To appear in the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA-2012). This is the authors' edition.

# Supporting Efficient Collective Communication in NoCs

Sheng Ma[†‡], Natalie Enright Jerger[‡], Zhiying Wang[†]
[†]School of Computer, National University of Defense Technology, Changsha, China
[‡]Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada
masheng@nudt.edu.cn, enright@eecg.toronto.edu, zywang@nudt.edu.cn

## Abstract

*Across many architectures and parallel programming paradigms, collective communication plays a key role in performance and correctness. Hardware support is necessary to prevent important collective communication from becoming a system bottleneck. Support for multicast communication in Networks-on-Chip (NoCs) has achieved substantial throughput improvements and power savings. In this paper, we explore support for reduction or many-to-one communication operations. As a case study, we focus on acknowledgement messages (ACK) that must be collected in a directory protocol before a cache line may be upgraded to or installed in the modified state. This paper makes two primary contributions: an efficient framework to support the reduction of ACK packets and a novel* Balanced, Adaptive Multicast *(*BAM*) routing algorithm. The proposed message combination framework complements several multicast algorithms. By combining ACK packets during transmission, this framework not only reduces packet latency by 14.1% for low-to-medium network loads, but also improves the network saturation throughput by 9.6% with little overhead. The balanced buffer resource configuration of BAM improves the saturation throughput by an additional 13.8%. For the PARSEC benchmarks, our design offers an average speedup of 12.7% and a maximal speedup of 16.8%.*[1]

## 1 Introduction

Efficient and scalable on-chip communication will be required to realize the performance potential of many-core architectures. To harness this performance, it is imperative that NoCs be designed to efficiently handle a variety of communication primitives. Collective communication lies on the critical path for many applications; the criticality of such communication is evident in the dedicated collective and barrier networks employed in several supercomputers, such as NYU Ultracomputer [15], CM-5 [24], Cray T3D [6], Blue Gene/L [2] and TH-1A [43]. Likewise, many-core architectures will benefit from hardware support for collective communications but may not be able to afford separate, dedicated networks due to rigid power and area budgets [34]; this paper explores integrating collective communication support directly into the existing NoC.

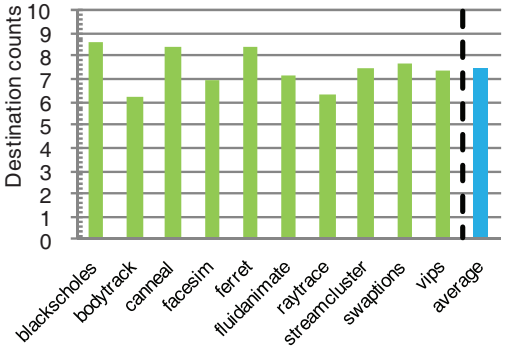



**Figure 1. Average destinations per multicast for the PARSEC benchmarks.**

Various parallel applications and programming paradigms require collective communication such as broadcast, multicast and reduction. For example, a directory-based coherence protocol relies heavily on multicasts to invalidate shared data spread across multiple caches [19] and Token Coherence uses multicasts to collect tokens [29]. Reductions and multicasts are used for barrier synchronization [35, 42]. These collective communications can have a significant effect on many-core system performance. Without any special hardware mechanisms, even if 1% of injected packets are multicast, there is a sharp drop in saturation throughput [12]. Recent work proposes efficient multicast routing support to improve NoC performance [1, 12, 20, 26, 37, 38, 39, 41].

Often a multicast will trigger an operation, such as invalidating a cache line [19] or counting available tokens [29]. To notify the source of the completion of these operations, the multicast destination nodes send out responses. The resulting many-to-one communication operation is called a reduction [9]. Figure 1 shows that a cache line invalidation message triggers on average 7.44 acknowledgement mes-

[1]This research was carried out while Sheng Ma was a visiting international student at the University of Toronto supported by a CSC scholarship.

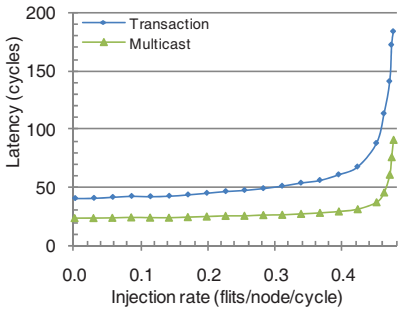



**Figure 2. Latency of multicast-reduction transaction. (Multicast: latency of last arriving multicast replica; Transaction: latency of last arriving ACK; the network is routed by BAM+NonCom as described in Section 5.)**

sages for the PARSEC benchmarks [3] in a 16-core system[2]. Prior NoC multicast proposals [1, 12, 20, 26, 37, 38, 39, 41] implicitly assume several unicast packets will deliver these messages to a single destination; this can lead to redundant network traversals and create transient hotspots in the network. To provide high performance, scalable NoCs should handle traffic in an intelligent fashion by eliminating these redundant messages.

Furthermore, the multicast-reduction transaction cannot complete until all responses are received [4, 10, 19, 22, 25, 29]. As a result, the transmission of reduction messages lies on the critical path of a multicast-reduction transaction. These multicast-reduction operations are often associated with stores; for out-of-order cores, stores do not lie on the critical path. However, these stores can delay subsequent loads to hotly contended cache lines. For CMPs that employ simple, in-order cores, stores will lie on the critical path and can significantly impact performance. Figure 2 shows the completion latency of multicast-reduction transactions in a $4\times4$ mesh running uniform random traffic. The transmission of reduction packets accounts for $\sim$40% of the total transaction latency. We propose a novel packet reduction mechanism to improve performance of the full multicast-reduction transaction.

A noteworthy property of coherence-based reduction messages is that they carry similar information in a simple format. For example, invalidation acknowledgement messages only carry the acknowledgement (ACK) for each node and token count replies merely carry the count of available tokens at each node. Therefore, these response messages can be combined without loss of information. Combining these messages eliminates redundant network traversals and optimizes performance. We propose an efficient message combination framework with little overhead. To simplify discussion, we focus on invalidation ACK packets in a directory-based coherence protocol. Our design can be easily extended to other types of reductions such as those used in Token Coherence [29].

[2]See Section 5 for detailed experimental configuration.

Our proposed message combination framework complements several multicast routing algorithms. Sending a multicast packet constructs a logical tree in the network. The framework steers each ACK packet to traverse the same logical tree back to the root (source) of the multicast. In each router, a small message combination table (MCT) records total and received ACK counts for active multicast transactions. When an ACK packet arrives at the router, the MCT is checked. If the router has not received all expected ACKs, the table is updated and the incoming ACK is discarded. If the router has received all expected ACKs, the incoming ACK packet is updated and forwarded to the next node in the logical tree. Dropping in-flight ACK packets reduces network load and power consumption.

Our goal is to improve overall network performance in the presence of both unicast and multicast-reduction traffic. The recently proposed Recursive Partitioning Multicast (RPM) routing algorithm utilizes two virtual networks (VNs) to avoid deadlock for multicasts [39]. However, the division of these two VNs results in unbalanced buffer resources between vertical and horizontal dimensions, which negatively affects performance. Therefore, we propose a novel multicast routing algorithm, *Balanced, Adaptive Multicast* (BAM), which does not need two VNs to avoid multicast deadlock. BAM balances the buffer resources between different dimensions, and achieves efficient bandwidth utilization by computing an output port based on all the multicast destination positions.

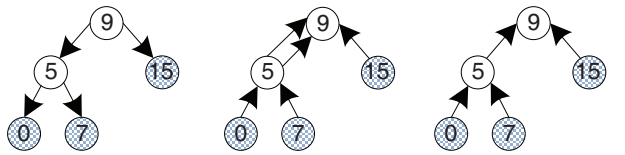
To summarize, our main contributions are the following:

- An efficient message combination framework that reduces latency by 14.1% and energy-delay product (EDP) by 20-40% for low-to-medium network loads and improves saturation throughput by 9.6%.
- A novel multicast routing algorithm which balances of buffer resources across different dimensions and improves network throughput by an additional 13.8%.

## 2 Message Combination Framework

In this section, we describe the proposed message combination framework. We use a multicast-reduction example to illustrate the framework. One multicast packet with destinations 0, 7 and 15 is injected by node 9. A logical multicast tree [13] is built as shown in Figure 3(a); grey nodes indicate destinations while white nodes are branches that are only traversed by the packet. Each multicast destination responds with an ACK message to the root node 9. Without combination, the ACKs are transmitted as unicast packets back to node 9 (Figure 3(b))[3]. These ACK packets travel some common channels; merging them can reduce the network load. Figure 3(c) shows the logical ACK tree with message combination, which is the same as the logical

[3]We assume the ACKs sent out by nodes 0 and 7 both traverse node 5.




(a) Logical Multicast tree. (b) Logical ACK tree. (w/o combination) (c) Logical ACK tree. (with combination)

**Figure 3. Message combination framework overview.**

multicast tree except with the opposite transmission direction. In this example, routers 5 and 9 serve as fork routers which are responsible for gathering and forwarding ACKs. Next, we address two important issues associated with our message combination framework: ensuring that ACK packets traverse the same logical tree as the multicast packet and ensuring that the fork routers are aware of the expected total and currently received ACKs.

In an $n \times n$ network, a $\log(n \times n)$-bit field in the multicast header is reserved to identify the router where the last replication happened (*pre_rep_router*). This field is initially set to be the source node and is updated when a multicast packet replicates during transmission. A 3-bit *ID* field is used to differentiate multicast packets injected by the same source node. This field increments when a new multicast packet is injected. The *src* field encodes the source node.

A small message combination table (MCT) is added to each router. A multicast allocates an MCT entry upon replication. This entry records the identity of the router where the last multicast replication occurred and the total expected ACK count. The transmission of a multicast packet establishes a logical tree. Each branch in the logical tree has an MCT entry pointing to the previous fork router.

Each multicast destination responds with an ACK packet. A $\log(n \times n)$-bit field (*cur_dest*) in the ACK header serves to identify the intermediate destination. Its value is set to the *pre_rep_router* field in the triggering multicast packet. Each ACK packet has two fields named *multicast_src* and *multicast_ID* which correspond to *src* and *ID* of the triggering multicast packet, respectively. An additional $\log(n \times n)$-bit field (*ACK_count*) is used to record the carried ACK response count of the combined packet.

When an ACK packet arrives at its current destination, it accesses the MCT. If the router has not yet received all expected ACKs, the incoming packet is discarded and the entry's received ACK count is incremented. If the router has received all expected ACKs, the incoming ACK packet updates its *cur_dest* field to be the next replication router. It will be routed to the fork router at the next level; thus, ACK packets traverse the same logical tree as multicast in the opposite direction.

## 2.1 MCT Format

Figure 4 illustrates the format of an MCT entry. The *V* field is the valid bit for the entry. The *src*, *ID* and

| V | src | ID | pre_rep _router | incoming _port | expected _count | cur_ACK _count |
|---|---|---|---|---|---|---|
| 1 bit | 4 bits | 3 bits | 4 bits | 3 bits | 4 bits | 4 bits |

**Figure 4. MCT entry format. (Port encoding: E: 0, W: 1, S: 2, N: 3, local: 4. Assuming 16 nodes.)**

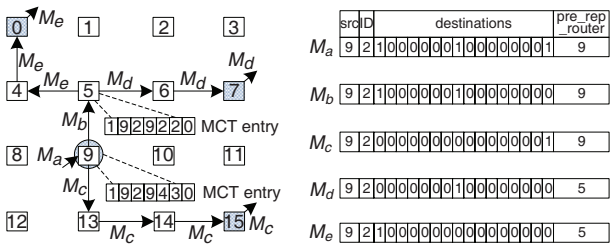
*pre_rep_router* fields are the same as the corresponding fields in the multicast packet initializing this entry. The MCT is a content-addressable memory (CAM); the *src* and *ID* fields work together as the tag. The *incoming_port* field records the incoming port of the multicast packet. The *expected_count* field indicates the total expected ACK count, which is equal to the number of destinations at this branch of the multicast tree. The value of *cur_ACK_count* field tracks the current received ACK count. As we will show later, recording the total expected ACK count instead of simply counting the number of direct successors is needed for handling full MCTs.

## 2.2 Message Combination Example

Figure 5(a) gives a multicast example within a 4×4 mesh network which is the same as Figure 3(a). Figure 5(b) shows the multicast header values. Although our framework is independent of the multicast packet format, we assume bit string encoding [5] for the destination addresses in the *destinations* field of the header for clarity. $M_a$ is the injected packet; its *destinations* field contains the three destinations. $M_a$ replicates into two packets, $M_b$ and $M_c$ at router 9. An MCT entry is created. The *src*, *ID* and *pre_rep_router* fields of this entry are fetched from $M_a$. The *incoming_port* field is set to 4 to indicate that $M_a$ comes from the local input port. The *expected_count* field is set to the total destination count of $M_a$: 3. The *cur_ACK_count* field is set to 0. At router 5, $M_b$ replicates into two packets: $M_d$ and $M_e$. An MCT entry is created with an *expected_count* of 2. The *pre_rep_router* fields of both $M_d$ and $M_e$ are updated to 5 since the last replication occurred at router 5.

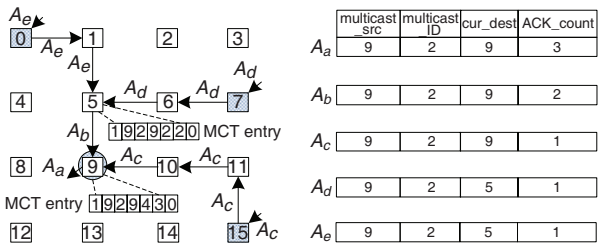
After a destination node receives a multicast, it responds with an ACK packet. Figure 6(a) shows the transmission of ACK packets corresponding to the multicast shown in Figure 5(a). Figure 6(b) gives the ACK header values. $A_c$, $A_d$ and $A_e$ packets are triggered by the $M_c$, $M_d$ and $M_e$ multicast packets, respectively. The *multicast_src*, *multicast_ID* and *cur_dest* fields of the ACK packets are equal to the *src*, *ID* and *pre_rep_router* fields of the triggering multicast packet, respectively. The *ACK_count* fields of these three ACK packets are set to 1 since they all carry an ACK response from only one node.

The *cur_dest* field defines the current destination of ACK packet. As shown in Figure 6(a), $A_e$ can be routed along a different path than $M_e$ to reach its intermediate destination at router 5. ACK packets only need to follow the same *logical* tree as the multicast packet giving our design significant




(a) Transmission procedure. (b) Header value.

**Figure 5. A multicast packet transmission example.**




(a) Transmission procedure. (b) Header value.

**Figure 6. ACK packets transmission example.**

flexibility. A similar scenario can be seen for $A_c$. $A_d$ and $A_e$ both set their *cur_dest* fields as router 5; at router 5, they will be merged. Analysis shows that the possibility of multiple simultaneous MCT accesses is quite low ($\leq 0.1\%$) as ACK packets will experience different congestion. A small arbiter is used to serialize concurrent accesses. Assuming $A_d$ arrives earlier than $A_e$; its *multicast_src* and *multicast_ID* are used together as the tag to search the MCT. The sum of *cur_ACK_count* field of the matched entry and the carried *ACK_count* of $A_d$ is 1, which is smaller than the *expected_count* of 2 in that entry. Therefore, $A_d$ is discarded and the *cur_ACK_count* field is incremented by 1.

When $A_e$ arrives at router 5, it accesses the MCT. Since router 5 has received all expected ACKs, $A_e$ will remain in the network. Its *cur_dest* field is updated to the *pre_rep_router* field of the matched entry and its *ACK_count* field is updated to 2 since it now carries the ACK responses of nodes 0 and 7 (see $A_b$). $A_b$ uses the *incoming_port* field of the matched entry as the output port. The combined ACK packet is required to use the same multicast path for one hop to avoid an additional routing computation stage which would add an additional cycle of latency. Now that router 5 has received all expected ACKs, the corresponding MCT entry is freed. Finally, node 9 receives $A_b$ and $A_c$ and combines them into $A_a$. The multicast-reduction transaction is complete.

### 2.3 Insufficient MCT Entries

So far, we have assumed that there is always an available MCT entry when a multicast packet replicates. However, since the table size is finite, we must be able to handle a full MCT. If there are no free MCT entries, the replicated multicast packet will not update its *pre_rep_router* field. In the previous example, if there is no available MCT entry at router 5 when $M_b$ replicates, $M_d$ and $M_e$ will keep their *pre_rep_router* field as 9. When $A_d$ and $A_e$ are injected, their *cur_dest* fields are set to 9; they will combine in router 9 instead of router 5. Both $A_d$ and $A_e$ must travel to router 9. In this case, router 9 will receive two ACK packets for the north-bound replication branch; this is why we record the expected total count of ACKs in MCT instead of recording the number of direct successors in the logic multicast tree. In our design, insufficient MCT entries may affect performance, but do not pose any correctness issues. We evaluate the effect of the MCT size in Section 5.3.

## 3 Balanced, Adaptive Multicast Routing

In this section, we describe our *Balanced, Adaptive Multicast* (BAM) routing algorithm. To achieve efficient bandwidth utilization, a multicast routing algorithm must compute the output port based on all destination positions in a network [39]. A simple and efficient routing algorithm, RPM, was recently proposed to deliver high performance [39]. As shown in Figure 7, RPM partitions the network into at most 8 parts based on the current position, and applies several priority rules to avoid redundant replication for destinations located in different parts; the goal is to deliver a multicast packet along a common path as far as possible, then replicate and forward each copy on a different channel bound for a unique destination subset.

We observe that although RPM provides efficient bandwidth utilization, it suffers from unbalanced buffer resources between different dimensions which negatively affects network performance. To avoid deadlock for multicast routing, RPM divides the physical network into two virtual networks (VNs): VN0 is for upward packets and VN1 is for downward ones. The horizontal VC buffers must be split into two disjoint subsets for the two VNs, while the vertical ones can be exclusively used by one VN [9, 39]. When a packet is routed in each VN, there are $2x$ more available vertical buffers than horizontal ones. This unbalanced buffer configuration negatively affect both unicast and multicast routing, since the more limited horizontal VCs become a performance bottleneck. Configuring different VC counts for different dimensions may mitigate this effect. However, it requires different control logic for each input port as the size of the arbiters in VC and switch allocators is related to the VC count; a heterogeneous router requires extra design effort [31]. Also, the critical path may increase since it is determined by the largest arbiter. Therefore, we assume a homogeneous NoC router architecture in this paper.

Based on these observations, we propose a novel adaptive multicast routing algorithm: *Balanced, Adaptive Multicast* (BAM). The deadlock freedom of BAM is achieved by utilizing Duato's unicast deadlock-avoidance theory [8] for multicast packets, rather than leveraging multiple VNs. The

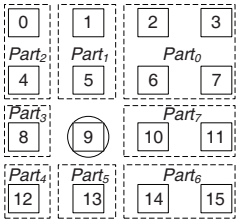



**Figure 7. Network partitioning. (The current position of a packet is Router 9.)**

multicast packets in NoCs are generally short as they carry only control packets for the coherence protocol; these are most likely single-flit packets [12], making the routing of each multicast branch independent. Thus, Duato's unicast theory can be applied to multicasts by regarding the routing of each multicast branch as an independent unicast. In Duato's theory, VCs are classified into escape and adaptive VCs. When a packet resides in an adaptive VC, it can be forwarded to any permissible output port. This property enables BAM to select the best output port based on all destination positions. An additional advantage is that this design is compatible with an adaptive unicast routing algorithm.

Figure 7 shows the partitioning of destinations into 8 parts. For $Part_1$, $Part_3$, $Part_5$ or $Part_7$, there is only one admissible output port. For $Part_0$, $Part_2$, $Part_4$ or $Part_6$, there are two alternative output ports. If a multicast packet has some destinations located in $Part_1$, $Part_3$, $Part_5$ and $Part_7$, the corresponding N, W, S and E output port must be used; these ports are called *obligatory* output ports for this multicast packet. To achieve efficient bandwidth utilization, we design a heuristic output port selection scheme for destinations located in $Part_0$, $Part_2$, $Part_4$ and $Part_6$: (i) If only one of the two alternative output ports is an *obligatory* output port for the multicast packet, the router will use this output port to reach the destination; (ii) If the two alternative output ports both are *obligatory* or not *obligatory* output ports, the router will adaptively select the one with less congestion. This scheme maximally reuses the *obligatory* output ports to efficiently utilize bandwidth.

Figure 8 shows the output port calculation logic for BAM. The one bit $P_i$ indicates whether there is a destination in $Part_i$. Take $Part_0$ as an example: $N_{p_0}$ and $E_{p_0}$ indicate that the router uses the north or east port to reach the destinations located in $Part_0$. $N_{ne}$ and $E_{ne}$ signals indicate whether the north or east output has less relative congestion in the northeast (ne) quadrant. These signals are provided by the routing computation module.

$Escape_n$, $Escape_w$, $Escape_s$ and $Escape_e$ indicate whether the multicast packet can use the escape VC for the N, W, S and E output ports, respectively. If a multicast packet uses the north output port to reach nodes in $Part_0$ or $Part_2$, it is not allowed to use the north escape VC since this packet will make a turn forbidden by DOR. A similar rule is applied to the south escape VC. The east and west

$$
\begin{array}{ll}
N_{p_0} = P_1 \cdot \overline{P_7} + P_1 \cdot P_7 \cdot N_{ne} + \overline{P_1} \cdot \overline{P_7} \cdot N_{ne} & N = P_1 + N_{p_0} + N_{p_2} \\
E_{p_0} = \overline{P_1} \cdot P_7 + P_1 \cdot P_7 \cdot E_{ne} + \overline{P_1} \cdot \overline{P_7} \cdot E_{ne} & W = P_3 + W_{p_2} + W_{p_4} \\
N_{p_2} = P_1 \cdot \overline{P_3} + P_1 \cdot P_3 \cdot N_{nw} + \overline{P_1} \cdot \overline{P_3} \cdot N_{nw} & E = P_7 + E_{p_0} + E_{p_6} \\
W_{p_2} = \overline{P_1} \cdot P_3 + P_1 \cdot P_3 \cdot W_{nw} + \overline{P_1} \cdot \overline{P_7} \cdot W_{nw} & S = P_5 + S_{p_4} + S_{p_6} \\
S_{p_4} = P_5 \cdot \overline{P_3} + P_5 \cdot P_3 \cdot S_{sw} + \overline{P_5} \cdot \overline{P_3} \cdot S_{sw} & Escape_n = \overline{N_{p_0}} \cdot \overline{N_{p_2}} \cdot P_1 \\
W_{p_4} = \overline{P_5} \cdot P_3 + P_5 \cdot P_3 \cdot W_{sw} + \overline{P_5} \cdot \overline{P_3} \cdot W_{sw} & Escape_w = W \\
S_{p_6} = P_5 \cdot \overline{P_7} + P_5 \cdot P_7 \cdot S_{se} + \overline{P_5} \cdot \overline{P_7} \cdot S_{se} & Escape_s = \overline{S_{p_4}} \cdot \overline{S_{p_6}} \cdot P_5 \\
W_{p_6} = \overline{P_5} \cdot P_7 + P_5 \cdot P_7 \cdot E_{se} + \overline{P_5} \cdot \overline{P_7} \cdot E_{se} & Escape_e = E
\end{array}
$$

**Figure 8. BAM routing computation logic.**

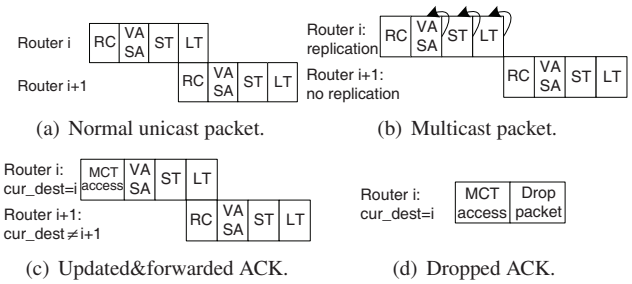
escape VCs are always available for routing. If a multicast packet resides in an escape VC, it will replicate according to DOR, similar to VCTM [12]. Once a multicast packet enters an escape VC, it can be forwarded to the destinations using only escape VCs; there is no deadlock among escape VCs. Any multicast packet residing in an adaptive VC has an opportunity to use an escape VC. This design is deadlock free [8]. Compared with RPM, BAM does not need to partition the physical network into two virtual networks to avoid multicast deadlock; it achieves balanced buffer resources across vertical and horizontal dimensions. Moreover, BAM achieves efficient bandwidth utilization as well.

# 4 Router Pipeline and Microarchitecture

Our baseline is a speculative VC router [7, 11, 36]. Look-ahead signals transmit the unicast routing information one cycle ahead of the flit traversal to overlap the routing computation (RC) and link traversal (LT) stages [17, 23]. We use a technique to pre-select the preferred output for adaptive routing; the optimal output port for each quadrant is selected one cycle ahead based on network status [16, 21, 27]. The pipeline for unicast packets is two cycles plus one cycle for LT, as shown in Figure 9(a).

Including multicast routing information in the look-ahead signals requires too many bits; therefore, we assume a 3-cycle router pipeline for multicasts, as shown in Figure 9(b). A multicast packet replicates inside the router if multiple output ports are needed to reach the multicast destinations. We use asynchronous replication to eliminate lock-step traversal among several branches; the multicast packet is handled as multiple independent unicast packets in the virtual channel allocation (VA) and switch allocation (SA) stage, except that a flit is not removed from the input VC until all requested output ports are satisfied [20, 39].

ACK packets are handled differently from other unicast packets. When an ACK packet arrives, its *cur_dest* field is checked. If this field does not match the current router, the ACK packet is handled like a normal unicast packet (Figure 9(a)). If they match, the ACK packet accesses the MCT instead of performing the routing computation. The MCT access is overlapped with the RC stage. As we show in Section 5.3, this operation can fit within a single pipeline stage; it does not add additional latency to the critical path. Fig-




(a) Normal unicast packet. (b) Multicast packet.

(c) Updated&forwarded ACK. (d) Dropped ACK.

**Figure 9. Router pipeline.**

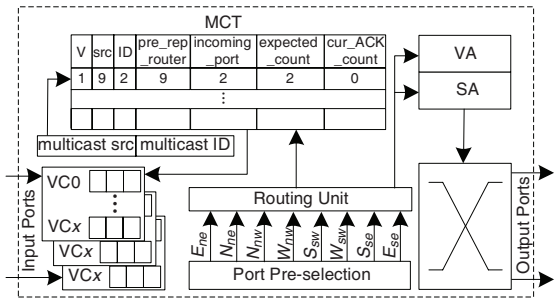



**Figure 10. Router microarchitecture.**

ures 9(c) and 9(d) illustrate the ACK packet pipeline.

Figure 10 illustrates the proposed router microarchitecture. If a multicast packet needs multiple output ports after the routing computation, an entry is allocated in the MCT. This operation is overlapped with the VA/SA operations and does not add delay to the critical path. The *Port Pre-selection* module provides eight signals indicating the optimal output port for each quadrant [16, 21, 27]. These signals are used by both unicasts and multicasts to avoid network congestion.

# 5 Evaluation

We evaluate our message combination framework with RPM and BAM using synthetic traffic and real application workloads. We modify the cycle-accurate Booksim simulator [7] to model the router pipelines and microarchitecture described in Section 4. For synthetic traffic, we configure two VNs to avoid protocol-level deadlock [7]: one for multicasts and one for ACKs. RPM further divides the multicast VN into two sub-VNs: one for upward packets and one for downward ones. BAM does not need to sub-divide the multicast VN. Normal unicast packets can use any VN. However, once injected into the network, a packet's VN is fixed and cannot change during transmission. Unicast packets are routed by an adaptive routing algorithm. For BAM's multicast, BAM's ACK and RPM's ACK VNs, the algorithm is designed based on Duato's theory [8] and uses one VC as the escape VC. The two sub-VNs of RPM's multicast VN enable adaptive routing without requiring escape VCs. We use a local selection strategy for adaptive routing: when there are two available output ports, the selection strategy chooses the one with more free buffers.

**Table 1. Simulation configuration and variations.**

| Characteristic | Baseline | Variations |
|---|---|---|
| Topology (mesh) | 4×4 | 8×8, 16×16 |
| VC configuration | 4 flits/VC, 8 VCs/port | 4 & 6 VCs/port |
| Packet length (flits) | Normal: 1 & 5 (bi-modal) ACK: 1; multicast: 1 | - |
| Unicast traffic pattern | uniform random, transpose, bit rotation, hot spot | - |
| Multicast ratio | 10% | 5%, 15%, 20% |
| Multicast dest. count | 2 - 10 (uniformly dist.) | 2 - 4, 4 - 14, 10 - 14, 15 |
| ACK resp. cycles | 1 - 4 (uniformly dist.) | - |
| MCT entries | 64 | 0, 1, 4, 16 |
| Warmup & total | 10000 & 100000 cycles | - |

**Table 2. Full system simulation configuration.**

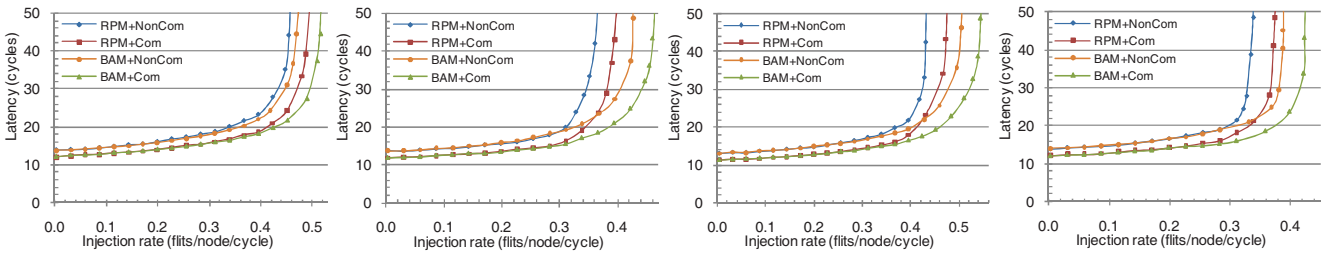
| | |
|---|---|
| # of cores | 16 |
| L1 cache (D & I) | private, 4-way, 32KB each |
| L2 cache | private, 8-way, 512KB each |
| Cache coherence | MOESI distributed directory |
| Topology | 4×4 2D-Mesh |

Multicasts and ACKs are single-flit packets, while the normal unicasts are bimodally distributed, consisting of 5-flit packets (50%) and 1-flit packets (50%). We use several synthetic unicast traffic patterns [7], including uniform random, transpose, bit rotation and hot spot, to stress the network for detailed insights. We control the percentage of multicast packets relative to whole injected packets. For multicasts, the destination counts and positions are uniformly distributed. A cache's ACK packet response delay is uniformly distributed between 1 and 4 cycles. We assume a 64-entry MCT; in Section 5.3, we explore the impact of MCT size. Table 1 summarizes the baseline configuration and variations used in the sensitivity studies.

To measure full-system performance, we leverage two existing simulation frameworks: FeS2 [33] for x86 simulation and BookSim for NoC simulation. FeS2 is a timing-first, multiprocessor x86 simulator, implemented as a module for Virtutech Simics [28]. We run the PARSEC benchmarks [3] with 16 threads on a 16-core CMP, consisting of Intel Pentium 4-like CPU. We assume cores optimized for clock frequency; they are clocked 5× faster than the network. We use a distributed, directory-based MOESI coherence protocol that needs 4 VNs for protocol-level deadlock freedom. The cache line invalidation packets (multicasts) are routed in VN1, while the acknowledge packets are routed in VN2. The VCs/VN, VC depth and MCT size are the same as the baseline (Table 1). Cache lines are 64 bytes wide and the network flit width is 16 bytes. All benchmarks use the *simsmall* input sets to reduce simulation time. The total runtime is used as the metric for full-system performance. Table 2 gives the system configuration.

## 5.1 Performance

We evaluate four scenarios: RPM without message combination (RPM+NonCom), RPM with message combination (RPM+Com), BAM without message combina-




(a) Uniform random. (b) Transpose. (c) Bit rotation. (d) Hot spot.

**Figure 11. Overall network performance. (10% multicast, average 6 destinations.)**

tion (BAM+NonCom) and BAM with message combination (BAM+Com).

**Overall Network Performance.** Figure 11 illustrates overall network performance. Both RPM+Com and BAM+Com see performance improvements compared to RPM+NonCom and BAM+NonCom respectively; not only are the network latencies reduced, but the saturation throughputs[4] are improved. Detailed analysis reveals that the combination framework reduces the average channel traversal count of ACK packets from 4.7 to 2.5, reducing approximately 45% of the network operations for ACKs.

The network latency for RPM+Com is reduced by 10%-20% under low-to-medium injection rates[5] compared to RPM+NonCom (average 14.1%); larger improvements are seen at high injection rates. Similar latency reductions are seen for BAM+Com vs. BAM+NonCom. Packet dropping shortens the average hop distance bringing latency reductions at both low and high network loads. The mitigation of ejection-side congestion is also beneficial to latency reduction. Saturation throughput improvements resulting from the framework (RPM+Com vs. RPM+NonCom) range from 8.5% to 11.2% (average 9.6%). BAM+Com sees similar throughput improvements over BAM+NonCom. Discarding in-flight ACKs reduces the network load, which is helpful to improve the saturation throughput.

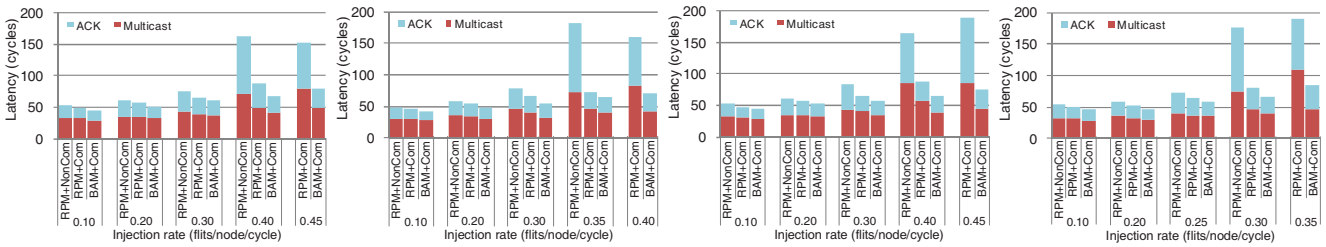
Across the four traffic patterns, both BAM+NonCom and BAM+Com improve the saturation throughput over RPM+NonCom and RPM+Com, respectively. For transpose, bit rotation and hot spot patterns, although BAM+NonCom has larger latencies than RPM+Com under low loads, its saturation throughputs are higher. The balanced buffer configuration between vertical and horizontal dimensions helps BAM to improve the saturation throughput. BAM+Com improves the saturation throughput by 14.2%, 27.6%, 26.4% and 25.1% (average 23.4%) over RPM+NonCom for the four traffic patterns. Both the ACK packet dropping and balanced buffer configuration contribute to this performance improvement. As shown in Figure 11, the trend between BAM+NonCom and RPM+NonCom is similar to the trend between BAM+Com and RPM+Com; thus, we omit BAM+NonCom in the following sections for brevity.

**Multicast Transaction Performance.** To clearly understand the effects of message combination on multicast transactions, we measure the multicast-reduction transaction latency. Figure 12 shows the results for five injection rates. The injection rate for the last group of bars exceeds the saturation point of RPM+NonCom. Under all injection rates, RPM+Com and BAM+Com have lower transaction latencies than RPM+NonCom; dropping ACK packets reduces network congestion and accelerates multicast-reduction transactions. ACK packet acceleration contributes more to the latency reduction than multicasts. A multicast needs to send out multiple replicas; releasing its current VC depends on the worst congestion each replica may face. Thus, the multicast is not as sensitive as the ACK to the network load reduction. For example, with low-to-medium injection rates ($\leq 0.30$) under uniform random traffic (Figure 12(a)), the average multicast delay is reduced by 9.5% for BAM+Com versus RPM+NonCom. Yet, ACK packet delay is reduced by 17.6%. These two factors result in an average 13.4% transaction latency reduction. The transaction acceleration increases with higher network load. Merging ACKs reduces the number of packets the source needs to wait for to finish a transaction. Waiting for only one ACK instead of multiple ACKs can improve performance since multiple packets may encounter significantly more congestion than a single packet, especially under high network load. For uniform random traffic with a high injection rate (0.40), RPM+Com and BAM+Com reduce the transaction latency by 46.2% and 57.6% compared to RPM+NonCom, respectively. The message combination framework accelerates the total transaction by almost a factor of 2. As injection rate increases, BAM+Com outperforms RPM+Com by a significant margin.

**Real Application Performance.** Figure 13 shows the speedups over RPM+NonCom for the PARSEC benchmarks. Although the message combination framework mainly optimizes the performance for one of the four VNs (the ACK VN) used in full-system simulation, collective

[4]The saturation point is measured as the injection rate at which the average latency is 3× the zero load latency.

[5]The low-to-medium injection rate is the injection rate at which the average latency is less than 2× the zero load latency.




(a) Uniform random. (b) Transpose. (c) Bit rotation. (d) Hot spot.

**Figure 12. Multicast-reduction transaction latency. (10% multicast, average 6 destinations.)**

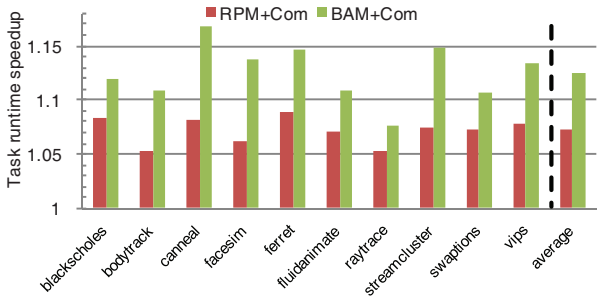



**Figure 13. System speedups against RPM+NonCom for the PARSEC benchmarks.**

communication often lies on the critical path for applications. Also, dropping ACK packets in one VN reduces switch allocation contention and improves the performance of other VNs. These factors result in RPM+Com achieving speedups over RPM+NonCom ranging from 5.3% to 8.3% for all applications. The efficiency of message combination depends on the multicast destination count. The multicast destination counts of *bodytrack*, *facesim* and *raytrace* are the lowest; the speedups of RPM+Com for these 3 applications are lower than the remaining 7 applications.

The balanced buffer configuration utilized in BAM+Com supports higher saturation throughput than the unbalanced one. BAM+Com improves the performance of applications which have high network loads and significant bursty communication. For *blackscholes*, *fluidanimate*, *raytrace* and *swaptions*, the additional performance gain due to the balanced buffer configuration (BAM+Com vs. RPM+Com) ranges from 2.3% to 3.7%. The network loads of these applications are low and do not stress the network. However, BAM+Com achieves additional speedups ranging from 5.5% to 8.6% over RPM+Com for *bodytrack*, *canneal*, *facesim*, *ferret*, *streamcluster* and *vips*. These applications have more bursty communication and higher injection rates. For all ten applications, BAM+Com achieves an average speedup of 12.7% over RPM+NonCom. The maximal speedup is 16.8% for *canneal*.

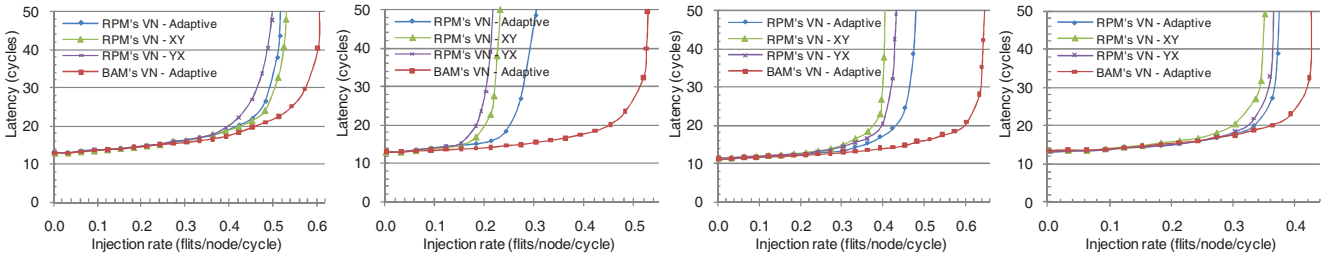
## 5.2 Comparing BAM's and RPM's Multicast VN Configuration

In this section, we delve into the effect of the unbalanced buffer configuration used in the multicast VN of RPM on both unicast and multicast packet routing. Since the ACK VNs of RPM and BAM are the same, we assume the network only has one VN: the multicast VN. Four VCs are configured in this VN, and RPM further divides this VN into two sub-VNs: the horizontal VC count of each sub-VN is two, while the vertical count is four.

**Unicast Performance.** Figure 14 compares the performance of BAM's and RPM's multicast VN configuration using only unicast packets. To extensively understand the effect of the unbalanced buffer configuration, we evaluate the performance of XY, YX and a locally adaptive routing algorithm (Adaptive) in RPM's multicast VN. XY efficiently distributes uniform random traffic and achieves the highest performance for this pattern. For the other three patterns, Adaptive has the highest performance; adaptively choosing the output port mitigates the negative effect of unbalanced buffer resources. Therefore, this work uses locally adaptive routing for unicast packets.

Although Adaptive has better performance than XY and YX, its performance is still limited by the unbalanced buffer resources used in each of RPM's multicast sub-VNs; the horizontal dimension has half the buffer resources of the vertical one. However, in BAM's VN, the number of buffers of different dimensions are equal. Adaptive routing in BAM's VN shows substantial performance improvement over Adaptive routing in RPM's VN. Transpose has the largest performance gain with a 73.2% saturation point improvement. Across these four traffic patterns, *BAM's VN-Adaptive* achieves an average saturation throughput improvement of 35.3% over *RPM's VN-Adaptive*.

**Multicast Performance.** Figure 15 shows the performance using 100% multicast packets. The *Adaptive* curve shows the performance of the adaptive multicast routing algorithm without our heuristic replication scheme; multicast replicas adaptively choose the output ports without considering the reuse of *obligatory* ports. This negatively affects bandwidth utilization. BAM has 8.7% higher saturation throughput than Adaptive. BAM achieves 47.1% higher saturation throughput over RPM. The effect of the unbalanced buffer resources is greater on multicasts than unicasts. A multicast packet is removed from its current VC only after all its replicas are sent out; the horizontal VC bottleneck affects




(a) Uniform random. (b) Transpose. (c) Bit rotation. (d) Hot spot.

**Figure 14. Unicast traffic performance for RPM's and BAM's multicast VN.**

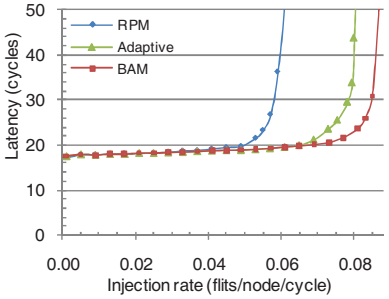
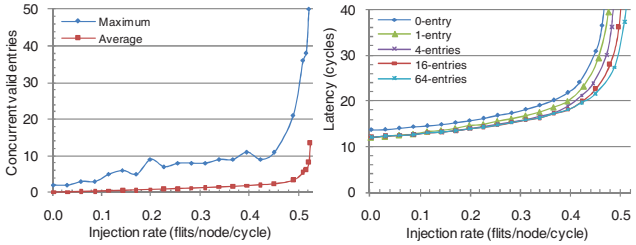



**Figure 15. Performance of 100% multicast traffic. (average 6 destinations.)**




(a) Concurrent valid entries. (b) Performance.

**Figure 16. Evaluation of MCT size. (10% multicast, average 6 destinations.)**

multicast performance more strongly than unicast performance. Average switch traversal counts are 8.6, 8.8 and 8.4 for RPM, Adaptive and BAM, respectively, which further demonstrates that our applied heuristic replication scheme achieves efficient bandwidth utilization.

## 5.3 MCT Size

As described in Section 2.3, the MCT size affects network performance but not correctness. Too few MCT entries will hamper ACK packet combination and force the ACK packets to travel more hops than with combination. To determine the appropriate size, we simulate an infinite MCT using the baseline configuration (Table 1) and uniform random traffic. Multicast packets are routed using BAM. Figure 16(a) presents the maximum and average concurrently valid MCT entries. For low-to-medium injection rates ($<$ 0.39), the maximum number of concurrently valid entries is less than 10 and the average number is less than 1.5. Even when the network is at saturation (0.52 injection rate), the maximum number of concurrently valid entries is 49 and the average is 10.15. These experimental results indicate that a small MCT can provide good performance.

Figure 16(b) shows the performance for different table sizes. The *0-entry* curve performs no ACK combination; all ACK packets are injected into the network with their destination set to the source node of the triggering multicast packet. Even with only one entry per router, ACK combination reduces the average network latency by 10%. More entries reduce the latency further, especially for high injection rates. Saturation throughput improvements range from 3.3% to 12.1% for 1 to 64 entries.

**Table 3. MCT overhead.**

| Entries | Area ($mm^2$) | Energy ($nJ$) | Time ($ns$) | Bytes |
|---|---|---|---|---|
| 16 | 0.0011 | 0.0008 | 0.138 | 48 |
| 32 | 0.0017 | 0.0013 | 0.146 | 96 |
| 64 | 0.0031 | 0.0026 | 0.153 | 192 |

Cacti [32] is used to calculate the power consumption, area and access latency for the MCT in a 32nm technology process. Table 3 shows the results. Assuming a 1 GHz clock frequency, a 64-entry table can be accessed in one cycle. This size provides good performance for nearly all injection rates for various traffic patterns. In the full system evaluation, we observe that the maximal concurrently valid entries for the PARSEC benchmarks are less than 25 entries. For area-constrained designs, fewer entries still provide latency and throughput improvements.

## 5.4 Sensitivity to Network Design

To further understand the scalability and impact of our design, we vary the VC count, multicast ratio, destination count per multicast packet and network size. Figure 17 presents the average performance improvement across the four synthetic unicast traffic patterns. In each group of bars, the first two bars show the saturation throughput improvement, and the second two bars show the latency reduction under low-to-medium loads.

**VC Count.** Figure 17(a) shows the performance improvement with 8, 6, and 4 VCs per physical channel. One interesting trend is observed: For smaller VC counts, the gain due to combination framework increases (RPM+Com vs. RPM+NonCom), while the improvement due to balanced buffer resources declines (BAM+Com vs. RPM+Com).

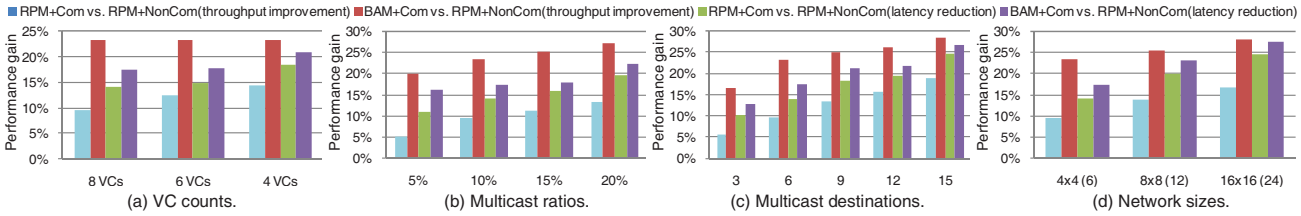



**Figure 17. Performance gains of RPM+Com and BAM+Com over RPM+NonCom for sensitivity study.**

The reasons for this trend are two-fold. First, fewer VCs per port makes the VCs a more precious resource; dropping ACK packets improves the reuse of this resource. For example, RPM+Com has a 14.8% higher saturation throughput than RPM+NonCom with 4 VCs per physical channel. As the number of VCs increases, this resource is not as precious; its effect on performance declines. However, even with 8 VCs per physical channel, dropping ACK packets still improves the saturation throughput by 9.6%.

Second, BAM+Com uses escape VCs to avoid deadlock. The horizontal escape VC can always be used, while the vertical one can only be used by DOR; there is some imbalance in the utilization of escape VCs and this imbalance increases with fewer VCs. However, the situation is worse for RPM+Com. In RPM's multicast VN configuration, the vertical dimension always has twice as many VCs as the horizontal one. Even with 4 VCs/port, the saturation point improvement of BAM+Com is still larger than RPM+Com's improvement by about 9.0%. With fewer VCs, RPM+Com's latency reduction increases; it achieves a 18.5% reduction with 4 VCs/port. BAM+Com further reduces latency due to the balanced buffer configuration among different dimensions. The latency difference between BAM+Com and RPM+Com is not as significant as the saturation throughput improvement. Adaptive routing mainly accelerates packet transmission at high injection rates by avoiding network congestion.

**Multicast Ratio.** Figure 17(b) presents the performance improvement for several multicast ratios: 5%, 10%, 15% and 20%. Increasing the multicast portion leads to greater throughput improvements due to message combination. The improvement contributed by the balanced buffer configuration remains almost constant (BAM+Com vs. RPM+Com). A higher multicast packet ratio triggers more ACK packets; ACK combination has more opportunity to reduce network load. The framework becomes more effective. BAM+Com achieves a 27.2% saturation throughput improvement with 20% multicast ratio. A higher multicast ratio also results in larger latency reductions. The VC count per physical channel is kept constant (8 VCs/port) in this experiment, so the gap between BAM+Com and RPM+Com remains almost the same.

**Destinations per Multicast.** Figure 17(c) illustrates the performance gain for different average numbers of multicast destinations: 3, 6, 9, 12, and 15 (broadcast). The trend is similar to varying the multicast ratio. Although the multicast ratio remains constant (10%), more destinations per multicast trigger more ACK packets. The framework combines more of these ACKs during transmission. As the destination count varies from 3 to 15, BAM+Com improves the saturation throughput by 17% to 28%. RPM+Com reduces latency by 10% to 25%; BAM+Com's latency reduction ranges from 13% to 27%.

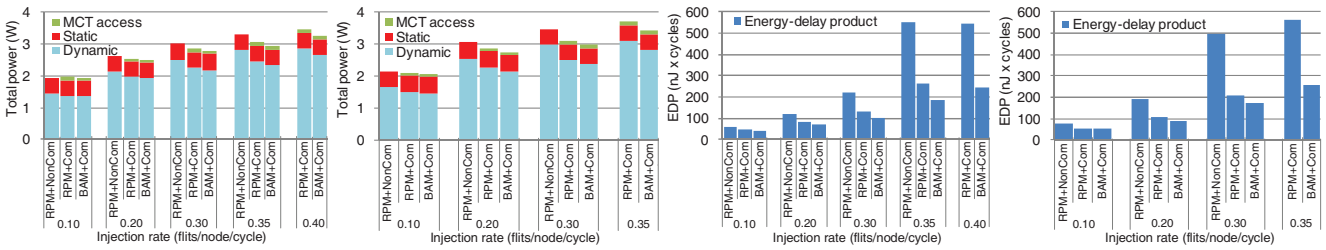
**Network Size.** Figure 17(d) shows the performance improvement for different network sizes: 4×4, 8×8 and 16×16 mesh networks[6]. Since 8×8 and 16×16 networks have more nodes, we increase the average destinations per multicast to 12 and 24, respectively. The message combination framework is more efficient at larger network sizes since packets traverse more hops on average. As a result, combining ACK packets eliminates more network operations. For the 16×16 network, the message combination framework improves the saturation point by 16.8%. Similarly, larger network sizes show greater latency reductions. RPM+Com and BAM+Com achieve 25% and 27% latency reductions for a 16×16 mesh, respectively. The efficiency of the balanced buffer configuration utilized by BAM+Com remains constant with different network sizes.

Throughout the sensitivity studies, the MCT size is 64 entries. One may think that with more multicast destinations, a higher multicast ratio or a larger network size that more entries will be required. Yet, analysis reveals changing these aspects reduces the injection rate at which the network becomes saturated. Thus, the maximal concurrently active multicast transactions supported by the network is reduced. As a result, a 64-entry MCT is able to achieve high performance for these different network design points.

## 6 Power Analysis

A NoC power model [30] is leveraged to determine overall network power consumption; network power consumption is contributed by three main components: channels, input buffers and router control logic. The activity of these components is obtained from Booksim. Leakage power is included for buffers and channels. The power consump-

[6]Bit string encoding is used in all experiments to encode the destination set; this method is impractical in large networks. However, further exploration of this issue is orthogonal to the message combination framework.




(a) Power (10% multicast). (b) Power (20% multicast). (c) EDP (10% multicast). (d) EDP (20% multicast).

**Figure 18. Power consumption and EDP results. (average 6 destinations.)**

tion of a MCT access is also integrated into the power model. We assume a 128-bit flit width. The technology process is 32nm and the network clock is 1 GHz based on a conservative assumption of router frequency. Figures 18(a) and 18(b) show the power consumption using transpose traffic for unicast messages. We measure the MCT access power and the static and dynamic network power for two multicast ratios: 10% and 20%.

MCT access power comprises only a small portion of the total power. The reason is two-fold. MCT is very small; each entry is only 3 bytes. With 64 entries, the size of MCT is 192 bytes, which is only 7.5% of the size of flit buffers. Second and more importantly, the MCT access activity is very low. Even when the network is saturated, only 7.2% of all cycles have an MCT access; if the network is not saturated, the activity is even lower.

RPM+Com reduces power consumption compared to RPM+NonCom for all injection rates due to the dropping of ACK packets. BAM+Com further reduces the power consumption due to more balanced buffer utilization among different dimensions. As the injection rate increases, the reduction in power consumption becomes more obvious. For 0.35 injection rate with a 10% multicast ratio, RPM+Com and BAM+Com achieve a 7.6% and a 10.8% power reductions over RPM+NonCom, respectively. With larger multicast ratios, the combination framework is able to reduce more power for both RPM+Com and BAM+Com.

The energy-delay product (EDP) [14] of the whole network further highlights the energy efficiency of our design, as shown in Figures 18(c) and 18(d). Dropping ACK packets during transmission not only results in fewer network operations, but also reduces network latency. For low-to-medium injection rates with a 10% multicast ratio, RPM+Com and BAM+Com show about 20%-40% EDP reductions. At a high injection rate (0.35), this reduction can be as much as 60%-75%. Higher multicast ratios result in greater EDP reduction.

## 7 Related Work

In this section, we review related work for message combination and NoC multicast routing.

**Message Combination.** Barrier synchronization is an important class of collective communication, in which a reduction operation is executed first followed by a broadcast. Gathering and broadcasting worms have been proposed [35]. Many supercomputers including the NYU Ultracomputer [15], CM-5 [24], Cray T3D [6], Blue Gene/L [2] and TH-1A [43] utilize dedicated or optimized networks to support the combination of barrier information. Oh et al. [34] observe that the using a dedicated network in many-core platform is unfavorable and propose the use of on-chip transmission lines to support multiple fast barriers. Our work focuses on collective communication that is used in cache coherence protocols where the multicast is sent first followed by the collection of acknowledgements. Bolotin et al. [4] acknowledge that ACK combination might be useful, but do not give a detailed design or evaluation. Krishna et al. [22] propose efficient support for collective communications in coherence traffic. Our message combination mechanism is quite different from their design. We utilize a CAM to record the ACK arrival information, while they keep the earlier arriving ACK in network VCs to merge with later arriving ones [22].

**NoC Multicast Routing.** Recent work explores various NoC multicast routing algorithms. Lu et al. [26] use path-based multicast routing, which requires path setup and acknowledgement messages resulting in a long latency overhead. Tree-based multicast mechanisms in NoCs avoid this latency overhead. VCTM [12] is based on the concept of virtual multicast trees. bLBDR [37] uses broadcasting in a small region to implement multicasting. RPM [39] focuses on achieving bandwidth efficient multicasting. Based on the isolation mechanism proposed in bLBDR, Wang et al. [41] extend RPM for irregular regions. MRR [1] is an adaptive multicast routing algorithm based on the rotary router. Whirl [22] provides efficient support for broadcasts and dense multicasts. The deadlock avoidance mechanisms of Whirl and our proposed BAM are similar; both are based on Duato's theory [8]. Both BAM and RPM use bit string encoding for multicast packet destinations; this method is not scalable for large networks. Some compression methods [40] can improve the scalability of this encoding scheme. In addition, coarse bit vectors [18], similar to what has been proposed for directories, are another possible ap-

proach to reduce the size of the destination set encodings. This type of encoding will increase the number of destinations per multicast and receive greater benefits from our proposal. Delving into this issue is left as future work.

## 8 Conclusions

Scalable NoCs must handle traffic in an intelligent fashion; to improve performance and reduce power they must eliminate unnecessary or redundant messages. The proposed message combination framework does just that by combining in-flight ACK responses to multicast requests. A small 64-entry CAM is added to each router to coordinate combination. In addition to the framework, we propose a novel multicast routing algorithm that balances buffer resources between different dimensions to improve performance. Simulation results show that our message combination framework not only reduces latency by 14.1% for low-to-medium network loads, but also improves the saturation point by 9.6%. The framework is more efficient for fewer VCs, larger network size, a higher multicast ratio or more destinations per multicast. The balanced buffer configuration achieves an additional 13.8% saturation throughput improvement. Our proposed message combination framework can be easily extended to support the reduction operations in Token Coherence and other parallel architectures.

## Acknowledgments

We thank the reviewers for their constructive suggestions and members of Prof. Enright Jerger's group for feedback on this work. We also thank Daniel Becker of Stanford for his detailed and valuable comments. This work is supported in part by the University of Toronto, NSERC of Canada, the Connaught Fund, 863 Program of China (2012AA010302), NSFC (61070037, 61025009, 60903039, 61103016), China Edu. Fund. (20094307120012), Hunan Prov. Innov. Fund. For PostGrad. (CX2010B032).

## References

[1] P. Abad *et al*. MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In *HPCA 2009*.

[2] N. R. Adiga *et al*. An overview of the BlueGene/L supercomputer. In *SC 2002*.

[3] C. Bienia *et al*. The PARSEC benchmark suite: characterization and architectural implications. In *PACT 2008*.

[4] E. Bolotin *et al*. The Power of Priority: NoC Based Distributed Cache Coherency. In *NOCS 2007*.

[5] C.-M. Chiang and L. M. Ni. Multi-address encoding for multicast. In *1st International Workshop on Parallel Computer Routing and Communication*, 1994.

[6] Cray Research Inc. Cray T3D system architecture overview. 1993.

[7] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA, USA, 2003.

[8] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320 – 1331, Dec. 1993.

[9] J. Duato *et al*. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[10] N. Enright Jerger. SigNet: Network-on-chip filtering for coarse vector directories. In *DATE 2010*.

[11] N. Enright Jerger and L.-S. Peh. *On-Chip Networks*. Morgan Claypool, 2009.

[12] N. Enright Jerger *et al*. Virtual Circuit Tree Multicasting: A case for on-chip hardware multicast support. In *ISCA 2008*.

[13] N. Enright Jerger *et al*. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO 2008*.

[14] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE J. Sol. St. Cir.*, 31(9):1277 –1284, sep 1996.

[15] A. Gottlieb *et al*. The NYU Ultracomputer - designing a MIMD, shared-memory parallel machine. In *ISCA 1982*.

[16] P. Gratz *et al*. Regional Congestion Awareness for load balance in networks-on-chip. In *HPCA 2008*.

[17] P. Gratz *et al*. Implementation and evaluation of a dynamically routed processor operand network. In *NOCS 2007*.

[18] A. Gupta *et al*. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In *ICPP 1990*.

[19] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, CA, USA, 3 edition, 2003.

[20] Y. H. Kang *et al*. Multicast routing with dynamic packet fragmentation. In *GLSVLSI 2009*.

[21] J. Kim *et al*. A low latency router supporting adaptivity for on-chip interconnects. In *DAC 2005*.

[22] T. Krishna *et al*. Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *MICRO 2011*.

[23] A. Kumar *et al*. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *ICCD 2007*.

[24] C. Leiserson *et al*. The network architecture of the connection machine CM-5. In *J. Parallel Distrib. Comput.*, pages 272–285, 1992.

[25] D. Lenoski *et al*. The directory-based cache coherence protocol for the DASH multiprocessor. In *ISCA 1990*.

[26] Z. Lu *et al*. Connection-oriented multicasting in wormhole-switched networks on chip. In *ISVLSI 2006*.

[27] S. Ma *et al*. DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In *ISCA 2011*.

[28] P. S. Magnusson *et al*. Simics: A full system simulation platform. *Computer*, 35:50–58, February 2002.

[29] M. Martin *et al*. Token coherence: decoupling performance and correctness. In *ISCA 2003*

[30] G. Michelogiannakis *et al*. Evaluating bufferless flow control for on-chip networks. In *NOCS 2010*.

[31] A. K. Mishra *et al*. A case for heterogeneous on-chip interconnects for CMPs. In *ISCA 2011*.

[32] N. Muralimanohar *et al*. Cacti 6.0: A tool to model large caches. Technical Report HPL-2009-85, HP Laboratories, April 2009.

[33] N. Neelakantam *et al*. FeS2: A full-system execution-driven simulator for x86. In *Poster presented at ASPLOS 2008*.

[34] J. Oh *et al*. TLSync: support for multiple fast barriers using on-chip transmission lines. In *ISCA 2011*.

[35] D. Panda. Fast barrier synchronization in wormhole k-ary n-cube networks with multidestination worms. In *HPCA 1995*.

[36] L.-S. Peh and W. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA 2001*.

[37] S. Rodrigo *et al*. Efficient unicast and multicast support for CMPs. In *MICRO 2008*.

[38] F. Samman *et al*. New theory for deadlock-free multicast routing in wormhole-switched virtual-channelless networks-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 22(4):544 –557, April 2011.

[39] L. Wang *et al*. Recursive Partitioning Multicast: A bandwidth-efficient routing for networks-on-chip. In *NOCS 2009*.

[40] L. Wang *et al*. Efficient lookahead routing and header compression for multicasting in networks-on-chip. In *ANCS 2010*.

[41] X. Wang *et al*. On an efficient NoC multicasting scheme in support of multiple applications running on irregular sub-networks. *Microprocess. Microsyst.*, 35:119–129, March 2011.

[42] H. Xu *et al*. Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers. In *ICDCS 1992*.

[43] X. Yang *et al*. The Tianhe-1A supercomputer: Its hardware and software. *J. Comput. Sci. Technol.*, 26(3):344–351, 2011.