on the right path(s) and throttle any redundant packets from the speculative nodes, restricting these packets to small "local" regions. A hybrid network architecture is introduced to mix these two types of fanout nodes. Figure 3(b) shows one possible hybrid fanout network for an 8x8 MoT. The detailed design of a speculative node is presented in Section 4(a).

The hybrid network also improves packet coding efficiency using a *simplified source routing*, which does not encode the addressing for the speculative nodes on the path to the destination(s). This simplification is a direct consequence of the simplicity of speculative nodes that always broadcast and therefore do not require any addressing. As a result, only a *subset* of fanout nodes (i.e. non-speculative ones) require address fields in the packet header.

The operation of the hybrid fanout network is illustrated using two simulations: for unicast and multicast.

Figure 4(a) shows the routing of a unicast packet in an 8x8 MoT network. The packet is first broadcast by the speculative root node, sending one copy on the right path and the other on the wrong path. The copy on the wrong path is throttled in the top sub-tree by non-speculative node 2. The copy on the right path is forwarded by non-speculative nodes 3 and 7 through their bottom output ports, based on actual addressing, towards D8.

Figure 4(b) shows the routing of a multicast packet. Similar to unicast, the speculative root node broadcasts, sending copies on the right and wrong paths. The latter is throttled in the bottom sub-tree by non-speculative node 3. As non-speculative nodes can *also* broadcast, the copy on the right path is broadcast by node 2 on both output channels. One copy is correctly routed to D1 by node 4 through its top output port, while the other is correctly routed to D2 and D3 by another broadcast at node 5.

So far, two architectural design points have been covered, non-speculative and hybrid, as shown in Figures 3(a) and (b). To complete the design space, a third extreme point is introduced, an almost fully speculative architecture. As shown in Figure 3(c), only the last level must be non-speculative, since the fanin network cannot throttle any misrouted packets. Such global speculation can achieve high performance, but suffers from major power overheads due to the large distances traveled by misrouted packets.

While the focus of the discussion of the above contributions and the evaluations in this paper is on 8x8 MoT, it is interesting to consider future directions of larger-sized networks. The hybrid architecture for the larger networks has more degrees of freedom to mix the speculative and non-speculative nodes and therefore a wider design space. Figure 3(d) shows one possible hybrid fanout network for a 16x16 MoT, out of a family of many possibilities.

Protocol optimizations. The above speculative and non-speculative nodes are efficient, but also suffer from overheads, which are minimized for multi-flit packets using protocol optimizations triggered by the header. Speculative nodes can create redundant copies and therefore lead to extra switching power. Non-speculative nodes have complex route computation and channel allocation logic.

Extra power due to speculative nodes is minimized by reverting to the non-speculative mode for body flits of pack-

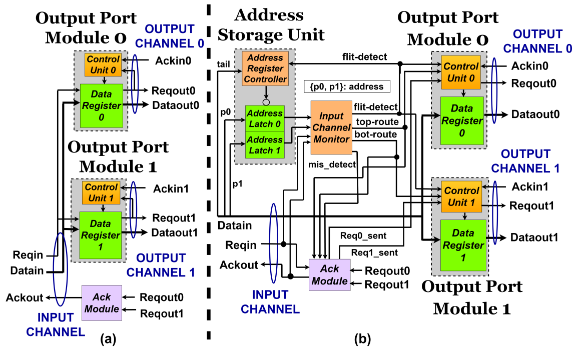


Figure 5: Unoptimized fanout nodes: (a) speculative, (b) non-speculative

ets intended for only one direction. Routing of the header is used by the node to close the output port on the wrong path before the trailing body flits arrive. Therefore, no redundant copies are created for these body flits. Arrival of tail flit is used by the node to return to always broadcast state. These optimized nodes are called as *power-optimized speculative nodes,* and are described in Section 4(c).

Latency and throughput of non-speculative nodes is optimized using channel pre-allocation. Routing of the header is used to reserve the correct output channel(s) for the remainder of the packet (body/tail flits). These flits are then fast forwarded through the output channel(s) after their arrival, optimizing latency of these flits and improving overall network latency and throughput. These optimized nodes are called as *performance-optimized non-speculative nodes,* and are described in Section 4(d).

Target parallel multicast networks. The above parallel multicast approaches are incorporated into five network configurations, representing three distinct points in the design space of speculative architectures: (i) *non-speculative,* (ii) *hybrid,* and (iii) *almost fully speculative.* The goal is to explore the tradeoffs associated with varying degrees of speculation and protocol optimizations.

In particular, the paper targets two non-speculative networks (**BasicNonSpeculative, OptNonSpeculative**), two hybrid networks (**BasicHybridSpeculative, OptHybridSpeculative**), and one extreme case of a nearly fully speculative network, with non-speculative nodes only at its leaves (**OptAllSpeculative**). To support the design of these networks, four distinct fanout nodes are introduced.

4. PROPOSED FANOUT NODE DESIGNS

This section presents the design and operation of the new fanout nodes, which are the main building blocks of the new parallel multicast networks. The basic new networks are composed of the unoptimized fanout nodes, and the more advanced new networks are composed of the power- and performance-optimized fanout nodes, discussed in turn.

(a) Unoptimized speculative fanout node.

Structure. The node's micro-architecture is shown in Figure 5(a). There are three main differences from the baseline fanout: (i) drastically simplified design, due to elimination of the Input Channel Monitor and Address Storage Unit; (ii) new normally-transparent Output Port Modules; and (iii) a new Ack Module. The node does no route computation, therefore an Address Storage Unit is not needed. In addition, since the node simply broadcasts every flit arriving on the input channel, Output Port Modules are also considerably simplified and now only do flow control with normally-transparent data registers. Finally, the Ack Module now completes handshaking on the input channel only after a flit is sent on both output channels, hence a *C-element* is used [12], as opposed to an *XOR* gate in baseline.

Operation. This node has a very simple operation, which is the same for any type of packet and its flits: always broadcast. A packet on the input channel can be a correctly routed unicast, or multicast going to either or both outputs, or any misrouted packet from previous node. When a flit arrives on the input channel, it is directed to both output channels, along with the generation of *Reqouts*. These *Reqouts* perform two concurrent operations: close the Output Port Modules for data protection, and enable Ack Module to generate *Ack*. Finally, when the downstream nodes toggle *Ackin(s)*, handshaking is complete on the output channel(s).

(b) Unoptimized non-speculative fanout node.

Structure. The micro-architecture of the new node is shown in Figure 5(b). Overall, the structure is similar to the baseline fanout with identical key components. However, all these units are now more complex, to support parallel replication for multicast and throttling of any misrouted packets. As illustrated in Section 3, these nodes use a 2-bit source routing address to support multicast. The address is passed through the Input Channel Monitor, both as routing information for the Output Port Modules and also to notify the Ack Module of any misrouted packets on the input channel. The Ack Module observes the output channels and the input channel. Handshaking is completed on the input channel for three cases: if a flit is sent out on exactly one of the output channels, or both output channels, or if it is a misrouted flit. The Ack Module also notifies the Output Port Module(s) control unit, when a flit has been sent on the corresponding output channel(s).

Operation. Three packet types can arrive on a node's input channel: unicast, multicast (going to one or two output ports), and a misrouted packet from the previous node.

In case of a unicast packet, the header is first directed to both Output Port Modules. The Input Channel Monitor detects the flit arrival, enables storing of the address and also partly enables both Output Port Modules. Depending on the address, the monitor generates the *top-route/bot-route* routing signals to enable the correct Output Port Module(s). Generation of *Reqout* leads to three concurrent operations: closing the Output Port Module for data protection, enabling Ack Module to generate *Req0/1_sent* control signals, and completing handshaking on the input channel. *Req0/1_sent* signals identify to the Output Port Modules if the flit on the input channel is new or stale; they disable the Output Port Module, right after a flit is sent out on the output channel, hence avoiding any potential resampling. Similar operations occur for body/tail flits.

In case of multicast packet, if intended for exactly one direction, a similar protocol to the unicast packet is followed. For multicast going to both outputs, the Input Channel Monitor enables both output ports for routing. After *Reqouts* are generated on both output channels, in parallel, the Ack Module completes handshaking on the input channel. All internal operations (data protection, no resampling) are similar to unicast, but now done for both Output Port Modules. Finally, for a misrouted packet on the input channel, the Input Channel Monitor detects this packet, and enables Ack Module to complete handshaking on the input channel.

(c) Optimized speculative fanout node. The idea of this optimization is to speculate on the header flit, then use actual address information to switch to a non-speculative operation for all body flits, thereby saving power.

Structure. There are three main differences from the basic speculative nodes: (i) the new Input Channel Monitor is instrumented to detect the arrival of flits, and the tail, on the input channel, (ii) more complex Output Port Modules, which can revert to non-speculative mode for body flits, and (iii) a new Ack Module that generates *Ack* for two different cases: body flits which are routed correctly only on one output channel, all other flits. For the former, *Ack* is sent after the flit is routed on exactly one output channel; for the latter, the protocol is same as the basic speculative nodes.

Operation. Once a header arrives, it is speculatively routed on both output channels. Its address is used by the Output Port Modules to identify the correct route (top, bottom, or both), and to block the incorrect route for all body flits. These modules return to their default normally transparent state after the tail arrives. Therefore, the tail also gets speculatively routed to both output channels.

(d) Optimized non-speculative fanout node. The basic idea of optimization is to use the header to pre-allocate the correct output channel(s) for body/tail. These flits are fast forwarded on their arrival, without route computation and output channel allocation.

Structure and operation. There is only one key difference in terms of more simplified Output Port Modules. Based on the routing of the header, these modules now pre-allocate the correct channel for trailing body/tail flits. The routing of the tail is used to release the channel.

5. EXPERIMENTAL RESULTS

This section presents the experimental framework for evaluation of the new parallel multicast solutions, along with node and network-level results on area, performance, power, and addressing overhead.

5.1 Experimental Framework

Experimental case studies. Two distinct case studies are used to evaluate the proposed parallel multicast solutions: (a) *contribution trajectory*, and (b) *architectural design space exploration.* The contribution trajectory incrementally evaluates the effectiveness of each contribution, against a serial baseline: parallel multicast, local speculation and a hybrid network, and protocol optimizations. Architectural design space exploration, on the other hand, only evaluates the effects of varying degrees of speculation on the new parallel multicast networks. To isolate the focus, only optimized networks are targeted, thereby eliminating any interference from the optimization strategies.

The **contribution trajectory** compares 4 networks: (i) *Baseline* [21], only supporting serial multicast; (ii) *BasicNonSpeculative*, using simple tree-based parallel multicast; (iii) *BasicHybridSpeculative*, using local speculation in a hybrid network; and (iv) *OptHybridSpeculative*, similar to the previous one, but including protocol optimizations.

The **architecture design space exploration** compares 3 optimized new networks with varying degrees of speculation: (i) *OptNonSpeculative*, with no speculation; (ii) *OptHybridSpeculative*, with local speculation; and (iii) *OptAllSpeculative*, with almost full speculation.

Experimental setup. Six different 8x8 MoT networks are implemented using FreePDK Nangate 45 nm technology. Designs are technology-mapped and pre-layout. Six types of nodes are implemented, as building blocks: five fanout and one fanin. Nodes are mapped to the Nangate standard cell library in the Cadence Virtuoso tool. Accurate gate-level models are extracted using the Spectre simulator (typical process corner), to determine rise/fall times for every I/O path of each gate. Channel lengths and delays are borrowed from a synchronous MoT chip [21] and scaled to 45 nm technology. These extracted models of nodes and channels are used to implement the networks in structural Verilog.

An asynchronous NoC simulator is used for both unicast and multicast traffic. It includes a Programming Language Interface (PLI) to connect a C-based traffic generator and test environment to the technology-mapped network. A fixed packet size of 5 flits is used. Injection of headers of different packets follows an exponential distribution. A procedure similar to [2] is followed to ensure long warmup and measurement phases. Two steps are used to measure power: (i) record and annotate precise switching activity of every wire in the network over a benchmark run, and (ii) compute total power using the Synopsys PrimeTime tool.

Benchmarks. Experiments are conducted on six synthetic benchmarks. There are 3 unicast benchmarks [2]: 1) *Uniform random,* 2) *Bit permutation:shuffle,* and 3) *Hotspot.* There are 3 multicast benchmarks: 4) *Multicast5* and 5) *Multicast10,* where all sources inject multicast traffic at rates of 5% and 10%, respectively, to random subsets of destinations, and otherwise do uniform random unicast, and 6) *Multicast_static,* where 3 sources perform only random multicast, and the others do only uniform random unicast.

5.2 Node- and Network-Level Results

(a) Node-level results. Area and latency of the four new fanout nodes (Section 4) and *Baseline* fanout were evaluated. The unoptimized speculative nodes, due to their simplicity, have significantly lower area and latency (247 μm^2, 52 ps) than *Baseline* (342 μm^2, 263 ps). The more complex unoptimized non-speculative nodes have only small overhead (406 μm^2, 299 ps) over *Baseline.* The optimized speculative nodes have moderate cost increases (373 μm^2, 120 ps) over unoptimized, but will provide substantial network-level power savings (see below). Interestingly, the optimized non-speculative nodes have slightly lower costs (366 μm^2, 279 ps) than the unoptimized ones.

(b) Contribution trajectory. This first case study explores the incremental impact of each key contribution: parallel multicast, local speculation, and optimizations.

Network latency. Figure 6(a) shows the average network latency results. We measure latency of each network at 25% of the saturation throughput of that network, up to the arrival of all headers at destinations. This load is high enough to show the impact of different benchmarks, while keeping the network largely uncongested. Moreover, long warmup and measurement times are used, for example, for Uniform Random/Multicast_static benchmarks, warmup is 320 ns/640 ns, and measurement is 3200 ns/6400 ns with injection of 2100/4000 flits at each active source.

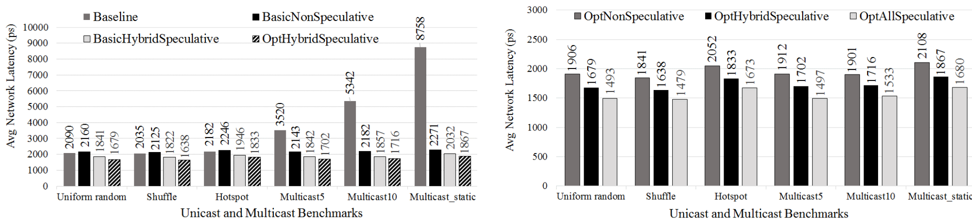
For multicast benchmarks, the simple tree-based parallel multicast network, *BasicNonSpeculative*, obtained significant benefits over the serial *Baseline*, from 39.1% (Multicast5) to 74.1% (Multicast_static), highlighting the severe overheads of the serial multicast approach. The *BasicHybridSpeculative* and *OptHybridSpeculative* show further improvements of 10.5-14.9% and 17.8-21.4%, respectively, over the *BasicNonSpeculative,* illustrating the individual benefits of hybrid design and optimizations.

For unicast benchmarks, *BasicNonSpeculative* incurs a small latency overhead over *Baseline:* since unicast is serial, the added node complexity to support parallel multicast becomes an overhead. However, the two hybrid networks provide noticeable benefits over *BasicNonSpeculative,* following similar trends as observed with multicast benchmarks. Interestingly, these latter results show that local speculation can significantly accelerate unicast traffic due to very fast speculative nodes.

Saturation throughput. Table 1 shows saturation throughput results. For multicast benchmarks, the new simple parallel network, *BasicNonSpeculative*, shows considerable benefits over the serial *Baseline*, ranging from 14.8% (Multicast5) to 39.5% (Multicast_static). The two hybrid networks exhibit additional improvements up to 9.5% and 19.7%, respectively, over *BasicNonSpeculative,* demonstrating that local speculation, with accelerated packet transmission, provides a higher threshold for saturation.

For unicast benchmarks, results are more complex. Hotspot is highly-adversarial, with identical throughput for every network. For Uniform random, the *OptHybridSpeculative* network showed substantial improvements (28.0%) over *BasicNonSpeculative.* For Shuffle, two new networks show moderate throughput degradation (*BasicNonSpeculative, BasicHybridSpeculative*) over the *Baseline*, while *OptHybridSpeculative* obtains 32.8% higher throughput than *BasicNonSpeculative* and 9.5% higher than *Baseline.*

Total network power. Table 1 shows power results for 4 benchmarks. An injection rate that is 25% saturation load measured in *Baseline,* for a normalized comparison of energy per packet. Overall, as expected, *Baseline* has the lowest power due to its low complexity and serial multicast approach. *BasicNonSpeculative* has moderate overhead over *Baseline* (5.8-11.9%), due to more complex nodes. The overhead increases significantly for *BasicHybridSpeculative*



(a) Contribution trajectory (b) Architectural design space exploration

Figure 6: Network latency at 25% saturation load of respective networks

	Saturation Throughput (GF/s)						Total Network Power (mW)			
Schemes/benchmarks	*Uniform-random*	*Shuffle*	*Hotspot*	*Multi-cast5*	*Multi-cast10*	*Multicast-static*	*Uniform-random*	*Hotspot*	*Multi-cast5*	*Multi-cast10*
	Contribution trajectory									
Baseline	1.26	1.48	0.29	1.28	1.28	1.29	12.6	3.8	14.7	17.1
BasicNonSpeculative	1.25	1.22	0.29	1.47	1.63	1.80	14.1	4.2	16.0	18.1
BasicHybridSpeculative	1.42	1.25	0.29	1.61	1.73	1.87	15.6	4.5	17.4	19.4
OptHybridSpeculative	1.60	1.62	0.29	1.76	1.84	1.96	13.9	4.1	15.7	17.6
	Architecture design space exploration									
OptNonSpeculative	1.52	1.57	0.29	1.72	1.82	1.93	13.1	3.9	15.0	17.0
OptHybridSpeculative	1.60	1.62	0.29	1.76	1.84	1.96	13.9	4.1	15.7	17.6
OptAllSpeculative	1.65	1.70	0.29	1.78	1.84	1.96	16.1	4.6	17.8	19.5

Table 1: Saturation throughput and total network power results

(13.4-23.8% over *Baseline*), due to creation of redundant speculative copies. However, using *OptHybridSpeculative*, most of this overhead is removed (only 2.9-10.3% over *Baseline*): due to elimination of all redundant body flits (speculative nodes), and reduced switching activity because of channel pre-allocation (non-speculative nodes).

(c) Architectural design space exploration. This second case study only includes evaluations of the optimized designs, while varying the degree of speculation.

Network latency. As shown in Figure 6(b), the hybrid network with local speculation (*OptHybridSpeculative*) achieves 9.7-11.9% latency improvements (unicast and multicast) over *OptNonSpeculative*, showing the effectiveness of the proposed techniques. The extreme case, *OptAllSpeculative*, exhibits 8.7-12.0% additional latency improvements over *OptHybridSpeculative* (18.5-21.7% over *OptNonSpeculative*), due to its almost fully speculative architecture (but will have significant power overheads).

Saturation throughput. For all benchmarks, in Table 1, the hybrid approach (*OptHybridSpeculative*) and extreme speculation (*OptAllSpeculative*) have nearly identical throughput to the non-speculative (*OptNonSpeculative*).

Total network power. Interestingly, even with its significant performance benefits, the optimized hybrid approach incurs only minor power overheads of 3.5-6.1% over the non-speculative approach, since redundant copies are restricted to small local regions, and a power-oriented optimization is applied to disable speculation for body flits. In contrast, the fully-speculative approach (*OptAllSpeculative*) incurs considerable power overheads (10.8-15.8% over *OptHybridSpeculative*, 14.7-22.9% over *OptNonSpeculative*) due to larger regions of speculation in *OptAllSpeculative*. It is expected that with larger MoT networks, these overheads will only increase, due to wider speculative regions.

(d) Addressing scheme comparisons. As highlighted earlier, an additional benefit of local speculation is to reduce the address field size. The serial Baseline has the shortest address field, using source routing, with a 1-bit address per fanout node on a unicast path: 3 bits for 8x8 MoT, and 4 bits for 16x16 MoT (not evaluated in this paper). However, the large performance overheads of these designs make them impractical for multicast.

Of the three proposed parallel architectures, each using source routing, address field sizes for an 8x8 MoT are: 14 bits in *non-speculative*, 12 bits in *hybrid*, and 8 bits in *almost fully-speculative*. For a 16x16 MoT, the benefits of speculation are even greater: 30, 20 and 16 bits, respectively. Effectively, the speculative architectures reduce the total number of address fields, by only addressing non-speculative nodes.

6. CONCLUSIONS AND FUTURE WORK

The paper presents a new lightweight multicast using Mesh-of-Trees based asynchronous NoCs. A new strategy, local speculation, is introduced, where fixed speculative switches always broadcast, but redundant packets are restricted to small regions. A hybrid network architecture is proposed, mixing speculative and non-speculative switches. For multicast, the network achieves 17.8-21.4% improvements in network latency with small power reductions over a tree-based non-speculative approach. The approach is the first general-purpose multicast for asynchronous NoCs. For future work, we plan to extend the approach to larger MoT networks, alternative topologies (e.g. 2D-mesh), as well as synchronous NoCs.

7. REFERENCES

[1] R. Marculescu, Y. Ogras, L.-S. Peh, N. D. E. Jerger, and Y. V. Hoskote, “Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives,” *TCAD*, vol. 28, pp. 3–21, 2009.

[2] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[3] D. Bertozzi, G. Dimitrakopoulos, J. Flich, and S. Sonntag, “The fast evolving landscape of on-chip communication - selected future challenges and research avenues,” *Design Autom. for Emb. Sys.*, vol. 19, pp. 59–76, 2015.

[4] N. D. E. Jerger, L. Peh, and M. H. Lipasti, “Virtual circuit tree multicasting: A case for on-chip hardware multicast support,” in *ISCA*, 2008, pp. 229–240.

[5] T. Krishna, L. Peh, B. M. Beckmann, and S. K. Reinhardt, “Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication,” in *MICRO*, 2011, pp. 71–82.

[6] S. Deb, A. Ganguly, K. Chang, P. P. Pande, B. Belzer, and D. H. Heo, “Enhancing performance of network-on-chip architectures with millimeter-wave wireless interconnects,” in *Conference on Application-Specific Systems Architectures and Processors*, 2010, pp. 73–80.

[7] C. Li, M. Browning, P. Gratz, and S. Palermo, “Energy-efficient optical broadcast for nanophotonic networks-on-chip,” in *Optical Interconnects Conference*, 2012, pp. 64–65.

[8] P. Merolla, J. V. Arthur, R. Alvarez-Icaza, J. Bussat, and K. Boahen, “A multicast tree router for multichip neuromorphic systems,” *IEEE Trans. on Circuits and Systems*, vol. 61-I, pp. 820–833, 2014.

[9] X. Wang, T. Ahonen, and J. Nurmi, “Applying CDMA technique to network-on-chip,” *IEEE Trans. VLSI Syst.*, vol. 15, pp. 1091–1100, 2007.

[10] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, “P2012: building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator,” in *DATE*, 2012, pp. 983–987.

[11] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, “A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems,” in *DATE*, 2013, pp. 332–337.

[12] Y. Thonnart, P. Vivet, and F. Clermidy, “A fully-asynchronous low-power framework for GALS NoC integration,” in *DATE*, 2010, pp. 33–38.

[13] D. Gebhardt, J. You, and K. S. Stevens, “Comparing energy and latency of asynchronous and synchronous NoCs for embedded SoCs,” in *NOCS*, 2010, pp. 115–122.

[14] P. Merolla *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[15] M. Daneshtalab, M. Ebrahimi, S. Mohammadi, and A. Afzali-Kusha, “Low-distance path-based multicast routing algorithm for network-on-chips,” *IET Computers & Digital Techniques*, vol. 3, no. 5, pp. 430–442, 2009.

[16] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, J. Flich, and H. Tenhunen, “Path-based partitioning methods for 3D networks-on-chip with minimal adaptive routing,” *IEEE Trans. Computers*, vol. 63, pp. 718–733, 2014.

[17] T. Krishna and L. Peh, “Single-cycle collective communication over a shared network fabric,” in *NOCS*, 2014, pp. 1–8.

[18] S. Kundu and S. Chattopadhyay, “Mesh-of-tree deterministic routing for network-on-chip architecture,” in *GLSVLSI*, 2008, pp. 343–346.

[19] A. O. Balkan, M. N. Horak, G. Qu, and U. Vishkin, “Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing,” in *HOTI*, 2007, pp. 21–28.

[20] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, “A fully-synthesizable single-cycle interconnection network for shared-L1 processor clusters,” in *DATE*, 2011, pp. 491–496.

[21] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, “A low-overhead asynchronous interconnection network for GALS chip multiprocessors,” *TCAD*, vol. 30, pp. 494–507, 2011.