Diamond Rings: Acknowledged Event Propagation in Many-Core Processors

Stefan Nürnberger[1], Randolf Rotta[1], Gabor Drescher[2], Daniel Danner[2], and Jörg Nolte[1]

[1] Brandenburg University of Technology, Cottbus-Senftenberg, Germany
{snuernbe,rrotta,jon}@informatik.tu-cottbus.de
[2] Friedrich-Alexander University Erlangen-Nuremberg, Germany
{drescher,danner}@cs.fau.de

Abstract. Hardware and software consistency protocols rely on global observability of consistency events. Acknowledged broadcast is an obvious choice to propagate these events. This paper presents a generalized ring topology for parallel event propagation with acknowledged delivery. Implementations for various many-core architectures show increased performance over conventional approaches. Therefore, diamond rings are a prime candidate for distributed memory model implementations.

1 Introduction

In software consistency protocols, a central building block is a low-latency, high-throughput notification mechanism that guarantees observability to the initiator. For this purpose, we propose a new broadcast topology we call *diamond rings*. While, in the context of software consistency, broadcast values (i.e., events) are usually rather small, acknowledgements for event propagation are crucial for a series of operations in this context. Events concerning large amounts of data usually only reference this data instead of transmitting it to every receiver. It is the receivers' duty to copy or update any additional data as necessary. Furthermore, observability of the event is often the most important property for consistency. Event handling may be postponed locally until a core's actions require the event to take effect, e.g., apply an invalidation queue.

A prime example of consistency related event propagation are coherence protocols. Coherence provides the multiple reader single writer (MRSW) invariant for memory locations. Before a write to memory may be performed, exclusive write access must be obtained by the respective node. The node that requires write access sends out a request for ownership of the location, while at the same time fetching the up-to-date value. Every node that has a valid copy must be informed of this request. Depending on the memory model specification, they usually respond with an immediate invalidation of their copy. At the very least, downgrading the copy to read-only mode is mandatory. The read for ownership is finished when the requesting node can be sure that all other copies have been downgraded and the up-to-date value has been retrieved. Therefore acknowledging the success handling of a request is essential to the originator. Low-latency is

clearly an issue for coherence traffic, but so is high throughput. The coordinating node of coherence traffic is a sequencer of ownership requests. It is under high pressure from all partakers in shared memory operations. Outstanding requests must be pipelined for maximum throughput. Other examples of high throughput, low-latency operations in consistency management include TLB shootdowns, and atomic operations.

Broadcast topologies provide a trade-off between latency and throughput. The latency is determined by processing overheads along the longest path. Keeping it short requires asymmetric topologies with a large number of communication partners per node [1], which introduces imbalanced overhead among the nodes. High throughput is usually achieved by pipelining multiple broadcasts. In this scenario, the throughput is mainly limited by the node with the most processing overhead, that is, with the most communication partners. Thus, broadcast topologies with balanced overhead and few communication partners per node increase the throughput. An extreme example are rings with a single predecessor and successor per node.

With the above assessment, conventional balanced trees present a sensible trade-off between latency and throughput. Acknowledgements require an additional reduction along the tree. The proposed diamond rings provide better throughput and latency than conventional balanced trees for acknowledged broadcasts. In comparison to asymmetric topologies, they provide better throughput in exchange for slightly worse latency.

Section 2 lists related work regarding broadcast and multicast topologies. In Section 3 we introduce the new diamond ring broadcast topology and state its benefits over common multicast trees. The broadcast topology was implemented for a variety of modern multi- and many-core architectures, and exercised in microbenchmarks. We present our results as well as the comparison to balanced multicast trees in Section 4.

2 Related Work

Efficient multicast-operations receive a lot of active research. This research can be divided into three categories: theoretical work, software-, and hardware-implementations. The theoretical work focuses on finding optimal broadcasting trees for specific models of computation and communication networks. [2] summarizes early work in this field for very basic models of communication, lacking message-latency and simple assumptions on the size of messages and the amount that can be sent simultaneously. For these straightforward models, static and minimal broadcast graphs were designed in which each node has the same cost for a broadcast operation. The most prominent enhanced models are the POSTAL[3] and the LogP[4] models. The POSTAL model considers communication latency and simultaneous I/O in the sense that a node can simultaneously receive and send a message. The LogP model extends on these parameters by incorporating processor overhead, communication bandwidth, finite network ca-

pacity and multi-port I/O. Optimal algorithms for building broadcast trees in the more general LogP model were found by [1].

Implementations of these models pose additional problems like varying network latencies and efficient computation of optimal broadcast-trees at runtime [5]. Research in networks with multiple communication-media that employ different latencies and the construction of minimal-height lopsided trees to model these latencies in packet-switching networks were done by [6]. Other implementations focus on the special network topology or special hardware to implement efficient multicast-operations. [7] evaluates different broadcasting algorithms on the Intel SCC [8], which is one of our evaluation platforms. Algorithms for the SCC that use the 2D physical structure of the network on chip to explicitly send messages to neighboring CPUs or tiles in a flood-fill style, perform better than naive implementations. Their best algorithm uses SCCs message-passing buffers that can be read like shared memory by other cores. This is done in a flat tree-layout that facilitates parallel operations.

In contrast to hardware-based approaches [9,10,11] that provide efficient multicast operations on router or switch level, our algorithm can be implemented in software and does not require specialized hardware other than simple point-to-point communication.

None of the previous works, theoretical, software- or hardware-based, considered multicasts or broadcasts with acknowledgement to the root of the operation as a combined operation. Analyzed were broadcasts without acknowledgement or, as in [12], only the acknowledgement part of a broadcast operation is developed in an efficient way.

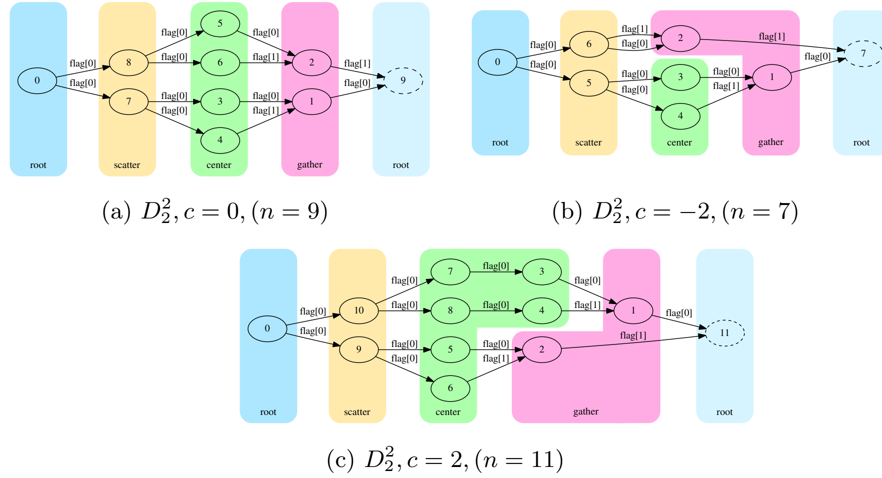
3 The Diamond Ring Structure

We propose a multicast topology based on a generalized parallel ring. The ring is a directed graph $D_k^l = (V, E)$ with $|V| = n$ vertices. It consists of four classes of nodes, called root, scatter, center, and gather nodes. The overall shape can be described as a mirrored k-ary tree with the root and its mirror being identical. This directed graph contains exactly k^l center nodes (leaves of the tree) and just as many cycles, i.e., rings. Each center node is part of exactly one cycle. The root node is the only node that is featured in every cycle of the graph.

Our topology has a couple of properties deemed beneficial for the implementation of low overhead network communication. The path length is bound by $O(\log n)$ due to the construction based on trees. Per-node memory requirement is $O(1)$. Memory is needed for the communication with neighbors in the topology, which is bounded by k (a small constant). We assume separate reception buffers for messages from different nodes.

3.1 Extending to Arbitrary Node Counts

The node count in a particular application may not match the exact number of vertices of a pure D_k^l. In such cases, c additional vertices must be introduced



(a) $D_2^2, c = 0, (n = 9)$

(b) $D_2^2, c = -2, (n = 7)$

(c) $D_2^2, c = 2, (n = 11)$

Fig. 1: Diamond Rings of arity 2 with different values of contamination. Nodes 0 and n are conceptually the same node but drawn separately for clarity.

into the graph beyond its regular topology, turning the graph into what we call a *contaminated* diamond ring. Fortunately, the number of surplus vertices never exceeds the breadth of the ring center, therefore all potential modifications can be accomplished by inserting or removing vertices in that class. With the center being the part exhibiting the highest degree of parallelism, this also implies that the cycle lengths are not increased by more than 1. One can either insert up to k^l additional nodes, forming chains of 2 vertices in the center, or, as an alternative, remove up to k^l vertices, skipping the center part. When deciding for the proper l for a given node count n, if $n > |V(D_k^l)| + k^l$ (i.e., more than k^l nodes have to be added), just choose D_k^{l+1} and remove some of the k^{l+1} nodes from the center. This always suffices, since

$$|V(D_k^{l+1})| = |V(D_k^l)| + k^l + k^{l+1}$$

There are alternative ways to handle contamination. For example, one can only add further center nodes. This is beneficial since the removal of nodes in the larger graph leads to multiple connections between the last level scatter and first gather node. However, the length of the longest path is also increased in this scheme. Figure 1 shows some examples for D_2^2 with different values of contamination.

3.2 Numbering and Addressing Scheme

A node is able to determine its neighbors solely by using its own node ID, the graph arity, and number of nodes in the topology. Beyond that knowledge, no further communication between the participating nodes is needed for topology

setup. The parameters l (level, or depth) and c (contamination) are determined as $min\{l \mid -(k^l) < c \leq k^l\}$, where $c = n - |V(D_k^l)|$. The node class (scatter, gather, etc.) is then determined by Equation 1. The neighborhood of a node is determined by Equation 2 for a D_k^l with $c = 0$. Considering contamination requires some special cases, which are not shown here for brevity. Also omitted is the computation of offsets for root and gather nodes that receive messages from multiple predecessors, which requires basically a residue check. This is only needed when disjoint message buffers or queues are used for the k predecessors.

$$type(id) = \begin{cases} \text{root node,} & \text{if } id = 0 \\ \text{scatter node,} & \text{if } \frac{n+k^l+1+c}{2} \leq id \\ \text{center node,} & \text{if } \frac{n-k^l-1}{2} < id < \frac{n+k^l+1+c}{2} \\ \text{gather node,} & \text{otherwise} \end{cases} \tag{1}$$

$$neighbors(id) = \begin{cases} \{n - i \mid i = 1..k\}, & \text{if } type(id) = \text{root} \\ \{n - (k(n - id) + i) \mid i = 1..k\}, & \text{if } type(id) = \text{scatter} \\ \{(id - 1)/k\}, & \text{if } type(id) = \text{center} \\ \{(id - 1)/k\}, & \text{if } type(id) = \text{gather} \end{cases} \tag{2}$$

3.3 Comparison to Tree-Based Broadcast with Reduction

The advantage of diamond rings over broadcast trees with reduction for acknowledgement is the significantly reduced number of messages sent. Considering the center of a diamond ring with k^l nodes, the balanced tree requires exactly one more level to reach all nodes in the graph before beginning the reduction. Together with the first reduction step to get back to the diamond ring's center, it adds up to $2 * k^l$ additional messages sent for the tree based reduction. The total number of nodes in a broadcast tree is $k^{l+1} - 1$. That means twice the number of messages for binary trees ($k = 2$). The factor decreases with larger k. With the same reasoning, it becomes clear that the longest path from the root, through all nodes, and back to the root is exactly two hops shorter for diamond rings.

The most important property of this broadcast is the reduced workload on inner nodes. Each node is an active part of the diamond ring broadcast structure exactly once. In contrast, balanced trees require the inner nodes to forward the messages *and* the acknowledgements later on. They are active in communication twice for a single broadcast. In the diamond ring this is only the case for the root node, sending out a request and later receiving the acknowledgement. Inner nodes in the balanced tree engage in $2 * (k + 1)$ active communications either sending or receiving a message.

However, there is also a drawback for diamond rings in comparison to balanced trees regarding latency. In the tree broadcast the first thing a receiving node does is forward the message. Then it will continue with the required work before handling the acknowledgement. In a diamond ring, all work that needs to be acknowledged must be performed before the message is forwarded to the

neighbors. Depending on the amount of work that is required for each message, latency may be worse for the diamond rings despite the marginally shorter hop count. If message reception (i.e., observability) is the only criterion that needs acknowledgement, forwarding may take place immediately. In a pipelined scenario, this increased latency does not affect overall throughput.

3.4 Root Node Overhead

In the proposed diamond ring, a root node sends messages to k different neighbors and also receives k messages. This is in contrast to all other nodes that engage in communication with a maximum of $k + 1$ neighbors in total. For throughput-oriented workloads, this may constitute a bottleneck. The situation can be alleviated quite easily. First, an additional gather node is introduced as companion to the root. This node takes the position of node n in Figure 1 and forwards the acknowledgement directly to the root. Hop count is thereby increased by one in comparison to the original diamond ring structure, leaving an advantage of one hop over balanced trees. Which of the two nodes (0 or n) is considered root and which helper is a matter of taste. The modified ring does not contain a single node with increased communication overhead, thus eliminating throughput bottlenecks.

4 Evaluation

This section evaluates the latency and throughput of acknowledged event broadcasts based on diamond rings against balanced trees. Trees are commonly used to propagate events and collect acknowledgements. Balanced trees were chosen because they achieve better throughput than skewed/asymmetric trees.

The measurements are carried out on the Tilera TILE-Gx72 and Intel Xeon-Phi many-core architectures. For comparison, a large multi-core Intel Xeon machine is included. Both many-core architectures are based on in-order execution cores whereas the multi-core utilises out-of-order execution. All three provide cache-coherent shared memory, which we use only for point-to-point message passing. Hence, porting to non-coherent shared memory is possible. The Tilera architecture features a low-latency network, which we use for message delivery.

Based on the analysis in Section 3.3, following hypotheses are examined: 1) The longest path is shorter in diamond rings than in balanced trees. Hence, diamond rings should have a slightly lower latency as long as processing the broadcast event itself costs no time. 2) The balanced tree nodes have to process more messages than diamond ring nodes. Hence, the latter should provide higher throughput. In consequence, diamond rings provide better trade-offs between throughput and latency than balanced trees and trees in general.

In order to evaluate these hypotheses, latency and throughput were measured in micro-benchmarks without actual application-level event payload. The latency is the time needed to complete individual broadcasts on otherwise idle

cores. The throughput is measured with bursts of up to 128 pipelined broadcasts. In contrast to application benchmarks, this approach allows to study the performance differences in isolation. Nevertheless, the benchmarks are based on a general purpose messaging framework such that the results are relevant for actual applications.

The next subsection summarises implementation details of the three benchmarks: diamond rings (DR), sequenced diamond rings (SDR), and balanced trees (BT). The following subsections present individual results for the three evaluation architectures and the final subsection discusses the overall results.

4.1 Benchmark Variants

The benchmarks are based on a lightweight task scheduling framework with nodes representing the cores or hardware-threads. Each node has a deque for local task scheduling and a message queue for asynchronous inter-node communication. Tasks and messages are simply pointers to task objects, which reside in shared memory. The task objects consist of a pointer to a handler function and additional payload for the handler similar to active messages. The handler functions run to completion, that is, cannot suspend their own execution. Continuation tasks are used to represent causal dependencies.

Balanced Trees (BT). The reference implementation is based on a balanced n-ary tree topology. Upon receiving a broadcast message, each node sends the broadcast message to its children and, then, processes the broadcast event locally. An acknowledgement message is sent back to the parent node after the local processing is completed and acknowledgement messages were received from all children. A separate node-local acknowledgement counter is used for each broadcast to track the outstanding acknowledgements at each node.

Diamond Rings (DR). The nodes operate differently depending on their position in the diamond ring. All scatter, center, and gather nodes begin processing the broadcast event on the first message received. They propagate the broadcast message to their successors after the local processing is complete and the broadcast message was received from each predecessor. Scatter and center nodes have just a single predecessor. The root node first sends the broadcast to its successors and, then, processes the event itself. The broadcast is completed at the root node after the local event processing completed and the broadcast message was received from each of the root's predecessors. Again, a node-local counter is used by gather nodes and the root to track outstanding messages.

Sequenced Diamond Rings (SDR). Tree and diamond ring topologies do not mandate any ordering of concurrently propagated events. The communication layer and the node-local task scheduling can reorder their messages and tasks. However, many application scenarios like, for example, request for ownership

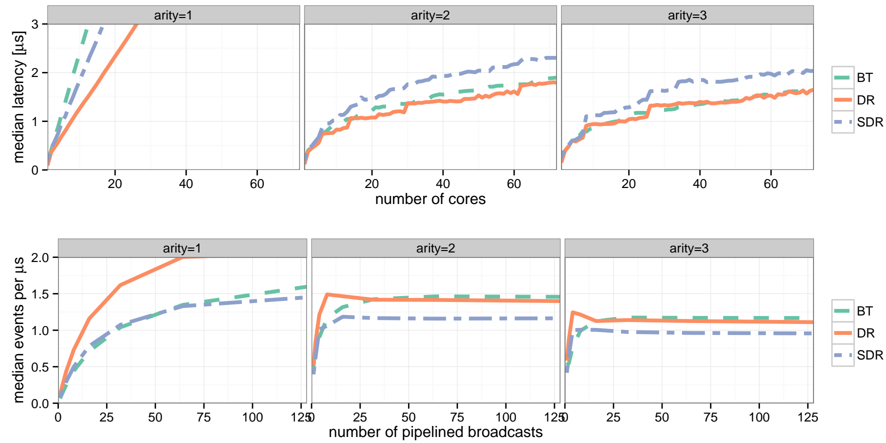


Fig. 2: Median latency and throughput on Tilera TILE-Gx72.

and distributed atomic updates require a strict ordering according to the event sequence at the root node.

The sequenced diamond ring implementation enforces in-order processing of pipelined broadcasts at each node. For this purpose, the root node assigns a sequence number to each event. A node-local sequence counter is used to delay broadcast tasks that are out of sequence. The sequencing can be implemented orthogonal to the broadcasts but a combined implementation was chosen to exploit cross-cutting optimisation opportunities.

4.2 72-core Tilera TILE-Gx72

This many-core processor contains 72 three-issue in-order VLIW cores running at 1GHz. The cores are interconnected through several 2D mesh networks. The benchmark implementation uses Tilera's low-latency user-dynamic network (UDN) to communicate the pointers to messages and shared memory to access the message contents. Figure 2 shows the latency and throughput results for arity 1, 2, and 3. All measurements were repeater 100 times. The mapping of topology nodes to cores was not specifically optimised. The highest throughput was achieved with arity 1 and diamond rings. With arity 2, the throughput is similar for diamond rings and balanced trees and diamond rings have a slightly better latency. On this processor, processing overheads seem to dominate the communication overhead and latency significantly. The additional processing needed to order pipelined broadcasts in the sequenced diamond rings increases the latency compared to un-ordered balanced trees while the throughput is similar.

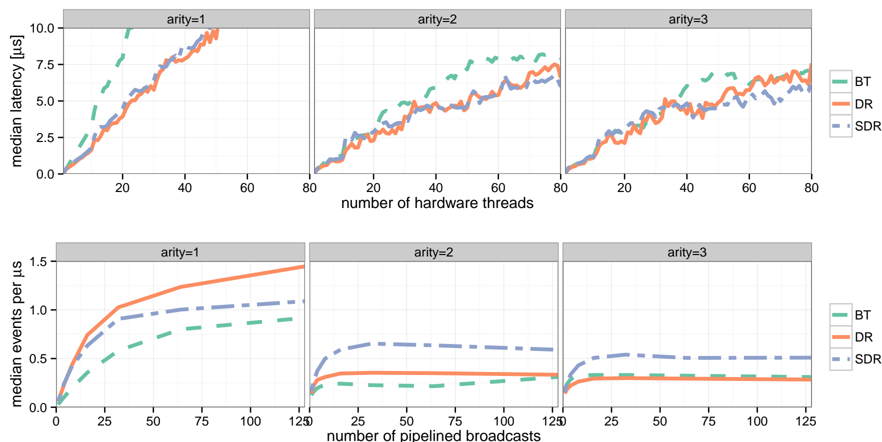


Fig. 3: Median latency and throughput on Intel Xeon E5 4640v2.

4.3 4x10-core Intel Xeon E5 4640v2

This machine consists of 4 processors with 10 out-of-order cores per processor and two hardware-threads per core running at 2.2GHz. The processors are connected through a cache-coherent QPI network. The communication is implemented via shared memory using a multiple-producer/single-consumer ring-buffer. Figure 3 shows the results for this machine. Again, the highest throughput was achieved with arity 1 and diamond rings. With arity 2 and 3, the throughput of sequenced diamond rings the best while the latency is similar for all three. Surprisingly, the sequenced diamond rings perform best, which could benefit from two characteristics of the architecture: The cores are designed for good single-thread performance, which compensates the additional processing overhead; And the enforced ordering might reduce the pressure on the limited bandwidth of inter-processor QPI links.

4.4 60-core Intel XeonPhi 5110P

This many-core processor contains 60 in-order cores with 4 hardware-threads per core running at around 1GHz. The cores are connected through two bi-directional rings. The benchmark uses the same ring-buffer implementation as above. Figure 4 shows the results for this machine. The best throughput was achieved with arity 2 instead of 1. This is probably due to the arity 1 pipeline being much longer (240 stages) than the number of pipelined broadcasts. Diamond rings achieved better throughput and latency than balanced trees. Ramos and Hoefler [13] presented an optimal design for small broadcasts and reductions on the XeonPhi processor using dedicated communication structures. For 60 cores, they report 10µs latency for broadcasts plus 10µs latency for reductions, i.e. acknowledgements. In comparison, the balanced 2-ary tree (24µs) and

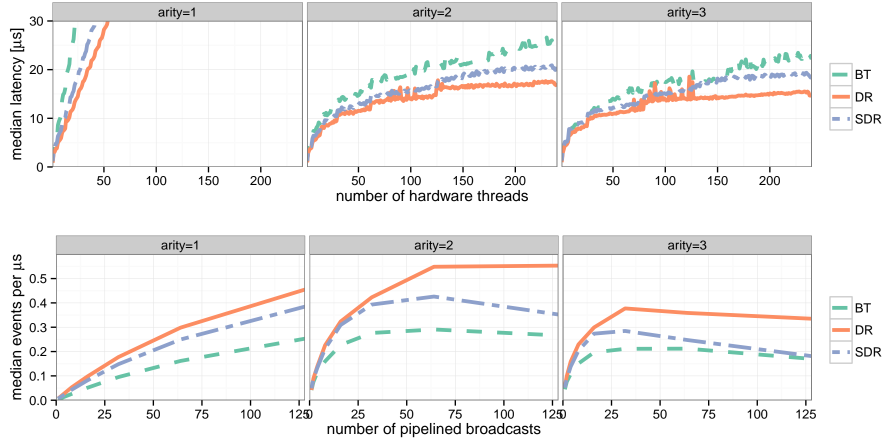


Fig. 4: Median latency and throughput on Intel XeonPhi 5110P.

2-ary diamond ring (15µs) presented here perform quite well while reaching all 240 threads and using general-purpose communication channels.

4.5 Discussion of the Results

Figure 5 compares the trade-off between latency and throughput directly for a larger selection of tree arities. The x-axis shows the peak median throughput and the y-axis the median latency for each benchmark variant. The first column shows the TILE-Gx72. The Pareto optimal variants, beginning with lowest latency, are diamond rings with arity 6 (1.6µs, 0.7 per µs), 4, 3, 2 (1.8µs, 1.5 per µs), and finally diamond rings with arity 1 (8µs, 2 per µs). For most arities, diamond rings performed better than balanced trees, which performed better than sequenced diamond rings.

The second column shows the results for the multi-core Xeon. The Pareto optimal variants, beginning with lowest latency, are sequenced diamond rings with arity 6 (5.4µs, 0.35 per µs), then arity 2(6µs, 0.65 per µs), and finally diamond rings with arity 1(17µs, 1.45 per µs). For most arities, sequenced diamond rings performed better than diamond rings, which performed better than balanced trees. Finally, the third column represents the many-core XeonPhi. The Pareto optimal variants, beginning with lowest latency, are diamond rings with arity 3 (15µs, 0.37 per µs) and finally arity 2 (17µs, 0.55 per µs). For most arities, the diamond rings performed better than sequenced diamond rings, which performed better than balanced trees.

On all three architectures, diamond rings achieved a higher throughput than balanced trees. The latency can be reduced by using larger arities. However, the latency does not decrease much beyond arity 2 while the throughput degrades quickly. In conclusion, diamond rings and sequenced diamond rings with arity 2 are a good choice for acknowledged event delivery.

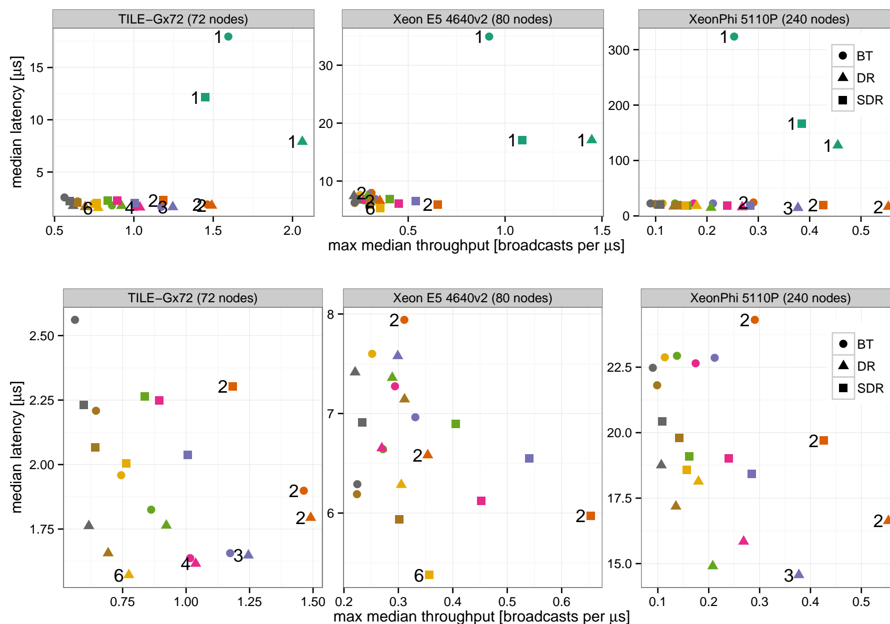


Fig. 5: Comparison of the single-broadcast latency against the largest observed throughput with pipelined broadcasts. Both figures show the same dataset but the lower figure excludes arity 1 for better readability. The arity is encoded by color and is, in addition, written left to interesting points. The better variants are to the lower right corner. Please note the different scales.

5 Conclusions

We have presented a novel topology for efficient acknowledged broadcast to be used in memory consistency protocols. By combining the advantages of low latency in tree-based topologies and the high throughput achieved in ring-shaped communication, our diamond ring topology balances the overall utilization of the network and resource requirements on the participating nodes. We have implemented diamond rings on a multitude of platforms and compared their performance to existing approaches. Referring to the hypotheses made in Section 4, our evaluation results show that 1) the shorter path lengths in diamond rings result in lower latencies on all measured platforms, 2) the reduced computational load per node gives rise to higher overall throughput performance in diamond rings when compared to balanced trees. The choice of arity offers a trade-off decision between both those performance indicators, with higher arity reducing the latency at the cost of throughput. This shows that diamond rings constitute a prime candidate for use as an underlying communication layer in software memory consistency protocols.

References

1. Karp, R.M., Sahay, A., Santos, E.E., Schauser, K.E.: Optimal broadcast and summation in the logp model. In: Fifth Annual ACM Symposium on Parallel Algorithms and Architectures. SPAA '93, ACM (1993) 142–153
2. Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.L.: A survey of gossiping and broadcasting in communication networks. Networks **18**(4) (1988) 319–349
3. Bar-Noy, A., Kipnis, S.: Designing broadcasting algorithms in the postal model for message-passing systems. SPAA '92, ACM (1992) 13–22
4. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., Von Eicken, T.: LogP: Towards a realistic model of parallel computation. Volume 28. ACM (1993)
5. Bruck, J., De Coster, L., Dewulf, N., Ho, C.T., Lauwereins, R.: On the design and implementation of broadcast and global combine operations using the postal model. IEEE Trans. on Parallel and Distributed Systems **7**(3) (1996) 256–265
6. Golin, M., Schuster, A.: Optimal point-to-point broadcast algorithms via lopsided trees. Discrete Applied Mathematics **93**(2) (1999) 233–263
7. Matienzo, J., Jerger, N.E.: Performance analysis of broadcasting algorithms on the intel single-chip cloud computer. In: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on, IEEE (2013) 163–172
8. Howard, J., et al.: A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In: Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International, IEEE (2010) 108–109
9. Jerger, N.E., Peh, L.S., Lipasti, M.: Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In: ISCA'08, IEEE (2008) 229–240
10. Malumbres, M.P., Duato, J.: An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors. Journal of systems architecture **46**(11) (2000) 1019–1032
11. Turner, J.S.: An optimal nonblocking multicast virtual circuit switch. In: Networking for Global Communications., 13th Proceedings IEEE, IEEE (1994) 298–305
12. Rothermel, K., Maihofer, C.: A robust and efficient mechanism for constructing multicast acknowledgement trees. In: Computer Communications and Networks, 1999. Proceedings. Eight International Conference on, IEEE (1999) 139–145
13. Ramos, S., Hoefler, T.: Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi. In: Proceedings of the 22nd Int. Symp. on High-Performance Parallel and Distributed Computing, ACM (2013) 97–108