# Virtual Tree Coherence: Leveraging Regions and In-Network Multicast Trees for Scalable Cache Coherence

Natalie D. Enright Jerger
*Dept of Electrical and Comp. Engineering*
*University of Wisconsin-Madison*
*Madison, WI 53706*
*Email: enrightn@cae.wisc.edu*

Li-Shiuan Peh
*Dept of Electrical Engineering*
*Princeton University*
*Princeton, NJ 08544*
*Email: peh@princeton.edu*

Mikko H. Lipasti
*Dept of Electrical and Comp. Engineering*
*University of Wisconsin-Madison*
*Madison, WI 53706*
*Email: mikko@engr.wisc.edu*

## Abstract

*Scalable cache coherence solutions are imperative to drive the many-core revolution forward. To fully realize the massive computation power of these many-core architectures, the communication substrate must be carefully examined and streamlined. There is tension between the need for an ordered interconnect to simplify coherence and the need for an unordered interconnect to provide scalable communication. In this work, we propose a coherence protocol, Virtual Tree Coherence (VTC), that relies on a virtually ordered interconnect. Our virtual ordering can be overlaid on any unordered interconnect to provide scalable, high-bandwidth communication. Specifically, VTC keeps track of sharers of a coarse-grained region, and multicasts requests to them through a virtual tree, employing properties of the virtual tree to enforce ordering amongst coherence requests. We compare VTC against a commonly used directory-based protocol and a greedy-order protocol extended onto an unordered interconnect. VTC outperforms both of these by averages of 25% and 11% in execution time respectively across a suite of scientific and commercial applications on 16 cores. For a 64-core system running server consolidation workloads, VTC outperforms directory and greedy protocols with average runtime improvements of 31% and 12%.*

## 1. Introduction

As transistor scaling affords chip architects the ability to place more and more cores on-chip, the coherence and communication substrates are going to become a significant performance bottleneck. Already we are seeing dozens of cores available on-chip from Intel [34] and Tilera [35]. Efficient coherence mechanisms are required to continue scaling these systems into the future.

Throughout literature, there are two predominant classes of cache coherence, broadcast-based protocols and directory-based protocols. Both these forms of coherence protocols exhibit problems for emerging many-core chip multiprocessors. Broadcast protocols do not scale well in performance and power since they rely on a central ordering point for all coherence requests and flood the interconnect with broadcast requests. Directory protocols provide greater scalability by distributing the ordering points to various directories; but latency penalties are paid for traveling to and accessing these ordering points. In large multi-chip multiprocessor systems, it is fairly easy to add extra bits to memory for directory storage and use directories to build scalable coherence protocols. However, to make performance acceptable for on-chip usage, directory information must be stored on-chip (not as part of off-chip memory as is done in distributed systems). Caching directory information on-chip represents substantial overhead; this additional area could be better utilized for other CMP components such as the core itself.

In [22], significant directory cache miss rates were observed for commercial workloads (up to 74%). Directory cache miss rates will become an even more significant performance problem with server consolidation workloads. Multiple server workloads sharing the same resources on chip will touch large amounts of memory, placing enormous pressure on these directory caches. So, while directory protocols do provide designers with scalability; they do so with significant performance degradations when directory indirections and directory misses are factored in.

Even with hundreds to thousands of cores on-chip, in the common case, only a few processors need to observe a given coherence request. With the single-writer, multiple-reader protocol invariant, only cores that are actively caching a block need to be made aware of a pending write to that block. Since providing global coherence, such as a broadcast, across thousands of nodes is impractical from both a performance and a power standpoint, a logical solution is to maintain coherence amongst just the current subset of readers. Infrequently, it will be the case that every core on-chip does need to observe a coherence request; this case must be handled correctly but not necessarily quickly.

In short, to address these key shortcomings of broadcast and directory-based protocols, the desirable properties for scalable on-chip coherence are:

- **Limit coherence actions to the necessary subset of nodes:** This will reduce power consumption and interconnect pressure.
- **Fast cache-to-cache transfers:** Send request to sharers as quickly as possible avoiding overheads of directory indirections.
- **Limited bandwidth overhead:** Limit unnecessary broadcasts to entire chip; only communicate amongst the subset of sharers where coherence needs to be maintained.
- **Limited storage overhead:** Make efficient use of on-chip directory cache storage and cache tag array storage.

The need for scalable coherence is not a new one; however,

This research was supported in part by the NSF under grants CCR-0133437, CCF-0429854, CCF-0702272, CNS-0509402, the MARCO Gigascale Systems Research Center, an IBM PhD Fellowship, as well as grants and equipment donations from IBM and Intel. The authors would like to thank Ravi Rajwar and the anonymous reviewers for their comments and suggestions on improving this work.

the tight integration and the abundance of on-chip resources provide a new set of opportunities in which to explore this significant challenge. The tight coupling and integration of dozens of computation elements on-chip necessitates the co-design of various components of the memory hierarchy. This work specifically leverages the important interplay of the on-chip interconnection network and the cache coherence protocol to achieve scalable on-chip coherence.

Specifically, in this paper, we propose Virtual Tree Coherence (VTC), which targets and achieves each of the above attributes of an ideal scalable on-chip coherence protocol:

**Limit coherence actions to the necessary subset of nodes:** We use virtual trees to connect and order sharers, with requests multicast through these virtual trees so coherence actions are usually limited to true sharers and not broadcast to all nodes. These virtual trees are the virtual circuit trees recently proposed to support efficient multicasting in on-chip networks [14] and can be mapped onto any unordered interconnect, thus enabling scaling to many-core chips.

**Fast cache-to-cache transfers:** The root of each virtual tree is used as the ordering point where all requests are first sent to and ordered, much like the home node of a vanilla directory protocol. However, as the root is one of the sharers and not a statically mapped home node that may be far away, the cost of indirection is reduced considerably.

**Limited bandwidth overhead:** By multicasting coherence only to the current sharers, VTC avoids the bandwidth overhead of broadcast-based protocols, while approaching their benefit of fast cache-to-cache transfers.

**Limited storage overhead:** VTC allocates virtual trees for sharers of coarse-grained regions [9], [26], rather than per cache line. The sharers of each region are tracked in local structures and are guaranteed to completely capture all current sharers of a region. Tracking on a coarse granularity allows VTC to reduce the state that needs to be maintained, and limits storage overhead.

We show that VTC improves performance by an average of 25% (16-core) and 31% (64-core) over a directory protocol and reduces interconnect bandwidth by an average of 35% (16-core) and 68% (64-core) over a broadcast protocol.

The rest of this paper is organized as follows: background information on the underlying interconnection network and recent coarse grain optimizations is provided in Section 2. The proposed coherence ordering mechanism, Virtual Tree Coherence is explained in Section 3 with its implementation details discussed in Section 4. Results are presented in Section 5 followed by related work in Section 6. Finally the paper is concluded in Section 7.

## 2. Background

Two key considerations for a coherence protocol are the network and the storage of state information. In this section, we provide background on the use of coarse grain tracking of coherence state as well as the multicast network design that we leverage for Virtual Tree Coherence.

### 2.1. Regions

In conventional systems, information about cache coherence is maintained on a per-block granularity. However, by tracking coherence information across multiple contiguous addresses (a region), more optimizations can be enabled. A region is defined as a contiguous portion of memory consisting of a power of two number of cache blocks. A cache line address maps to one and only one region and will map to the same region for all processors.

Coarse Grain Coherence Tracking (CGCT) [9] has been proposed to eliminate unnecessary broadcasts in order to improve the scalability of broadcast-based systems. Requests to non-shared regions of the address space can send a request directly to the memory controller rather than order their request on the broadcast bus which is a precious and limited resource. Tracking sharing patterns on a coarser granularity than cache lines can take advantage of spatial locality and reduce the storage overhead associated with maintaining this information. RegionScout [26] makes similar observations about the benefits of tracking information on a coarse granularity for coherence purposes.

Tracking coherence state on a region granularity has several benefits. If a processor holds a region exclusively, it can upgrade any cache line in that region to a modified state without sending a request to other processors. If a processor knows no other core is caching a region, it can send its cache miss directly to memory without snooping those caches. The CGCT work tracks *if* other cores are caching a region; we expand this to encompass not only *if* but *who* is caching each region.

The structures required by CGCT and RegionScout have been generalized in RegionTracker [37] to scalably incorporate additional functionality. Specifically, RegionTracker replaces a conventional tag array with region tracking structures, and is shown to achieve comparable performance to a conventional fine-grained tag array with the same area budget. With RegionTracker, one structure is used to encompass the functionality of both the fine-grained tags of a conventional cache and maintain additional information about larger regions. For details on how RegionTracker achieves this low overhead, we refer the reader to [37].

Further distinguishing this work from previous proposals that used region tracking structures to filter away unnecessary broadcasts [9], [26], perform DRAM-speculation [3], and prefetching [10], Virtual Tree Coherence leverages these structures to keep track of the current set of sharers of a region. Additional bits are thus added to the Region Vector Array (RVA) of RegionTracker to track the current region sharers and the region root node (these additions will be discussed in Section 3). The low overhead design of RegionTracker allows us to optimize for region based sharing with only a very modest area increase over a conventional L2 cache design.

### 2.2. Virtual circuit tree multicasting (VCTM)

To reduce bandwidth overhead, Virtual Tree Coherence utilizes multicasting to send requests to sharers within a region, rather than broadcasting to all nodes. However, as state-of-the-art on-chip networks do not support multicasting (due to the high hardware overhead of providing such support), multicast functionality has to be synthesized by sending multiple unicast messages destined for multiple nodes. This creates high bandwidth overheads which translate to high interconnect delay and power, reducing the attractiveness of multicast coherence protocols such as Virtual Tree Coherence.

Recently, a lightweight multicast network design, Virtual Circuit Tree Multicasting (VCTM), that can be implemented within the tight delay, area and power budgets of on-chip networks has been proposed [14]. VCTM thus makes multicast coherence protocols such as the Virtual Tree Coherence proposed here viable for CMPs.

VCTM essentially overlays virtual multicast trees on any unordered interconnect (e.g. mesh, torus, hierarchical meshes). Figure 1(a) illustrates how a multicast message will have to be realized with multiple unicasts in a network that does not support multicasting. VCTM observes that it is much more efficient (in both power and bandwidth) to only replicate the messages at forks in the tree. In this example, a message is thus sent from A to the closest nodes B and C and then those messages are in turn forwarded to the remaining leaves (Figure 1b). Finally, in Figure 1c, it is shown how VCTM maps the logical tree to an underlying mesh network.

Each tree is associated with a fixed tree route from the source to the destinations in the multicast destination set. This tree route is set up upon the first multicast to a set, with setup packets snaking through the network to build up the tree. The tree route information is then stored in virtual circuit tree (VCT) tables at each router along the path. Subsequent multicasts to the same set of sharers reuse this tree route by looking it up in VCT tables. At each intermediate router in the network, the packet follows the predetermined tree route, while network resources (virtual channels [12] and buffers) are granted on a per-hop basis so as to maintain high network bandwidth.

As may be evident from this brief description of VCTM, its performance improvement and power savings are predicated on reuse of the multicast tree, as tree reuse amortizes the setup time of the tree route, and reduces the size of the VCT table needed. This makes a VCTM network particularly suited for VTC: VTC use a multicast tree per coarse-grained region, and reuses this multicast tree for all requests to sharers of this region. In the rest of this paper, we will discuss how Virtual Tree Coherence leverages VCTM to lower its bandwidth overhead, and how VCTM needs to be modified to support the ordering properties required by VTC.

# 3. Virtual Tree Coherence

Most cache coherence protocols rely on an ordering point to maintain correctness. In directory-based protocols, the directories serve as an ordering point for requests. With bus-based protocols, the bus serves as the ordering point for requests. Recently several proposals have leveraged the partial ordering properties of a ring to facilitate cache coherence [24], [31], [32]. Exploiting the partial ordering properties of a ring eliminates some of the disadvantages of using a bus; however, there are some downsides. A ring is a significantly less scalable topology than a mesh or torus. Logically embedding a ring into a mesh topology [32] provides a higher bandwidth communication for data but still suffers from large hop counts and lower bandwidth for ordering requests.

Virtual Tree Coherence observes that ordering can be achieved through structures other than a ring or a bus, in particular, through a tree. Specifically, rather than realizing ordering through a *physical* tree interconnect [16]. Virtual Tree Coherence maintains coherence through *virtual* trees. These trees are embedded into a physical network of arbitrary topology by VCTM, with a virtual tree connecting the sharers of a region. The root of this virtual tree now serves as the ordering point in place of a directory protocol's home node.

Virtual Tree Coherence is a hierarchical protocol. At the first level, snooping is achieved through logical trees. The second level of the protocol, the coarse directories provide the caches with information about which processors must be involved in first level snooping.

In a nutshell, every request is first sent to its virtual tree root node (obtained from the local region tracking structure). This root node orders the requests in order of receipt, then multicasts requests to all sharers of the region through a VCTM virtual tree for that region. The above works for all *current* sharers. *New* sharers, however, need to take a 2-level approach: they have to first go to the directory home node to obtain the root node. This new sharer will then be updated into the local region tracking structure and added incrementally into the virtual tree.

Virtual Tree Coherence results in several benefits. First, the root node can be strategically selected to be one of the sharers to cut down on latency, especially when sharers are physically clustered close-by. In a conventional directory, the directory home node for an address is statically defined given an address, and is thus not necessarily a sharer. Second, latency is low, comparable to a broadcast-based protocol, since we do not need to collect acknowledgments nor wait for a directory lookup; messages that reach the root node begin their tree traversal immediately. Third, bandwidth overhead is low compared to a broadcast-based protocol, as only the current sharers are involved in the multicast, not all nodes.

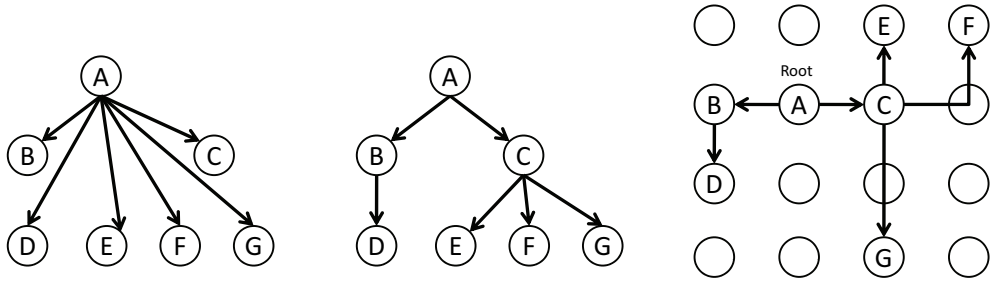
## 3.1. Walk-through Example

TABLE 1
STEPS CORRESPONDING TO FIGURE 2

| | |
|---|---|
| (1) | Both A and B issue requests to the root to modify the same block owned by F with A,B,E,F caching this region, E is the root node for this region |
| (2a)<br>(2b) | E receives B's root request and it becomes ordered.<br>E receives A's root request, it becomes ordered after B |
| (3) | E forwards B's request to all leaf nodes of the region tree<br>A sees B's request has been ordered prior to its own request<br>A knows it will receive data from B after B has completed<br>A must invalidated its existing copy so as not to read stale data |
| (4) | E forwards A's request to all leaf nodes of the region tree<br>B sees its own request forwarded from E. B knows its request has been ordered. B does not need to wait for acknowledgments<br>F sees B's request and responds with the Data |
| (5) | B receives the Data response from F and completes its transition to Modified State. A,B,F see A's ordered request |
| (6) | B invalidates its Modified copy and sends the data to A |
| (7) | A receives the data from B and completes its transition to Modified State |

An example of Virtual Tree Coherence is illustrated in Figure 2 with the corresponding step descriptions presented in Table 1. In this example, Node E is the root of this region's tree; as such, all requests to addresses within that region must be ordered through Node E.

In the example, A and F are initially caching the block in question (A has the block in shared state and F has the block in owned state). Both invalidate their blocks when they see B's request from the root node. This prevents A from reading a stale copy of the block after B has written it.

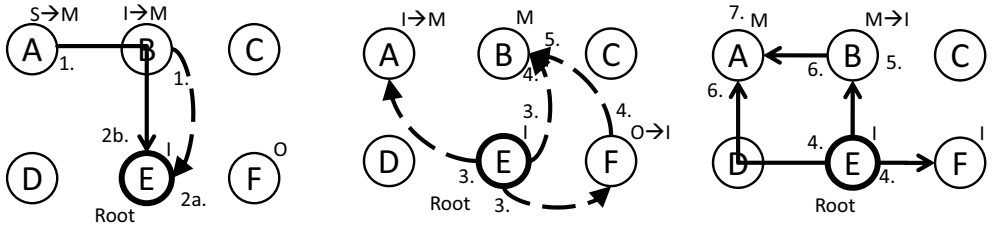



(a) Multicast message from A to B-G realized as multiple unicasts

(b) Logical multicast tree

(c) VCTM: Logical multicast tree mapped to mesh

**Figure 1. VCTM Overview**




(a) Time 1-2

(b) Time 3-5

(c) Time 4-7

**Figure 2. Virtual Tree Coherence Example: This example illustrates two exclusive requesters to the same address in the tree-order protocol. Dashed and curved arrows represent messages originating at or intended for B, solid and straight arrows represent messages originating at or intended for A. E is the root node for the region being accessed.**

Invalidation acknowledgments are unnecessary with VTC for writes to complete since the virtual trees are snoop-based. This is analogous to the lack of acknowledgments in a snooping bus protocol. With VTC, a write can complete when it sees its own request returned from the root node.

Currently we use a first touch assignment policy to determine the root node. When the first system-wide request for a region is made to the directory, the requester will become the root node. The directory stores the identity of the root node as well as the sharing vector for each region block. Other root assignment policies could be employed; however, the impact of these policies as well as root migration is left for future work.

### 3.2. Ordering

Virtual Tree Coherence provides the following ordering invariants:

- Ordering Point: Each region is mapped to a single ordering point so all requests to the same address will go to the same ordering point. This is achieved by assigning a single virtual tree per region, and having the root of that virtual tree as the ordering point. Requests are then unicast to the tree root. This is similar to the use of a directory as an ordering point.
- Sharers observe the same ordering of requests: requests multicast from the root node must arrive at leaf nodes in the same order. Logically, the tree needs to maintain the ordering of a bus: sending a request to the root node of the tree is equivalent to arbitrating for the bus (observing the forwarding of one's request from the root node is equivalent to gaining access to the bus). All requests sent out from the root of the tree will be in a total order. In other words, requests on the same virtual tree must not be reordered by the underlying physical interconnect. This is achieved by modifying VCTM to ensure that each virtual tree is tied to a single virtual channel.
- Cores caching a block must see all coherence requests to that block: A multicast must contain *all current* sharers. Additional non-sharing cores can be included but never fewer. This is achieved with the second-level directory always having a complete list of the sharers. When a non-sharer requests a block, it must first get the sharing list from the directory and be added to the sharing list at that time. Then when the Tree Root multicasts the *new sharer's* request to all sharers, the *current* sharers will add the requester to their sharing list so that region sharing lists at the L2 cache are kept up-to-date.
- Write serialization: Ordering through the root node serializes all writes to the same address region. In Section 4.2, we discuss how that write order is preserved through the network from the root to all leaf nodes. Requests to the same virtual tree maintain a total order. Since write order is maintained from the ordering point to the leaf nodes, invalidation acknowledgments are not necessary.

- Write propagation: A write can complete once it sees its ordered request returned from the root node; this guarantees that any subsequent request to that cache block by any processor will receive the new value written. It is essential that all cores caching the region be involved in the virtual tree; stale values are invalidated when the root forwards the write request to all leaf nodes.

### 3.3. Coherence States, Requests, Actions

Coherence information is maintained at the processor at two granularities. The local last-level cache (L2) maintains coherence state information on a cache block granularity. Coherence information is then maintained by RegionTracker, encompassing multiple contiguous cache blocks. On a per-region granularity, we track which external cores are caching the region and the location of the root node for this region.

CGCT was first proposed for SMP systems; on clean-shared misses a request would go directly to the memory controller rather than waste precious bus bandwidth. Since our work deals with a many-core CMP, we want requests to stay on-chip to save miss latency. Therefore, clean-shared misses are multicast to other cores caching the region. Table 2 gives an overview of the steps taken based on the region state for loads, stores and upgrades. Numbers with multiple parts (e.g. 1a and 1b) indicate actions that occur in parallel.

When a core replaces a region, it notifies the root node for that region to remove it from the sharing list. The root node will then construct a new tree for that region with one fewer leaf node. The other cores caching the region do not need to be notified since only the root is responsible for sending coherence requests to all sharers.

### 3.4. Relationship between trees and regions

To maintain coherence, all cores caching a region must see coherence requests to that region. A single tree is maintained at the root node for that region. Remember, an address maps to *only one* region; so that address participates in a single tree connecting all sharers. All requests use this tree; it is not possible for a single region to map to multiple trees. If that is allowed, it would mean that different cores had different sharing lists for that region and incoherence would result. Multiple regions can map to the same tree; this simply means that multiple regions are being shared by the same set of processors with the same root node. If all cores are caching a region (for example, a lock variable), a single tree will be constructed at the root node with all cores as leaf nodes.

## 4. Implementation

The following section details issues related to the hardware implementation of Virtual Tree Coherence. Specifically, we first present high-level changes to the system architecture. Next we discuss the requirements placed on the underlying interconnection network to preserve coherence ordering. Finally, we discuss the area overheads of additional structures.

### 4.1. High level Architecture

Figure 3 presents a high level diagram of the architecture we are proposing. Each node consists of a core, private L1 instruction and data caches. Modifications are made starting at the private L2 cache; the key structures for this architecture are highlighted with bold borders. A RegionTracker sits alongside the Level 2 cache and maintains sharing lists for currently cached address regions; this structure also encompasses the functionality of the Level 2 tags, thus resulting in very small area overhead. The directory at each node is coarse grained and also contains sharing lists but is decoupled from the private L2 cache as regions are distributed across all the directories in the system. Upon a miss in the RegionTracker, a request has to go to the directory to obtain the current sharing lists, thereby refreshing the RegionTracker. Entries are evicted from the RegionTracker in an LRU fashion. The packet-switched router has been modified to include VCTM extensions, in particular the virtual circuit tree (VCT) table. For details about the VCT overheads, we refer the readers to [14].

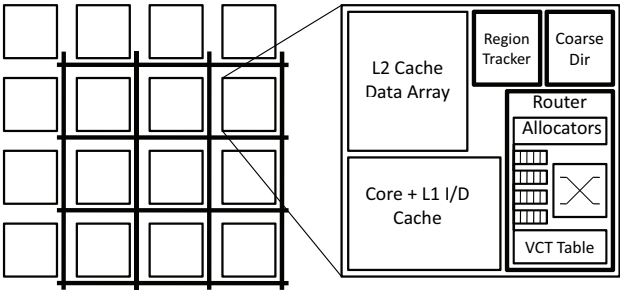



**Figure 3. Hardware Architecture Diagram**

### 4.2. Network design

**Virtual tree construction.** VCTM maintains a small content addressable memory (CAM) of currently active trees at the network interface of each node. RegionTracker provides the destination set required for the coherence request to the network interface controller which then searches the CAM to determine if there exists an active tree for this destination set. If a tree exists, the CAM returns the Virtual Circuit Tree Id which is used to index into the VCT lookup tables at each router. If no match is found in the CAM, a new tree must be setup; this is done with low overhead, per the VCTM mechanisms.

When a sharer has been added to a region, this will result in a new destination set being provided to the network interface controller. This will likely trigger a new tree setup unless that destination set is being actively used by another region. Region sharing lists are decoupled from Virtual Circuit Trees; multiple regions can map to a single tree.

**Preserving order in the network.** We modify VCTM to restrict virtual channel allocation so as to maintain the ordering invariants associated with a tree. Specifically, requests must not be reordered from the time they depart the root node to the time they arrive at each leaf node. An unrestricted virtual channel allocation policy would permit such a reordering, leading to a leaf node seeing a different order than other leaves. As such, each region is assigned a single virtual channel. All requests for that region must use the same virtual channel and cannot change virtual channels at intermediate nodes in the network. It should be noted that here, ordering only needs to be maintained within a single virtual tree, not across the entire network. Hence, different regions and different trees can be assigned to different virtual channels, which will avoid degrading the network performance.

TABLE 2
VIRTUAL TREE COHERENCE

| Region State | Cache Miss | VTC Coherence Actions |
|---|---|---|
| Invalid: no information about remote copies of region | Load/Store | 1. Request Region Destination Set Information from Directory<br>2. Directory responds with region sharing list<br>3. Region state set to Exclusive/Modified if sharing list is null<br>else Region State is set to shared<br>4. Load/Store actions performed according to steps below |
| Shared: Other cores are caching the region (may be clean/dirty) | Load | 1a. Send Read Request to Root Node<br>1b. Request data from memory: speculative memory request (partially overlap mem. latency w/ ordering)<br>2. Request is ordered by Root Node and forwarded to region sharers<br>3. Observe own request - ordered w.r.t. other requests to this address<br>4. Multicast sharers caching data, respond to Read Request with Data<br>5. If data not on chip, wait for memory response. |
| | Store | 1a. Send Store Request to Root Node<br>1b. Request data from memory: speculative memory request<br>2. Request is ordered by Root Node and forwarded to region sharers<br>3. Observe own request<br>4. Region sharers caching data, response to Store Req. w/ Data and invalidate own copy<br>5. Receive data from multicast sharer or wait for memory response if not cached on chip<br>Once observed own request and received cache line, it is safe to perform store |
| | Upgrade | 1. Sent Upgrade Request to Root Node<br>2. Root Node Forward Upgrade to all sharers<br>3. Region sharers caching data observe upgrade request and invalidate cache block<br>4. If Observe other store/upgrade request, another request ordered before own<br>Invalidate cache line, now request that was ordered prior to mine will supply fresh data<br>5. else if Observe own request, Upgrade complete |
| Exclusive or Modified: No other core caching region | Load | 1. No other cores caching region - request does not need to be ordered<br>2. Send Read Request to Memory |
| | Store | 1. No other cores caching region - request does not need to be ordered<br>2. Send Store Request to Memory |
| | Upgrade | 1. Can upgrade without sending message |
| Replace | – | 1. Invalidate all copies cached in region<br>2a. Send Region Invalidate Acknowledgment to Directory<br>2b. Notify root of invalidation<br>3. Directory removes sharer from sharing list for region |

**Scalability**. As the number of nodes in the network grows, VCTM requires additional storage for multicast trees. Each node in the network is allocated a portion of the Virtual Circuit Tree tables located in each router. If 64 trees are given to each core, in a sixteen node system, this will result in 1024 entries in each VCT table. With 64 cores in the system, this grows to 4096 entries. (Note: the size of each VCT entry remains the same, with Destination Set CAM entries widened to account added destinations). As the system grows, so do the number of unique trees; however, server consolidation workloads will not realistically access a large percentage of trees. To better scale the VCTM mechanism, we propose replacing the original Destination CAM (used to track active trees at the network interface controller) with a Ternary CAM (TCAM). A TCAM allows us to collapse similar trees by specifying "don't care" bits in the search for an active tree. This will allow us to *hit* on an active tree that includes all necessary destinations for our request with some extra *unnecessary* destinations included in the tree. To prevent the use of tree collapsing from driving up bandwidth, we restrict the maximum size of a similar tree to be within 2 links of the original tree. As a result, 8 trees per core can achieve the same performance and hit rates as 64 trees per core, greatly reducing the storage requirements at each router. Additionally trees with many destinations can map to a single broadcast tree as well.

### 4.3. Storage overheads

The storage overheads of Virtual Tree Coherence stem largely from three components: (1) the RegionTracker structure in each core, (2) the second-level coarse-grained directory cache in each core, both adding to the storage overhead, but (3) as RegionTracker obviates the need for L2 tags, L2 tag array storage overhead is saved. Additionally, the second-level coarse-grained directory cache replaces the fine-grained directory cache required for the baseline directory protocol. The sizes of these three components clearly depend on the size of a region, $R$.

To arrive at $R$, we perform a characterization study across our suite of benchmarks. In Figure 4, we present the sharing patterns for a variety of scientific and commercial workloads based on various region sizes. Similar to what has been previously observed [7], [19], the number of sharers for a cache block (cache block size = 64B) is small. As the region size increases (16 and 64 cache lines), the number of sharers increases slightly; this increase is a result of false sharing. Sending multicasts to an increased number of cores will utilize additional bandwidth but at a substantial area savings for tracking this information. Previous work leveraging regions has used a 1KB region size; similarly, we believe this is a good trade-off between area overhead and multicast sharing information.

The original RegionTracker proposal consumes area comparable to a conventional L2 tag array assuming 1KB regions and an 8MB data array. We've added additional bits of information, specifically 16 bits to track the multicast sharing vector and 4 bits to track the multicast root node, for 16 cores. The region array size per core is determined by Equation 1 in terms of $N$, number of cores and region geometry (e.g. $RSets$, number of sets in Region, $RegSize$, size of each region, $RWays$, the associativity of the region array). Each entry contains the Region Tag, 3 bits of state, $N$ bits for the multicast sharing

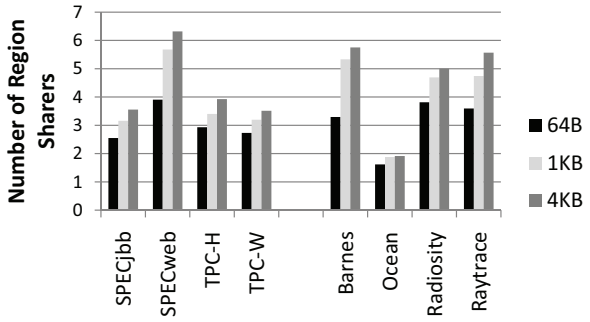



**Figure 4. Characterization of Sharers by Region Size**

vector, $log_2(N)$ bits to identify the root and 4 bits of state per cache line ($validbit + way$) in the region.

$$RegionArraySize = (Tag + 3 + N + log_2(N) + (RegSize/CacheLineSize) \times 4) \times RSets \times RWays \quad (1)$$

For example, assuming a 50 bit address, 1KB regions, 1024 region sets, and 8 region ways, we find the region array size to be 113 KB. Our simulation parameters assume a L2 cache of 1MB; where the size of conventional cache tags would be 74 KB.

The second level directories also consume area; however, compared to a conventional fine-grained directory, coarse-grained directories have significantly greater memory reach which improves performance by reducing the directory miss rate. Computing the directory size (in bits) is done using Equation 2, assuming each directory entry contains $N$ bits for the sharing vector and $log_2(N)$ bits to identify the owner node (in a conventional directory) or $log_2(N)$ bits to identify the tree root with VTC. Directory sizes assuming 1024 directory sets and 16 directory ways are presented in Table 3.

$$DSize = (Tag + N + log_2(N) + 3) \times DirSets \times DirWays \quad (2)$$

Bit requirements for the baseline configuration (a conventional L2 + fine-grained directory cache) are compared against the RegionTracker + coarse-grained directories in Table 3, with the parameters detailed above. We see VTC has a 21% storage overhead over the baseline directory-based protocol.

However, this storage overhead can be tuned by reducing the size of the coarse-grained directories, since a single entry in the coarse-grained directory covers 16 times more memory than a corresponding entry in a fine-grained directory, with a region size that is 16 times a cache line size. So, we can reduce the coarse-grained storage to trade off the storage overhead of RegionTrackers, while still being able to cache and cover more memory than conventional fine-grained directories. Setting the number of bits of the *fine-grained directory + conventional L2 tag array = coarse-grained directory + RegionTracker*, we have enough bits for a smaller coarse grain directory composed of 690 sets and 16 ways. With 690 sets and 16 ways, the small (area-equivalent) coarse-grained directory can cache 11MB of memory versus 1MB of memory that is cacheable with the fine-grained directory.

Using the same number of sets and ways in both the fine- and coarse-grained directories, the *coarse-grained directory + RegionTracker* configuration consumes 21% more bits. More geometries could be explored to find additional area and memory-reach trade-offs.

TABLE 3
STORAGE COMPARISONS IN KBITS

| | |
|---|---|
| **Conventional Directory-Based Protocol** | |
| Conventional L2 Tag Array | 592 |
| Fine-Grained Directory | 896 |
| **Virtual Tree Coherence** | |
| RegionTracker | 904 |
| Coarse-Grained Directory | 864 |
| Area-equivalent Coarse-Grained Directory | 582 |

(a) Storage Breakdown

| | |
|---|---|
| Conventional L2 Tag Array + Fine-Grained Directory | 1488 |
| RegionTracker + Coarse-Grained Directory | 1768 |
| RegionTracker + Area-equivalent Coarse-Grained Directory | 1486 |

(b) KBit Totals

**4.3.1 Scalability.** In Section 4.2, we discussed a TCAM technique to improve the scalability of VCTM for larger systems. For other added hardware discussed in this section, we consider scalability issues as the number of cores in the system grows. First, we expect that much larger regions will provide benefit for systems running server consolidation workloads. Coarse address regions will include more false sharing but will still keep virtual machines isolated from one another providing performance benefits and scalability.

A concentrated mesh (CMESH) [5] has been proposed for large systems. A CMESH groups four cores to one single router; so a 64-core system would require a 4x4 mesh. This clustering can be applied to our storage structures as well; 4 cores can share a region array, last level caches and a coarse grain directory to reduce the amount of required storage. A CMESH will also reduce pressure on the VCTM hardware; multicasts can be routed to network nodes and then broadcast to the 4 tiles connected to each router.

## 5. Evaluation

In the following sections, we present our evaluation methodology as well as detailed information regarding the baseline systems, followed by results.

### 5.1. Methodology

We use PHARMsim, a full system multiprocessor simulator [8], [18] built on SIMOS-PPC. Included in our simulation infrastructure is a cycle-accurate network model including pipelined routers, buffers, virtual channels and allocators for both the baseline packet-switched mesh and the routers augmented with VCTM. Our simulation parameters are given in Table 4. Results are presented for the following commercial workloads: TPC-H and TPC-W [33], SPECweb99 and SPECjbb2000 [30] and several Splash2 workloads [36]. Details for each workload are presented in Table 5. We compare Virtual Tree Coherence against two baselines, a directory protocol and a greedy-order protocol which are explained below. Statistical simulation is used to quantify overall performance with 95% confidence intervals [4].

TABLE 4
SIMULATION PARAMETERS

| | |
|---|---|
| Cores | 16 in-order & 64 in-order cores |
| **Memory System** | |
| L1 I/D Caches (lat) | 32 KB 2 way set assoc. (1 cycle) |
| Private L2 Caches | 1 MB (16 MB total) 8-way set assoc. (6 cycles), 64 Byte lines |
| RegionTracker (associated with each L2) | 1024 sets, 8 ways, 1KB regions |
| Memory Latency | 500 cycles |
| **Interconnect** | |
| Packet Switched Mesh | 3 Pipeline Stages<br>8 VCs w/4 Buffers per VC |
| VCTM | 64 Trees per source node (1024 total) |

We have configured our simulation environment to support server consolidation workloads [15], [25] for up to 64 cores. For the server consolidation workloads, we create homogeneous combinations of each of the commercial workloads listed in Table 5; e.g. we run 4 copies of SPECjbb to create a 64-core workload. Each virtual machine is scheduled to maintain affinity among the threads of its workload.

TABLE 5
BENCHMARK DESCRIPTIONS

| Bench. | Description |
|---|---|
| SPECjbb | Standard java server workload utilizing 24 warehouses, executing 200 requests |
| SPECweb | Zeus Web Server 3.3.7 servicing 300 HTTP requests |
| TPC-W | TPC's Web e-commerce benchmark, DB Tier |
| TPC-H | TPC's Decision Support System Benchmark, IBM DB2 v6.1 running query 12 w/ 512MB database, 1GB of memory |
| Barnes | 8K particles, full end-to-end run including initialization |
| Ocean | 514x514 full end-to-end run (parallel phase only) |
| Radiosity | -room -batch -ae 5000 -en .050 -bf .10 (parallel phase only) |
| Raytrace | car input (parallel phase only) |

**5.1.1 BaselineI: Directory-based Coherence.** The first baseline we evaluate VTC against is a standard Directory protocol modeled after the SGI-Origin protocol [17]. This protocol suffers from the latency overheads associated with an indirection through a directory on each cache miss. Additionally, to make this protocol amenable for a many-core architecture, directory caches must be used. Misses to these directory caches suffer the latency overhead of going off-chip to memory and can be quite frequent for server workloads and even more frequent for server consolidation workloads. For a set of commercial workloads, miss rates between 22 and 74% have been observed [22].

**5.1.2 BaselineII: Greedy order region coherence.** Greedy order protocols have been proposed for ring interconnects [6], [24], [29] and overlaid atop unordered interconnects [32]. Here, as a second baseline, we map and optimize a greedy order protocol that can leverage the region tracking structures and multicast network that VTC uses. The key difference is that VTC relies on the virtual tree for ordering, while greedy order does not.

In greedy order protocols, requests are ordered by the current owner. Requests are live as soon as they leave the requester; in other words, they do not need to arbitrate for a shared resource such as a bus or pass through a central ordering point such as a directory. A request becomes ordered when it reaches the owner of the cache block (another cache or memory). In the common case when no race occurs, these requests are serviced very quickly because they do not require the additional latency of an indirection through a directory. In [24], requests complete after they have observed the combined snoop response that trails the request on a ring. Based on the combined response, a request is successful or must retry.

TABLE 6
STEPS CORRESPONDING TO FIGURE 5

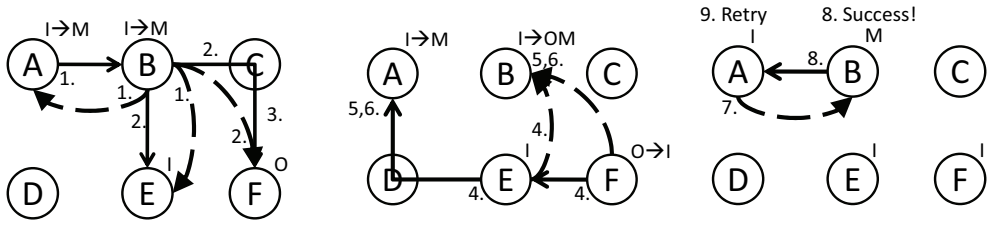
| | |
|---|---|
| (1) | Both A and B issue requests to all processors caching region to modify a block owned by F with A,B,E,F caching this region |
| (2) | A's request reaches B and is replicated and forwarded to E and F. B's request reaches E and F |
| (3) | A's request reaches F<br>B's request reached F (Owner) first, so B's request will win |
| (4) | E and F respond with acknowledgments to A and B's request.<br>B gets an owner acknowledgment from F<br>F transitions from owned to invalid |
| (5) | A receives E's acknowledgment, B receives E's acknowledgment |
| (6) | A receives F's acknowledgment, B receives F's owner acknowledgment |
| (7) | B knows its Modified request will succeed, it sends a negative acknowledgment to A |
| (8) | A receives a negative acknowledgment from B, it has now collected all acknowledgments and did not succeed so it will acknowledge B's request and it must retry its own request |
| (9) | B collects its final acknowledgment from A and successfully transitions to Modified State. |

Mapping and extending Greedy-Order protocols onto an unordered interconnect such as a mesh requires acknowledgments to be collected from all relevant processors. Also, extending to regions rather than individual cache lines mean that acknowledgments have to be collected only from cores that are caching that particular address region rather than all cores. So, the number of acknowledgments that are expected is derived from the sharing vector in the region cache. The collection of acknowledgments is similar to the combined response on the ring but requires more messages. This is also similar to the process of collecting invalidates in a directory protocol. The owner sends an *owner acknowledgment* signifying the transfer of ownership. If no *owner acknowledgment* is received, then another request was ordered before this one and this request must retry. Greedy-Order can be applied in a broadcast fashion as well, where no sharers are tracked and acknowledgments are gathered from every processor. An example of Greedy-Order is depicted in Figure 5 and walked through in Table 6.

## 5.2. Results

In the following sections, we present quantitative performance results comparing Virtual Tree Coherence against our two baselines. Additionally, we present a comparison between Virtual Tree-Multicast (VT-M) and Virtual Tree-Broadcast (VT-B). With VT-B, a virtual tree connects all nodes; however, regions are used to designate the root node so that there is not a single root bottleneck. Significant network bandwidth and dynamic power can be saved by limiting coherence actions to multicasts instead of broadcasts.

In Figure 6, results are presented for Greedy-Order Multicasting, Virtual Tree Broadcast and Virtual Tree Multicast Coherence. All results are normalized to BaselineI: Directory




(a) Time 1-3 (b) Time 4-6 (c) Time 7-8

**Figure 5. Greedy Order Example: This example illustrates two exclusive requesters in the greedy-order region protocol. Dashed and curved arrows represent messages originating at or intended for B, Solid and straight arrows represent messages originating at or intended for A. Time is progressing from left to right in the figure.**

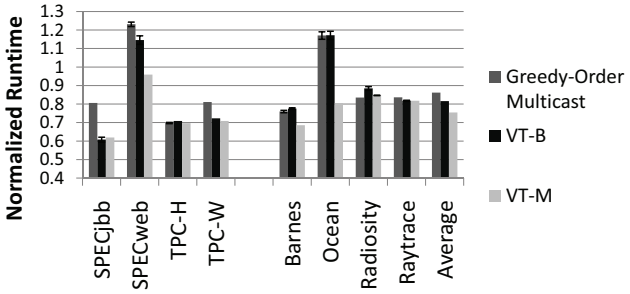
Coherence. Overall, significant performance gains are achieved by VT-B and VT-M, up to 39% and 38% respectively (19% and 25% on average) when compared to the directory protocol. Virtual Tree Coherence outperforms Greedy-Order by up to 31% with an average improvement of 11%. In a 4x4 system, the differences between VT-B and VT-M are minor; however, as systems scale, the difference between these two become much more pronounced with favorable results for VT-M (Figure 6b).

In a couple of instances, notably, TPC-H and SPECjbb, VT-B outperforms VT-M. With larger memory footprints, and irregular access patterns, these workloads experience much larger region miss rates which incur additional overhead to re-fetch region information from the second-level directory. For SPECjbb and TPC-H, 21% and 18% of L2 misses also result in region misses; the rest of the workloads experience region miss rates of less than 10%. SPECweb sees only a small performance improvement from VTC (5%); SPECweb sees the sharpest increase in traffic which limits the performance improvement. Techniques to improve the region hit rate and lower the number of false-sharers will lead to performance improvements for SPECweb.

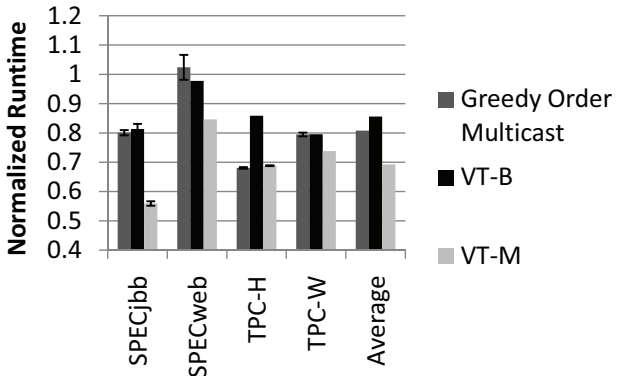
In Figure 6b, the difference between VT-B and VT-M becomes more pronounced, VT-M outperforms VT-B by an average of 11% and up to 16%. With a 64-core system, broadcasting becomes more expensive (both in performance and power). VT-M provides more isolation for the virtual machines; coherence requests are only sent to nodes involved in sharing. VT-B sends broadcasts to all nodes (across multiple virtual machines).

In Figure 6, Greedy-Order, VT-B and VT-M, all leverage the benefits of VCTM. On a non-VCTM packet-switched mesh, the peformance of VT-M degrades by an average of 15%. Greedy-Order which places significant pressure on the interconnect due to retries sees performance degradations of 48% when VCTM is removed. VCTM simplifies ordering of coherence requests in the network and is essential for performance improvements and scalability. Without VCTM, in a 64-core system, VT-B sends out 63 coherence packets for each cache miss which would likely saturate the network.

Both VTC and directory protocols require the indirection to the ordering point for coherence requests. VTC derives performance benefit in part from reducing the cost of these indirections. With VTC, the hop count to the ordering point is reduced by 15% for 16-core and 50% for 64-core since the root node is a region sharer. Furthermore, on average $4.2x$ more coherence requests are ordered in zero hops with VTC than with the directory protocol.




(a) Performance of single workloads on 16 cores




(b) Performance of 64-core server consolidation workloads

**Figure 6. Performance of Coherence Protocols.**

The interplay between region size and the efficiency of a VCTM network is an interesting motivation for the need to co-design the coherence protocol and the interconnect. Choosing small regions results in a much larger number of unique trees that are needed; this large number of unique trees causes the virtual circuit trees to thrash in the network. The virtual circuit tree hit rate in the interconnect ranges from 78% to 99% for 1KB regions; the hit rate drops to 65% to 95% for 64B

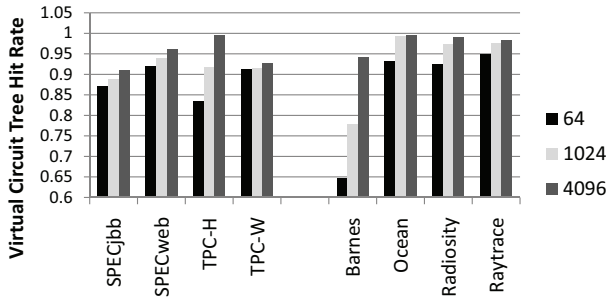



**Figure 7. Impact of Region Size on Virtual Circuit Tree Hit Rate (16 cores)**

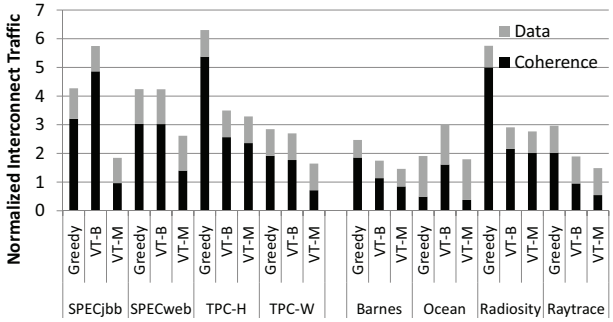



**Figure 8. Interconnect Traffic Comparison Normalized to Directory for 16 cores (measured in link traversals by flits)**

regions as depicted in Figure 7. With a lower hit rate there are more tree setups in the network; recall from the original VCTM proposal, tree setup requires replication of the multicast message into many unicasts resulting in a short burst of traffic during the setup phase which impacts interconnect latency and throughput. As a result, 1KB region which suffers from a modest amount of false sharing compared to 64B regions, actually has 3% less interconnect traffic by making better use of virtual circuit trees. With 4KB regions, performance is similar to 1KB regions; with slight degradations observed for SPECweb and Ocean.

Restricting a virtual tree to a single virtual channel is neccesary to maintain in network order; however with a large number of available trees, we are able to use network bandwidth efficiently and only see a degradation of 3% in end-to-end network latency (versus an unrestricted virtual channel allocation).

**5.2.1 Activity Comparisons.** Figure 8 shows the network activity (based on link traversals by flits) for each coherence protocol relative to directory coherence. Of course, directory coherence has the lowest interconnect traffic since nearly all of the messages are of a unicast (point to point) nature (invalidation requests from the directory are the exception). Data traffic is similar for each protocol; the main difference lies in the required coherence traffic. Greedy-Order requires the most interconnect bandwidth of all the protocols, averaging $3.8x$ the number of link traversals as the directory protocol. VT-M consumes less network bandwidth than VT-B, 35% usage reduction on average, with the most significant reductions of 68% for SPECjbb and 40% for Ocean. A large fraction of Ocean's references are memory misses; VT-M will optimize and go directly to memory if no other cores are caching the region. These memory misses are broadcast to all cores in VT-B resulting in a bandwidth spike when compared to VT-M and Greedy-Order. The interconnect traffic difference between VT-B and VT-M grows from 35% with 16 cores to 68% with 64 cores. VT-M requires $1.6x$ more traffic than a directory protocol for 64 cores.

Network activity is only part of the story for power consumption differences in VT-B and VT-M. VT-B will consume significantly more power since all caches will snoop all coherence requests; VT-M eliminates a significant fraction of cache accesses required with VT-B. The retries in Greedy-Order also increase the number of cache accesses required.

## 5.3. Discussion

In the previous sections, we illustrated that VTC could provide substantial performance improvements over a directory-based protocol. VTC Broadcast performs well without the need to track multicast sharing groups; however the broadcast protocol places heavy demands on the interconnection network and consumes significant unnecessary power limiting its ability to scale.

VTC largely out-performs Greedy-Order, but there are a few benchmarks where Greedy-Order does slightly better. The key downside to Greedy-Order protocols is that they can have an unbounded number of retries when certain race conditions occur (which can lead to starvation). In practice, we found the number of retries to be small (fewer than 10 retries per 1000 L2 misses) with a few exceptions; Radiosity and TPC-H exhibit large numbers of retries, 696 and 327 for 1000 L2 misses which accounts for the large spike in traffic for Greedy-Order. An additional downside of a Greedy-Order protocol on a mesh as compared to its use on a ring, is the additional messages required. With no combined snoop response, there is a significant increase in acknowledgment messages placing high bandwidth demands on the interconnect.

In addition to the problem of retries, the Greedy-Order protocol makes no use of the implicit ordering properties of the underlying network functionality. Greedy-Order only utilizes the virtual multicast trees to overcome the overhead associated with multicasting on a mesh network. Greedy-Order also places substantially more pressure on the VCTM hardware; with VTC, each region has at most one tree. With Greedy-Order, each region sharer creates its own tree resulting in a forest of trees for widely shared variables.

In Section 1, we posited 4 key features of a scalable coherence protocol for many-core architectures. Here we examine how well VTC achieves these goals.

- **Limit coherence actions to the necessary subset of nodes:** Coherence requests are seen by an average of 4 cores with 1KB region sizes. Increasing the region size trades off storage overhead at the expense of more unnecessary cores becoming involved in coherence requests. For example, with a region size of 4KB, on average 10% more cores see a coherence requests.
- **Fast cache-to-cache transfers:** The ordering point, the virtual tree root is located at one of the region sharers reducing the latency of ordering a request. Virtual Tree

Coherence outperforms Directory Coherence by 25% on average, a direct result of fast cache-to-cache transfers. VTC out-performs Greedy-Order by an average of 11% by avoiding retries.

- **Limited bandwidth overhead:** VTC reduces on-chip bandwidth utilization by an average of 35% over a broadcast protocol and 45% over a greedy protocol. We increase on-chip bandwidth utilization by a factor of $2.1x$ over a directory protocol. Only in one instance (SPECweb) did this bandwidth increase result in limited performance improvement.
- **Limited storage overhead:** In our modified system *RegionTracker + Coarse-Grain Directories* consume 21% more area than the *baseline L2 tags + Fine-Grain Directories*. However, with this increase in area comes significantly more reach for our directories. The coarse directories are able to touch $16x$ more memory with 1KB regions and the same number of directory entries. Additionally, with a fixed area budget (baseline: L2 Tags + Fine-Grain Directories, VTC: RegionTracker + Coarse-Grain Directories), the Coarse Grain Directories have a 4% lower directory cache miss rate than the Fine-Grain Directories. Considerable performance improvement is possible with VTC without incurring large area overheads.
- **Scalability:** With a 64-core system, VTC outperforms both a directory protocol and VT-B by an average of 31% and 15% respectively.

## 6. Related Work

Cache coherence research has been of significant interest in both single and multi-chip multiprocessor systems. A variety of protocols have been proposed/implemented to achieve performance and scalability on both ordered and unordered interconnects. We contrast these prior works with Virtual Tree Coherence in the following sections.

### 6.1. Ordered Interconnect

Multicast snooping and destination set prediction [7], [19] use prediction mechanisms to determine which processors will likely need to see a coherence request. In contrast, our work determines exactly who must be included in a multicast. Extra cores might be contained in the destination set but never fewer cores than necessary. These protocols rely on a totally ordered interconnect for sending out multicast requests. Our design relaxes this constraint, permitting a higher performance interconnect. Requests to the same address region use the same virtual tree and are restricted to using the same virtual channel as prior requests to the same address region. This virtual channel restriction prevents messages from becoming reordered with respect to each other in the network.

Bandwidth Adaptive Snooping [21] employs a hybrid protocol that achieves the latency of broadcasting when bandwidth is plentiful but converts to a directory style protocol when bandwidth is limited. This work relies on a totally ordered interconnect but overcomes some of the pressure that large snooping systems can place on the interconnect.

### 6.2. Unordered Interconnect

Token coherence provides the token abstraction to decouple performance from correctness [20]. Several variants of Token Coherence have been proposed including one based on broadcasts and one on directories. TokenB, the broadcast Token protocol requires more bandwidth than multicasting with Virtual Tree Coherence. Extensions have been proposed for multi-chip CMP based systems in [23].

Virtual hierarchies [25] propose cache coherence variations targeting server consolidation workloads running on chip multiprocessors. One proposal utilizes two levels of directories to provide fast local coherence and correct (and substantially slower) global coherence (with the observation that global coherence is rare). The other proposal still utilizes local directories for fast coherence within a server application and a backing broadcast protocol for global coherence. The alternative of utilizing a local broadcast backed by a global directory protocol is mentioned but not explored. The coherence mechanism in this work is similar to the latter case. First of all, we examine a different hierarchy than what is proposed in Virtual Hierarchies. As such, we do not provide a direct quantitative comparison; however, we do provide some points that further distinguish our proposal.

Some of the performance improvements of Virtual Hierarchies are predicated on the ability of the scheduler to provide locality between communicating and sharing cores or threads. Virtual Hierarchies will work when locality is not preserved; however, we believe that the coupling of multicast coherence with a fast multicast substrate (VCTM) results in superior performance. Virtual Tree Coherence will support flexible placement and scheduling of communicating threads, whereas the benefits achieved with Virtual Hierarchies are predicated on physical proximity.

The actions of the second level directories in Virtual Tree Coherence are very simple unlike directories in other hierarchical protocols. Virtual Hierarchies requires a very large number of states and transitions in the coherence protocol to accommodate two levels of directories; this is not the case for Virtual Tree Coherence. The directories contains the sharing list of each region that is cached anywhere on chip, the identity of the tree root for that region, and whether a block is owned on-chip or if memory is the owner.

In-Network Cache Coherence [13] replaces directories by embedding sharing information in tree structures within the network. These virtual trees (different from VTC's virtual circuit trees) are used to locate data on-chip. When a request is en-route to the directory, it can bump into a tree which will redirect the request to the appropriate core that is sharing the cache block. This optimization targets the directory indirection latency and can lead to fewer interconnect hops to find a valid cache line.

However, with In-Network Coherence, depending on the route taken by a request, a sharer may be nearby, but the request may miss it and still have to make its way the whole stretch to the directory. In such scenarios, VTC will perform better. Cache misses that do bump into a tree in In-Network Coherence will be satisfied more quickly than requests that have to travel significant distance to the root node in VTC. It should be noted though that in VTC, it is always the case that the root node is a sharer of the region, which may be closer than the statically-mapped directory node in In-Network Coherence. Also, VTC utilizes coarse-grained tracking which requires less storage overhead than the per-line,

per-hop storage needed in In-Network Coherence.

UnCorq [32] broadcasts coherence requests on an unordered interconnect (e.g. a mesh) and then orders snoop responses via a logical ring. Similarly, we utilize the ordering implied by logical trees to maintain coherence; however, we order requests via virtual trees rather than responses. An additional difference is the use of multicasting instead of the full broadcast used by UnCorq. Greedy-Order bears some similarity to UnCorq; requests are sent to sharers quickly without regard to order. UnCorq then orders responses via a logical ring, whereas Greedy-Order uses the owner to order requests.

Trees have also been leveraged in previous proposals to build more scalable directories for large distributed shared memory machines [11], [28]. These trees are used to reduce storage overhead but the directories still serve as the ordering point.

### 6.3. Network Designs Cognizant of Cache Coherence

In addition to relevant work in the domain of cache coherence protocols, VTC also examines the requirements that are placed on the interconnect to maintain correctness. These requirements have also been examined in prior work.

The Rotary Router [1], [2] provides mechanisms to maintain the ordering of coherence requests in the network and prevent coherence deadlock within the interconnect. Buffer resource allocation is divided up between dependent messages to prevent dependent messages from deadlocking in the network due to unavailable resources. In-order delivery is guaranteed by forcing ordered messages to traverse the same path.

Another common solution to avoiding deadlock and preventing message reordering is to dedicate a separate virtual network to each class of coherence message (e.g. requests vs responses), where each virtual network has distinct virtual channels. This technique is employed by the Alpha 21364 [27]. VTC forces the underlying network to deliver coherence requests in order by restricting a virtual tree to use a single virtual channel.

## 7. Conclusion

This work proposes a new coherence protocol for many-core architectures, Virtual Tree Coherence. Unordered interconnection network topologies such as a mesh or a torus can be overlaid with an ordering invariant to more easily facilitate cache coherence mechanisms. We utilize one such network, Virtual Circuit Tree Multicasting to realize Virtual Tree Coherence as a scalable on-chip cache coherence solution that improves performance by an average of 25% over a directory-based protocol. By relying on coarse-grained region coherence state, we reduce the on-chip storage overhead for coherence state substantially, without the expected negative side effects of false sharing and coherence thrashing. Instead, we employ efficient on-chip multicasting to reach all nodes in the sharing set, and maintain a total order of messages to the same region by restricting each tree to single virtual channel. Virtual Tree Coherence is simple in concept and implementation since it relies on a straightforward ordering invariant based on a logical tree. In summary, we have extended the fruitful space of region-based optimizations to include a scalable multicasting protocol. Extending Virtual Tree Coherence to the domain of server consolidation workloads [15], where sharing is generally limited to within a virtual machine and to subsets of cores within that VM results in even more substantial benefits with an average improvement of 31%.

## References


[1] P. Abad, V. Puente, and J. Gregorio, "Reducing the interconnection network cost of chip multiprocessors," in *NOCS*, 2008.

[2] P. Abad, V. Puente, J. Gregorio, and P. Prieto, "Rotary router: an efficient architecture for cmp interconnection networks," in *ISCA*, 2007.

[3] N. Aggarwal, J. Cantin, M. Lipasti, and J. E. Smith, "Power-Aware DRAM Speculation," in *HPCA-12*, 2008.

[4] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *Proceedings of HPCA-9*, 2003.

[5] J. Balfour and W. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *International Conference on Supercomputing*, 2006.

[6] L. A. Barroso and M. Dubois, "The performance of cache-coherent ring-based multiprocessors," in *ISCA-20*, 1993.

[7] E. Bilir, R. Dickson, Y. Hu, M. Plakal, D. Sorin, M. Hill, and D. Wood, "Multicast snooping: A new coherence method using a multicast address network," in *Proc. of ISCA*, May 1999.

[8] H. Cain, K. Lepak, B. Schwarz, and M. H. Lipasti, "Precise and accurate processor simulation," in *Workshop On Computer Architecture Evaluation using Commercial Workloads*, 2002.

[9] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Improving multiprocessor performance with coarse-grain coherence tracking," in *ISCA-32*, 2005.

[10] ——, "Stealth prefetching," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[11] Y. Chang and L. N. Bhuyan, "An efficient tree cache coherence protocol for distributed shared memory multiprocessors," *IEEE Transactions on Computers*, vol. 48, no. 3, 1998.

[12] W. J. Dally, "Virtual-channel flow control," in *ISCA*, 1990.

[13] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *International Symposium on Microarchitecture*, 2006.

[14] N. Enright Jerger, L.-S. Peh, and M. H. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *Proceedings of ISCA-35*, 2008.

[15] N. Enright Jerger, D. Vanatrease, and M. Lipasti, "An evaluation of server consolidation workloads for multi-core designs," in *IISWC*, 2007.

[16] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren, "Architecture and design of AlphaServer GS320," in *Architectural Support for Programming Languages and Operating Systems*, 2000.

[17] J. Laudon and D. Lenoski, "The SGI Origin: a ccNUMA highly scalable server," in *ISCA-24*, 1997.

[18] K. M. Lepak, H. W. Cain, and M. H. Lipasti, "Redeeming IPC as a performance metric for multithreaded programs," in *Proceeding of 12th PACT*, 2003, pp. 232–243.

[19] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood, "Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors," in *Proceedings of the 30th ISCA*, June 2003.

[20] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in *ISCA-30*, 2003.

[21] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood, "Bandwidth adaptive snooping," in *HPCA-8*, 2002.

[22] M. R. Marty, "Cache coherence techniques for multicore processors," in *PhD Dissertation, University of Wisconsin - Madison*, 2008.

[23] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood, "Improving multiple-cmp systems using token coherenece," in *HPCA*, February 2005.

[24] M. R. Marty and M. D. Hill, "Coherence ordering for ring-based chip multiprocessors," in *MICRO-39*, December 2006.

[25] ——, "Virtual hierarchies to support server consolidation," in *ISCA-34*, 2007.

[26] A. Moshovos, "Regionscout: Exploiting coarse grain sharing in snoop-based coherence." in *ISCA-32*, 2005.

[27] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 network architecture," *IEEE Micro*, vol. 22, no. 1, pp. 26–35, 2002.

[28] H. Nilsson and P. Stenstrom, "The scalable tree protocol - a cache coherence approach for large-scale multiprocessors," in *IPDPS*, 1992.

[29] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner, "Power5 system microarchitecture," *IBM Journal of Research and Development*, vol. 49, no. 4, 2005.

[30] SPEC, "SPEC benchmarks," http://www.spec.org.

[31] K. Strauss, X. Shen, and J. Torrellas, "Flexible snooping: Adaptive forwarding and filtering of snoops in embedded ring multiprocessors," in *International Symposium on Computer Architecture*, 2006.

[32] ——, "Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors," in *MICRO-40*, 2007.

[33] TPC, "TPC benchmarks," http://www.tpc.org.

[34] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 tflops network-on-chip in 65nm cmos," in *IEEE International Solid State Circuit Conference*, 2007.

[35] D. Wentzlaff, P. Griffin, H. Hoffman, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. B. III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, pp. 15–31, 2007.

[36] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA-22*, June 1995.

[37] J. Zebchuk, E. Safi, and A. Moshovos, "A framework for coarse-grain optimizations in the on-chip memory hierarchy," in *MICRO-40*, 2007.