High-Speed Formal Verification of Heterogeneous Coherence Hierarchies

Jesse G. Beu, Jason A. Poovey, Eric R. Hein, Thomas M. Conte
Georgia Institute of Technology, Atlanta GA
jesse.beu@gmail.com, japoovey@gmail.com, ehein6@gatech.edu, tom@conte.us

Abstract

As more heterogeneous architecture solutions continue to emerge, coherence solutions tailored for these architectures will become mandatory. Coherence hierarchies will likely continue to be prevalent in future large-scale shared memory architectures. However, past experience has shown that hierarchical coherence protocol design is a non-trivial problem, especially when considering the verification effort required to guarantee correctness.

While some strategies do exist for verification of homogenous coherence hierarchies, support for reasonable verification of heterogeneous coherence hierarchies is currently unavailable. Ideally, hierarchical coherence protocols composed of 'building block' protocols should be able to take advantage of incremental verification to side step the state-space explosion problem which hampers any large-scale verification effort. In this work, we prove this can be accomplished through the use of the Manager-Client Pairing (MCP) framework, which provides encapsulation and permission checking support that enables a form of state-space symmetry. When combined with an inductive proof, this ensures the validation properties of proper permission distribution and livelock/deadlock freedom are enforced by any hierarchical composition of MCP compliant protocols. Demonstration of this methodology through the MurPhi formal verifier shows several orders of magnitude improvement in verification cost compared to full hierarchy verification.

1. Introduction

It is well established that power constraints have caused a major paradigm shift in computer architecture towards parallel processing for performance scaling. With it have come new opportunities and design spaces for architects to explore. Among these are heterogeneous architectures, where on-chip network and processor diversity can be exploited for performance benefit or power/energy savings [1-5]. Such systems benefit from the design of diverse interacting coherence protocols, where each protocol is optimized to take advantage of properties of a homogeneous region within the overall heterogeneous architecture. This comes at a cost however, in that the design and verification complexity of such systems is substantially higher than that of their homogenous coherence counterparts.

Despite this cost, the benefit of heterogeneous coherence has resulted in real-world applications of coherence heterogeneity. The Wildfire architecture, for example, was built using the existing first level protocol of the Sun E6500 in a larger hierarchy that enabled *Coherent Memory Replication* for improved node locality [3]. The Piranha architecture [4] had an *intra*-chip coherence management mechanism that was integrated with an independent *inter*-chip coherence protocol engine. This allowed for efficient use of on-chip caches and fast intra-chip data transfers while another DRAM directory-based protocol could be leveraged to enable scalability and performance at the inter-chip granularity. The HP Superdome [5] also employed a similar strategy as Wildfire, but with a different goal in mind. An inter-chip communication layer interfaced the native intra-chip protocol to a higher-level directory protocol. The resulting system was able to restrict message broadcast scope to the local protocol in many cases, enabling the use of commodity parts (i.e., those with "glueless" multiprocessor buses) in a large-scale system while maintaining performance. These examples suggest that heterogeneous coherence hierarchies will become more attractive in the present era as current technology trends continue.

Another factor motivating heterogeneous coherence support is the emergence of *Partitioned Global Address Space* (PGAS) languages, such as X10 [7], which explicitly express physical locality of memory through *places* and processor/thread affinity. Depending on the relationship between the size of the address spaces assigned to a place, the number of active threads operating within a place, and the available architectural resources, localized coherence protocols can be beneficial. Localized protocols can be optimized for a particular *place*'s partition of the address space and architectural real estate, while still maintaining global ad-

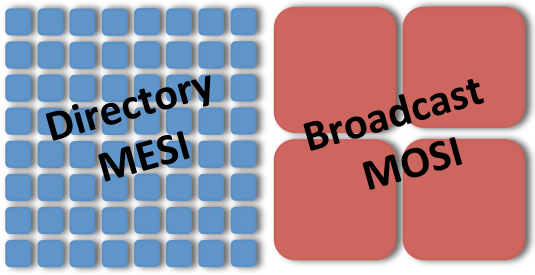

Figure 1 - Example of a heterogeneous multi-chip system that would benefit from heterogeneous coherence hierarchy support.

dress space coherence with respect to other localized protocols.

The designers of future architectures can also benefit from coherence heterogeneity. Consider, for example, a production heterogeneous chip that is partitioned across several different development teams. Each team wants to design its own highly optimized and specialized coherence protocol, tailored and verified for one architectural region. Each design group could work independently if a well-defined heterogeneous coherence composition framework were available to integrate the protocols into a final, *verified* hierarchical protocol, as shown in Figure 1. This concept of distributed coherence protocol design does not have to be limited to a single chip. With a composition framework, multi-chip systems comprised of diverse chips (GPUs and CPUs), from different vendors, could be combined and verified into a global coherence protocol.

Before implementing a coherence protocol in hardware, it is important that the protocol be *verified*. Given the extreme rate of processor requests that a protocol handles per second, even the smallest flaw will inevitably lead to a system failure. An incorrectly designed coherence protocol could cause the chip to deadlock or corrupt data by allowing multiple processors to modify the same block simultaneously. One approach to formally verifying a protocol involves modeling the protocol components and examining every possible reachable state for invalid behavior. The total number of global states to be explored increases exponentially with every new node, message type, or state that is added to the protocol.

Intractable verification complexity has the potential to dissuade architects from using hierarchical coherence approaches, despite their many benefits. While many strategies and tools already exist to assist in the verification effort of flat protocols [14-20], hierarchical coherence breaks these tools by exacerbating many of the problems associated with verification, such as the state space explosion problem [10]. Recent publications [8,11] have demonstrated very powerful techniques to accelerate verification for hierarchical coherence protocols, but they are limited by a fundamental assumption: that the hierarchy being verified is composed of *homogenous* and *self-similar* protocols. Such an assumption severely limits the utility and scope of hierarchical coherence for heterogeneous designs or PGAS models. Extending verification to hierarchies of distinct coherence protocols is a hard problem. However, as discussed earlier, there will be a strong desire for flexible, *heterogeneous* coherence hierarchies in the near future. A solution to the verification problem must be found.

We believe that a recently published framework for coherence composition holds the key to heterogeneous hierarchy verification. The *Manager-Client Pairing* (MCP) composition framework [12] enables rapid development of heterogeneous coherence hierarchies through the definition of a standardized protocol interface and component protocol encapsulation. In [12], the authors demonstrate a methodology for composing heterogeneous protocols with minimal effort, and present results for a variety of multi-tiered coherence hierarchies. They do not, however, make any claims regarding verification of these hierarchies.

In this work, we extend MCP by proving that using *MCP compliant* protocols in an MCP hierarchy enables rapid verification through a form of *protocol symmetry* [10]. This avoids the need for full state space exploration, reducing verification cost from an intractably large combinatorial space down to verifying each component protocol independently. The contributions of this work are as follows:

- Introduce a new form of protocol structural symmetry called *encapsulation symmetry*, and show how it can reduce verification cost.
- Prove that MCP supports *encapsulation symmetry* and thus can be leveraged as a *verification composition framework* for heterogeneous hierarchies when the hierarchy is composed of formally verified *MCP compliant* protocols.
- Present *remote proxy client* as a technique for porting pre-existing, verified protocols to *MCP compliance* with little design and verification overhead. As a motivating example, this technique is applied to the Broadcast-MOSI protocol from GEMS [6] to enable its integration with a Directory-MESI protocol to form a MCP hierarchy.
- Show through the MurPhi formal checker [14] that this new MCP hierarchy is verified. Further, we use this result to compare the cost of full state-space exploration with that of independent component verification via *encapsulation symmetry*.

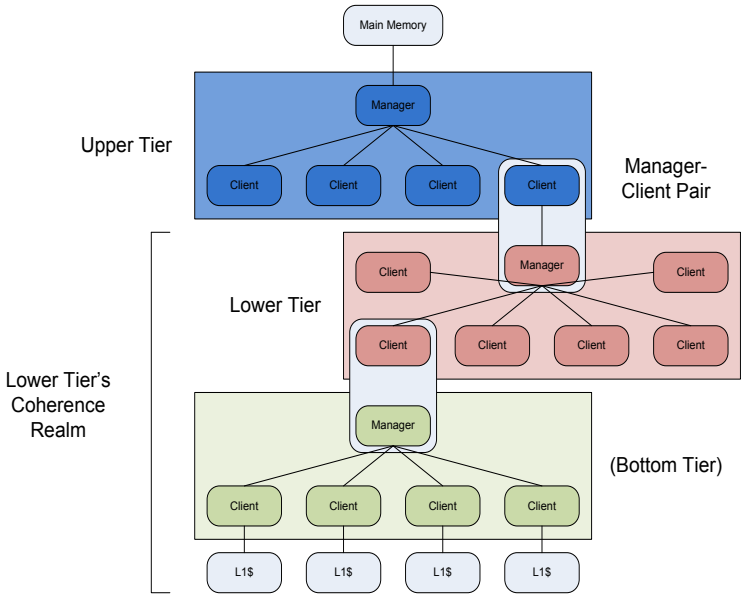

Figure 2 - Manager-Client Pairing coherence hierarchy organization with parts labeled: Manager, Client, Tier, and Realm for the Coherence Domain.

The remainder of the paper is organized as follows: Section 2 outlines the related work. Section 3 presents an overview of the MCP framework. Section 4 explains the state enumeration verification strategy in preparation for Section 5, which presents a proof for verification through MCP composition. Section 6 outlines how to adapt existing protocols to be *MCP compliant* via a remote proxy client. Section 7 presents MurPhi verification results followed by a conclusion in Section 8.

2. Related Work

Due to the importance of verification, there is a large body of related work available. For brevity, this section will only mention those most closely related to the problem of *hierarchical coherence verification*.

Ladan-Mozes and Leiserson [11] propose a deadlock-free, tree-based coherence protocol in order to ensure forward progress in a fat-tree network. By enumerating invariant properties that ensure all children in the tree are coherent with parents, permission guarantees can be made with respect to exclusive write permission, while supporting multiple readers.

An important work that eases *homogeneous* hierarchical coherence verification is Fractal Coherence [8]. In this work, Zhang et al. propose a tree-based coherence protocol, with the intention of simplifying coherence verification through perfect self-similarity. A fractal based coherence protocol, where children are coherent with their parents, can be verified through the validation of only the kernel coherence protocol. The authors also describe how a bus-based version of the protocol could also be executed through fractal buses. Fractal coherence has many similar features to MCP. The recursive nature of the interfaces proposed by MCP is analogous to the self-similarity of fractal coherence's kernel protocol.

The most important distinction between prior work and this work is that prior work was specifically designed with *homogeneity* as a requirement. Neither work discusses the benefits of heterogeneous coherence composition nor why it is an important consideration. In fact, both [8] and [11] are explicitly incompatible with heterogeneity since they both rely heavily on homogeneity in their proofs. It is worth mentioning, however, that because these techniques do produce verified coherence protocols, they would be compatible as a component within an MCP coherence hierarchy if made *MCP Compliant*.

3. Review of MCP framework

Manager-Client Pairing (MCP) eases hierarchical coherence protocol design by distinguishing manager agents, those that manage permissions (e.g. directory), from client agents, those that hold permissions (e.g. private caches) [12]. By pairing the client agent of a higher protocol with the manager agent of the lower protocol, the client agent behaves as a permissions gateway for the paired manager's protocol. This is possible because MCP defines a permission-checking algorithm that enables component protocols to communicate with each other through a generic query-and-acquire interface, eliminating the need to expose internal operation details outside the protocol's scope. By linking protocols together, coherence hierarchy composition can distribute the coherence responsibility throughout the hierarchy's *coherence realms*. The top-tier *coherence realm* encompasses all users of data

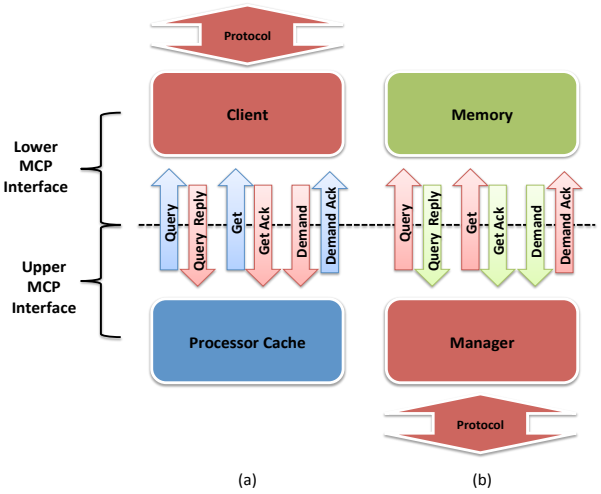

Figure 3 – MCP Interface for (a) lower processor tier and (b) top memory tier

within the coherent memory system being monitored by the hierarchical protocol. Each lower-tiered *coherence realm* monitors successively smaller subsets of node coherence. Figure 2 shows an example MCP hierarchy, labeled with MCP terminology.

Due to the general interface definition and resultant low level of integration required between realms, previous work [12] demonstrated that component coherence encapsulation is well preserved, meaning the design details of the protocols used to comprise the system are largely opaque with respect to one another. Furthermore, because this interface's functionality is very similar to the processor and memory interfaces in a conventional flat coherence protocol, the majority of the effort required to adhere to *MCP compliance* is a straightforward one-to-one mapping between MCP actions and already present coherence actions. We define a protocol to be *MCP compliant* if it is a verified invalidation-based coherence protocol (see Section 5) that only communicates with the external world through upper (memory) and lower (processor) MCP interfaces as shown in Figure 3.

4. Reachable State Enumeration Overview

Before constructing the complete proof for MCP-hierarchy validation, an understanding of the underlying verification principles is required. In this section we introduce the problem of *verification through reachable state enumeration.* We discuss verification through enumeration, review the state-space explosion problem, and explain how past research has mitigated this problem through the use of *protocol symmetry*. This leads to our key observation, that the state-space explosion due to hierarchical protocol interactions can also be mitigated if viewed as a form of symmetry.

4.1. State Enumeration

Reachable state enumeration is a common strategy employed in coherence protocol verification that automates the process. First, the protocol state machines and surrounding communication medium are described in a protocol description language, such as *MurPhi* [14]. A set of invariants is then defined to establish what conditions must be met for the system to be valid (e.g., *only one modifiable copy of a cache block exists at any time*). Relevant parameters regarding the system configuration (*number of clients, manager organization, network properties*, etc.) are provided, as well as an initial system state from which the verification process can begin. All possible states are then exhaustively generated and invariants checked, following the actions provided in the description. This can be done by either applying a depth-first or breadth-first search, where next-states are generated by applying all possible valid rules to the current state (e.g., *new request generation, request/response event delivery*, etc.). Each new state checks the invariants and, if no violation occurs, marks the current state of the system as *reached* (this is often implemented through the use of a hash table populated with a compressed state notation). If a future-state sequence encounters a state that has already been reached, that branch of the search can be terminated since it has previously been verified. Eventually, all branches will terminate, and, if no violation has been encountered, the protocol can be labeled as verified.

4.2. State-Space Explosion and Symmetry

For even reasonably simple coherence protocols, the state space that needs to be exhaustively searched can become intractable quickly. This is due to all the possible state interactions between the clients state machines, manager state machine, and various states of message delivery and ordering, which is aggravated rapidly by how many nodes (i.e. cores) are being modeled. While prior research has proven that modeling of a single cache block address is sufficient to verify a coherence protocol [10], there is no proof that a large-scale system can be fully verified from a similar, scaled-down system. As each additional node is added to the system, the number of possible global states increases exponentially due to all possible interactions between the newly-added client's state machine (and messages) with the previous system's state-space, as well as the additional possible manager states from extending the tracking mechanism to encompass the new node's tracking. For an example of the latter, consider moving from 8 bits to 9 bits in a sharer bit-vector: this results in an increase from 2^8 to 2^9 possible vector states *for each manager state* that requires bit-vector information. Because of the combinatorial na-

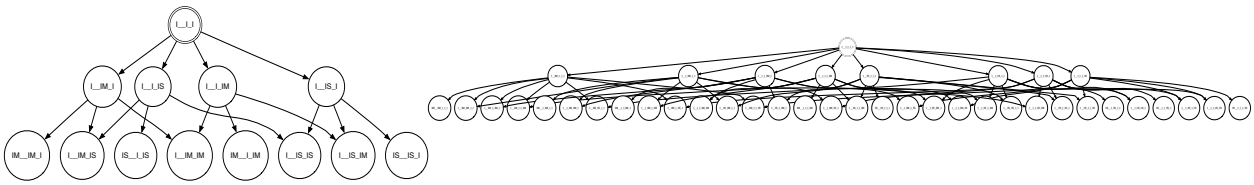

Figure 4 - Example of state space explosion when adding 2 additional nodes to a 2-node MSI protocol.

ture of the state space problem, we see in Table 1 a dramatic increase in the number of reachable states as the client count increases. These results were collected from a full state space exploration using MurPhi. Figure 4 presents a visual representation of what happens during state-space explosion. This example only shows the reachable states after the first two possible rules are applied to an overly simplified MSI protocol consisting of 2 nodes vs. 4 nodes.

Due to the often-homogenous nature of client state machines in a coherence protocol, *state symmetry* has been shown to be a powerful way to combat the state-space explosion problem, and can reduce the state-space search scope by as much as 90% [10]. In this approach, several distinct states can be shown to overlap with one another through the exploitation of structural symmetries in the protocol's design, such as abstracting sharer client ID information to a sharer client count. For example, the 4-node composite states {S,S,I,O}, {I,S,S,O} and {O,S,S,I} are symmetric with one another because a simple substitution can show that applying the same sequence of rules that lead from the initial state to each of these states will yield identical results if node ids are rotated/mixed (e.g. {I,S,S,O} becomes {O,S,S,I} if node 0 and node 3 are switched). Again, because of the homogeneity of the client's state machines, there is no behavioral difference at the higher-level description of the protocol behavior; specific node identity information is unimportant. In this way, global state can be viewed as a combination rather than a permutation. In short, *if two system-wide states are symmetric with one another, only one has to be verified to automatically verify the other*. The authors of [10] demonstrate that the notion of structural symmetries extends beyond just node ID abstraction to encompass many other parts of coherence protocol design, including "addresses, data values, memory module-ids and message-ids." In this work we extend this to encompass the *encapsulation symmetries* present in hierarchies composed of independent, well-encapsulated protocols.

Table 1 – Verification cost of Directory-MESI and Broadcast-MOSI protocols using MurPhi

Protocol	# of States	Time to Verify [s]
2-client Directory-MESI	599	0.10
3-client Directory-MESI	7,077	0.13
4-client Directory-MESI	108,203	3.33
5-client Directory-MESI	1,345,019	91.76
6-client Directory-MESI	26,361,918	15,980.70
2-client Broadcast-MOSI	3,117	0.10
3-client Broadcast-MOSI	166,562	4.79
4-client Broadcast-MOSI	4,307,049	331.82
5-client Broadcast-MOSI	132,871,278	303,244.00
6-client Broadcast-MOSI	500,000,000+	4,000,000+

4.3. Encapsulation Symmetry

Encapsulation symmetry is different from state symmetry in that it does not manifest as a result of protocol homogeneity. Rather, encapsulation symmetry happens when portions of the global state representation can be proven to be independent from other parts of the global state. The simplest example of this phenomenon would be the state-space exploration of two completely isolated state machines, n and m, operating simultaneously. If the size of each state machine's state-space could be expressed as $size_n$ and $size_m$, the state space of both operating simultaneously is ($size_n$ * $size_m$). This is evident because a simple scan could explore the entire space by repeatedly applying a single rule to n, followed by full exploration of state machine m's space.

To express this another way, if the overall state of a system is represented as a string, the state space of each independent state machine can be expressed as strings $string_m$ and $string_n$. The entire state space of these operating simultaneously could then be expressed as the combination of all valid $string_m$ strings concatenated with all valid $string_n$ strings. Figure 5 and Figure 6 show the symmetry in the state space visually for a pair of simple state machines.

Leveraging this kind of symmetry for coherence hierarchy verification would be extremely powerful in combating the state space explosion problem, allowing each component protocol to be verified independently and then merged. However, this symmetry requires proving that the integrated protocols are sufficiently isolated from one another through some form of encapsulation. Additionally, valid merging would require all possible concatenation combinations of these state spaces to guarantee invariant violation freedom. Section 5 will develop this further and demonstrate that the interfaced and permission summarizing nature of *MCP compliance* will produce *encapsulation symmetry* in the state space that can safely be leveraged for rapid verification.

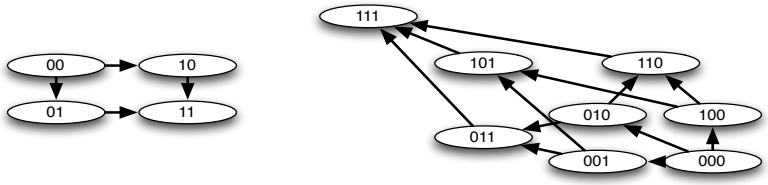

Figure 5 - State-space of two simple state machines, where each element may transition from 0 to 1

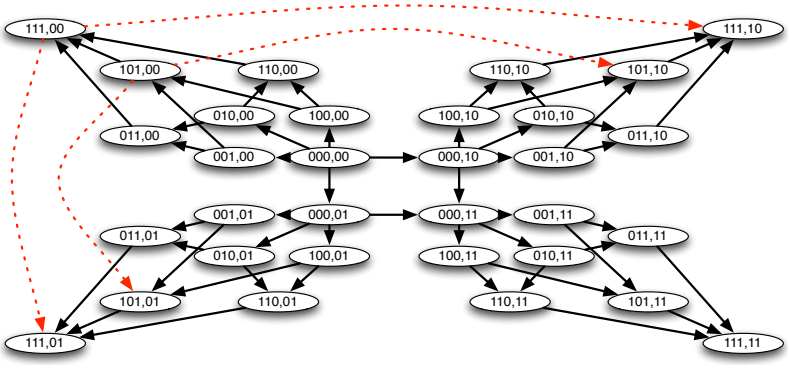

Figure 6 - Full state space exploration of both state machines operating simultaneously, where black arcs represent transitions using the 'execute a single n rule, followed by full m exploration' methodology, and red dotted lines show a few of the alternative paths that would encounter redundant states in the space.

5. Formal Verification Strategy for MCP

We propose the use of MCP as a framework for high-speed formal verification of large-scale hierarchical, heterogeneous protocols. In this section we will prove that when formally verified MCP-compliant protocols are assembled into a hierarchy and connected through MCP-interfaces, the hierarchy is also verified. We define '*verified*' to mean a protocol can guarantee the following properties: (1) There can be at most one lowest-tier client with write permission to a block of data; (2) There can be one or more lowest-tier clients with read permission to a block of data if no other lowest-tier client has write permissions; (3) reads are guaranteed to supply the requestor with the most recently written data value at the time the read was inserted into the global order; (4) The system is deadlock and livelock free. These provide a guarantee of coherence protocol design correctness.

Definition 1 –
A protocol is said to be verified if:

1) ∀ reachable global states in a protocol x, a node may have write permissions to a block iff there are no other nodes with read or write permissions to that block.

2) ∀ reachable global states in a protocol x, one or more nodes may have read permissions to a block iff there are no nodes with write permissions to that block.

3) ∀ reachable global states in a protocol x, read requests to a block obtain the value written by the most recent previous write in the global order, w.r.t the read, to that block.

4) ∀ reachable global states in a protocol x, there are no states without possible exits (deadlock) and no condition where a given data block is locked by one node such that it is permanently prevented from being accessed by other nodes (livelock) [20].

As mentioned previously, we define a protocol to be *MCP compliant* if it is a verified invalidation-based coherence protocol that only communicates with the external world through MCP interfaces.

5.1. Theorem 1 – Two-tier MCP composition and verification

Where R(u,l) := Coherence Realm from interfacing of upper-tier MCP compliant protocol u with lower-tier MCP compliant protocol l through pairing of a u-client with the l-manager.

Lemma 1 – MCP permission distribution ensures R(u, l) will satisfy conditions (1, 2, 3)

Lemma 2 – For R(u, l), MCP Get/GetAck and Demand/DemandAck pairs do not violate condition (4); all requests are eventually satisfied since both protocols u and l have been previously verified

Theorem 1 – ∴ ∀u ∀l, where u and l are MCP compliant protocols, R(u, l) is also verified and MCP compliant.

The supporting lemmas for Theorem 1 have two main themes: Lemma 1 is concerned with proper distribution of permission guarantees to ensure that conditions 1, 2 and 3 of verification are enforced (one writer, multiple readers, read consistency) while Lemma 2 focuses on livelock/deadlock adherence. In

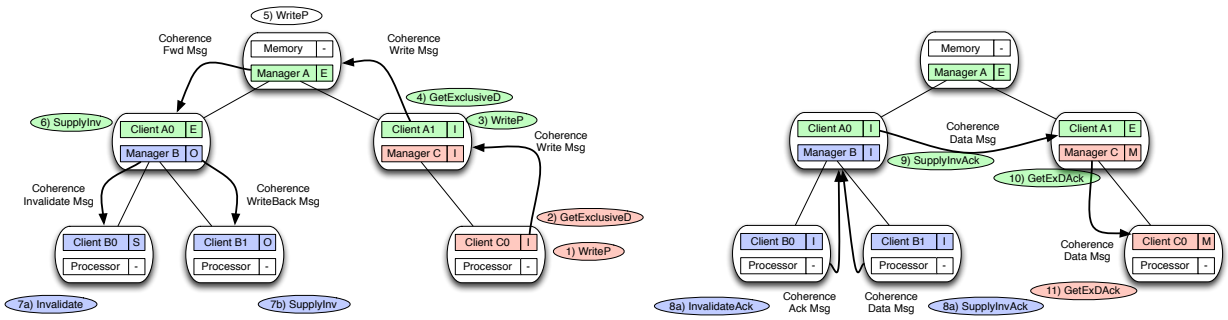

Figure 7 - Permission distribution example for an MCP composition

Lemma 1, Condition 3 is satisfied because the manager/client pairing is located at the ordering point for its realm, ensuring global ordering of reads and writes is maintained throughout the hierarchy. Conditions 1 and 2 are fundamental properties of MCP composition, and are discussed in depth in prior work [12] which details the permission allocation algorithm and how the permission inclusion property described by Ladan-Mozes and Leiserson in [11] is implemented by MCP. The realm-miss example from [12] demonstrating this is reproduced here for completeness.

In Figure 7, the sequence of MCP interface events and corresponding coherence actions to acquire data across realm boundaries is shown, starting with (a) the request and demand chain of events and (b) the ack event sequence replying to these requests and demands.

First, the processor paired with Client C0 discovers it has insufficient permission to satisfy a write (1). This results in a *GetExclusiveD* call to Client C0 (2) which spawns a coherence message to Manager C requesting the data and write permission. Following the MCP algorithm, before responding to the coherence request the paired client A1 is consulted (3). Since A1 does not have sufficient permissions, Manager C temporarily stalls the coherence request from C0 and Manager C issues a GetExclusiveD to its paired client A1 (4). This results in coherence traffic that leads to Client A0 *Demand*ing the lower realm managed by Manager B to supply data and self-invalidate (6). Coherence messages are sent to invalidate all nodes in the realm and request data writeback (7a and 7b).

At this point traffic begins to flow back towards the originating request through reply acknowledgments (8a and 8b). Coherence traffic flows back to Manager B, enabling it to transition to the invalid state and supply data to its paired client A0 (9). The paired client can now proceed by taking its native protocol action, forwarding data to A1 and self invalidating. Upon arrival at A1, the client state transitions to Exclusive and a *GetExclusiveDAck* is issued across the MCP interface to Manager C. Finally, Manager C can resume processing of the original coherence write message and respond with a coherence data message. Upon reception at Client C0, the write action is complete. This demonstrates that despite having multiple discrete, encapsulated protocols that treat each other as black boxes, permissions are properly enforced across the entire hierarchy because of MCP.

Lemma 2 leverages the fact that all incoming *Get* actions observed by the lower-tier's lower interfaces (e.g. processor caches) will be satisfied either (a) locally by the lower-tier, (b) remotely by the upper-tier through the lower-tier's upper interface, or (c) by memory via issuance of a Get action from the upper-tier's upper interface with memory. Similarly, all upper-tier lower interfaces not connected to the lower-tier realm will be satisfied either (d) locally by the upper-tier, (e) remotely by the lower-tier through the MCP interface or (f) by memory via the upper-tier's upper interface. In all these instances, eventual completion is guaranteed since memory will always respond and the upper-tier and lower-tier protocols are guaranteed to be livelock and deadlock free prior to composition as a condition of being *MCP compliant*. All requests are satisfied locally, satisfied by memory, or deferred to another protocol that can guarantee eventual response to any request. Additionally, due to the tree-like organization of an MCP composition and permission distribution, there is no possibility of a cycle in which two *MCP compliant* protocols are waiting on each other to eventually respond.

Since *MCP compliant* component protocols are independently verified, we know that no *Get* action can be delayed indefinitely since *Get* actions are functionally equivalent to cache actions (read, write, evict). In an MCP composition, *Get* requests are either satisfied locally, deferred upwards via another *Get* request which in turn will recursively do the same until satisfied, or deferred downwards via a *Demand* request (cache-to-cache forwarding behavior, for example). This ensures that all requests will make forward pro-

gress as they traverse up or down the tiers until satisfied, proving livelock is not possible. Finally, because MCP does not introduce new states or messages to the component protocols, no new state without exit can arise or be reached, protecting against deadlock.

MCP components meet all the conditions from the definition of verifiability. Rules regarding read permission and write permission distribution for all lowest level clients (i.e., caches) are enforced while guaranteeing livelock and deadlock freedom for all reachable states. Therefore Theorem 1 is proven: a composition of an upper-tier *MCP compliant* protocol and a lower-tier *MCP compliant* protocol, connected through an MCP interface will properly distribute permissions and data while retaining livelock and deadlock freedom. Since no verification violation can possibly occur when merging these two protocols, we can safely say the cross-product of their respective state spaces into a unified state space will not introduce any new violating states, enabling us to apply *encapsulation symmetry* from Section 4.2 for verification.

5.2. Theorem 2 – Arbitrarily deep MCP Hierarchies

Axiom 1: R(u, l) is both verified and has MCP compliant upper and lower interfaces, being a composition of MCP protocols. ∴ R(u, l) is also a verified MCP compliant protocol.

Theorem 2 – Arbitrarily deep MCP coherence hierarchies are verifiable through induction via the following:

H(2) = R(u, l), where H(2) is a verified MCP compliant protocol hierarchy of two tiers, and

H(n +1) = R(H(n), l), where H(n+1) is a verified MCP compliant protocol hierarchy of (n +1) tiers

Axiom 1 stems from the structurally recursive nature of MCP composition. From Theorem 1 in the previous sub-section, we know a 2-tier coherence realm composed of independently verified MCP protocols is also verified. Additionally, because each component protocol only has an upper interface and lower interface(s), and the upper interface of protocol '*l*' is attached to one of the lower interfaces of protocol '*u'*, the remaining unconnected interfaces are a single valid upper interface (the '*u*' protocol's upper interface), and multiple valid lower interfaces (includes all the lower interfaces of 'l' and all the lower interfaces of 'u' except the most recently connected). Therefore, the whole is a verified protocol with valid upper and lower MCP interfaces, with no other external communication interfaces, meeting all the conditions for *MCP compliance*.

In a k-tiered MCP hierarchy, the highest coherence realm in the hierarchy (which begins by encompassing only the two top-most tiers of the system) can be proved to be a verifiable MCP compliant protocol through Theorem 2 and Axiom 1. As a result, the two tiers of this realm can logically be replaced by a 'single' MCP compliant protocol, which is the merger of these two tiers (shown in the equations supporting Theorem 2). Through this process, the k-tiered MCP hierarchy has become a (k-1) tiered hierarchy, where the highest protocol in the hierarchy is itself a coherence hierarchy. This can be applied repeatedly until all k-tiers have been merged into the single verified MCP compliant protocol. Figure 8 demonstrates this induction graphically.

5.3. Fractal Coherence Viewpoint

Theorem 2 can also be understood through the theorems in the verification process of Fractal Coherence [8]. The two most important properties required for application of Fractal Coherence verification is that (a) the minimum system is formally verified and (b) the hierarchy is observationally equivalent.

Rather than assuming only a single minimum system being replicated, MCP composition assumes multiple systems being integrated that may not be identical. However, if each component is independently formally verified, this is similar to a single kernel protocol being verified and used repeatedly. Additionally, a kind of observational equivalence can be gained through the use of a standardized interface, which MCP provides. From Theorems 1 and 2, we know each component of

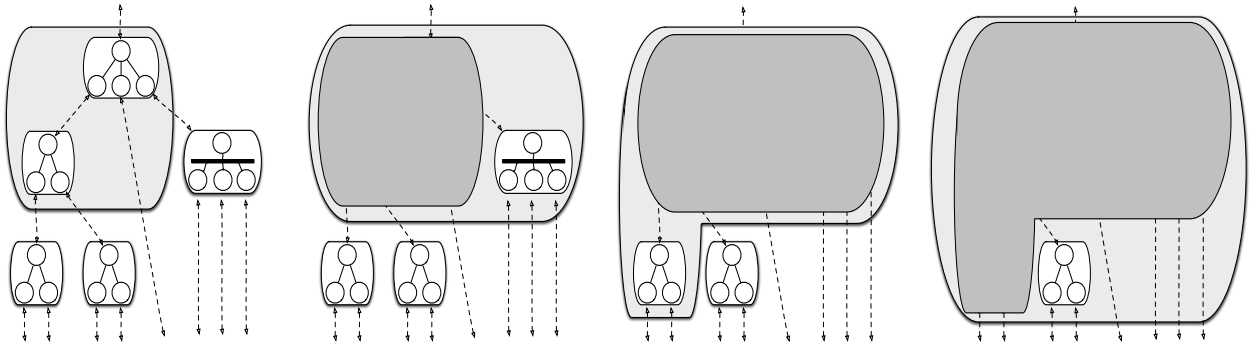

Figure 8 - Graphical representation of coherence hierarchy inductive proof, where the shaded enclosed region represents H(n) for n = 2 → k (k = 4).

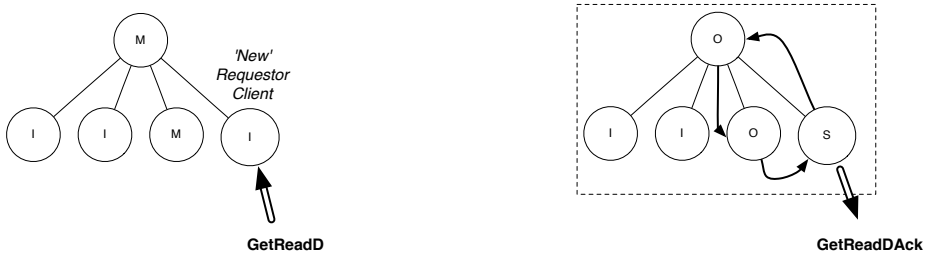

Figure 9 - Local GetReadD Get sequence (Request and Response) in a MOSI protocol that currently holds write permissions.

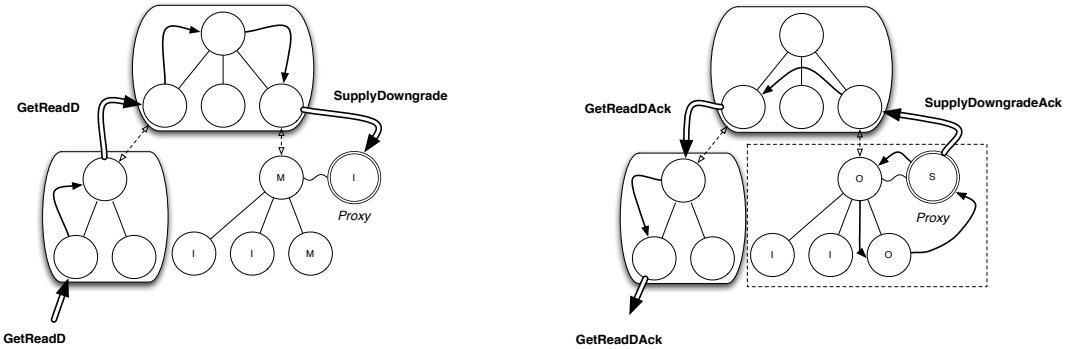

Figure 10 - Remote GetReadD Get sequence (Request and Response), using a proxy client to satisfy SupplyDowngradeAck Demand request in a MOSI tier that currently holds write permissions

an MCP hierarchy is formally verified, and connection of these through manager-client pairing of interfaces does not violate verification. As described in Section 5.2 regarding Axiom 1 and the interfaces, when treated as black boxes, compositions of MCP component protocols are *nearly* observationally equivalent because each additional tier connects to either upper or lower interfaces while providing new upper or lower interfaces that are functionally equivalent. They are not *strictly* observationally equivalent because the number of interfaces changes depending on the number of clients in the newly attached MCP component protocol. In contrast to Fractal Coherence, which enforces observational equivalence by the 'component protocols' being perfectly self similar, MCP enforces a looser observational equivalence through adherence to a standardized interface definition.

6. Remote Proxy Client

6.1. Theorem 3 – Verification of protocols modified with remote proxy client

Where M(x) := An MCP compliant version of protocol x

Theorem 3 – If protocol x satisfies conditions (1,2,3) for verification, and replaces one client with a remote proxy client, the resulting protocol M(x) does not introduce changes that violate conditions (1,2,3)

∴ ∀x where protocol x is verified, M(x) is also verified

Recall from Section 3 and Appendix A, there are three major parts to the MCP interface: *Queries* (permission checking), *Gets* (permission acquisition) and *Demands* (permission surrendering). Let us first consider the *Query* and *Get* portions of the MCP interface. For *Queries*, permission status requests cannot modify the state of the protocol as they are simple Boolean checks and therefore have no verification impact. In order to evaluate Theorem 1 with respect to applying the MCP interface specification to a verified non-*MCP compliant* protocol, the only actions from Appendix A that must be considered are those that can result in state change in the underlying protocol. Each such action much be mapped to an already existing action in the protocol, based on the internal state of the protocol.

As mentioned in Section 3, however, the *Get* functionality required for supporting MCP corresponds directly to functionality that must already be present for handling and satisfying processor requests in a non-MCP version of the protocol interfaced directly with a processor.

The biggest hurdle when mapping a pre-existing protocol's functionality to the MCP interface is implementing upper-tier initiated *Demands*. The memory controller interface is typically not able to issue requests for invalidations or downgrades in a conventional flat coherence protocol. However, these actions can be emulated very easily if upper-tier *Demands* are modeled as requests from a local client, similar to the pseudo-CPU mechanism in DASH [2]. In our framework, this functionality is served by the *remote proxy client*.

The *remote proxy client* acts on behalf of the upper-tier, issuing local protocol requests to satisfy incoming *Demand* requests. In addition to this, the *remote proxy client* is also stateful; it becomes a summary of all the permissions held by nodes external to this coherence realm. As long as the protocol is verified, permission will be assigned to this proxy correctly. This ensures permission exclusion is preserved when necessary, and permission will eventually be passed to the appropriate originator that caused the *Demand*. The *remote proxy client* does not communicate directly across tier boundaries; remote traffic is routed through the MCP interface in the manager.

Figure 9 and Figure 10 show an example of how the *Demand* handling behavior of the *remote proxy client* is similar to the behavior of a local requestor in a MOESI directory protocol. The actions required by the coherence realm to satisfy a *SupplyInvalidate* (See Appendix A) are identical to those required for handling a write request from a client in the invalid state. A message is sent to the owner client (the client in either the M, O, or E state) to initiate a cache-to-cache transfer and a self-invalidation. Invalidation messages are sent to all other sharers in the bit vector. When this sequence concludes, the realm manager and other clients will have given write permissions and a copy of the data to this client by removing all readable copies from the realm. In the case of the *remote proxy client*, the realm can respond with a *SupplyInvalidateAck* upon completion of the protocol sequence since all conditions are met (i.e., the realm no longer has any copies with read or write permission and the most recent copy of the data is available and ready for forwarding).

Implementing *remote proxy client* does not actually require adding another client to the protocol. Rather, an existing client can be sacrificed to act as the *remote proxy client.* So a protocol that is verified for four clients could be made into a 3-client *MCP compliant* protocol by selecting one of the clients to serve the role of a *remote proxy client.* Since *Demand* handling does not introduce new states or messages when a proxy is present, there are no changes to the original state machine, and the protocol remains verified.

In summary, encapsulating a protocol via MCP interfaces can be seen as applying a translation layer that introduces no new additional state transitions, states, or new messages to the protocol. This preserves the verification properties of the original component protocol. Since MCP does not modify the state machine, if the base protocol correctly distributes permissions without deadlocking or livelocking, so does an MCP compliant version of the same protocol.

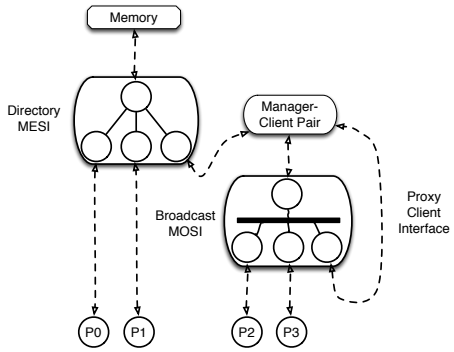

Figure 11 – Evaluated Heterogeneous Hierarchical Protocol Structure

7. Results

In order to demonstrate the usefulness of MCP as a verification technique, a heterogeneous hierarchy was implemented and verified using the MurPhi toolkit. The hierarchy was created from the composition of two protocols: Directory-MESI and Broadcast-MOSI. We designed the Directory-MESI protocol to be natively MCP compliant, without the need for a *remote proxy client.* The Broadcast-MOSI protocol is a MOSI protocol communicating over a shared bus and was ported directly from the GEMS implementation to MurPhi. To make this protocol MCP compliant, a client was scavenged to serve as the *remote proxy client;* no other changes were made to the underlying "off-the-shelf" GEMS protocol.

The evaluated heterogeneous hierarchical protocol is shown in Figure 11. The number of clients in each protocol is varied and the configurations are denoted in Table 2. The figure illustrates the Dir3 + B3 configuration, where one client in the Directory-MESI is paired with the Broadcast-MOSI, and one client in the Broadcast-MOSI is dedicated as a *remote proxy client.*

The hierarchical protocol was evaluated via full state exploration using MurPhi, and was also verified by leveraging MCP structural symmetry. The results of this verification effort are denoted in Table 2. The number of states represents the full state space of the combined protocols, and the time to verify without MCP is the total time for MurPhi to complete a full state exploration (similar to Figure 6). The verification costs with MCP is simply the sum of the verification costs of the component protocols, due to encapsulation symmetry. As is evident from these results, MCP greatly reduces the verification cost, especially at the higher client count design points.

Table 2 – Comparing verification cost of heterogeneous hierarchical protocols with and without leveraging MCP protocol structural symmetry[1]

Protocol	# of States	Time to Verify [s]	# of States (w/MCP)	Time to Verify (w/MCP) [s]
Dir2 + B2	11,861	0.34	3,716	0.20
Dir2 + B3	425,990	17.31	167,161	4.89
Dir3 + B2	182,197	8.51	10,194	0.23
Dir3 + B3	5,367,735	542.44	173,639	4.92
Dir4 + B3	71,642,216	84,734.63	274,765	8.12
Dir3 + B4	143,552,706	317,891.00	4,314,126	331.95
Dir4 + B4	500,000,000+	7,000,000+	4,415,252	335.15

8. Conclusion

There is a strong interest in multi-core architectures that use flexible, heterogeneous coherence hierarchies, such as CPU+GPU pairings or multi-vendor coherent shared memory ensembles. But without a verification solution, these protocols—and the potentially powerful and energy-efficient systems they enable—cannot be built. It is clear that a solution to the verification problem must be found. Prior solutions were limited to *homogeneous* hierarchies wherein every level of the system must practice the same protocol [8, 11]. This paper leveraged the recently published Manager-Client Pairing encapsulation composition framework [12], which explicitly supports heterogeneity. Using MCP, we proved that any heterogeneous protocol could be verified in no more time than it would take to validate each individual protocol in isolation.

The theoretical nature of this paper is inescapable. However, we have tried to bring this work to reality by implementing a coherence hierarchy in a formal verification tool. The intractability of obtaining results for our largest simulations establishes the need for formal verification acceleration. We defined a new form of protocol structural symmetry for coherence hierarchies, based on protocol encapsulation and permission distribution. We proved how MCP can be used as a verification composition framework for heterogeneous hierarchies composed of pre-verified protocols. With the framework presented here, hierarchical, heterogeneous coherence can become an industrial success rather than being limited by practical verification complexities.

References

[1] S. Haridi and E. Hagersten, "The Cache Coherence Protocol of the Data Diffusion Machine," presented at the Proceedings of the Parallel Architectures and Languages Europe, Volume I: Parallel Architectures, 1989.

[1] Dir4+B4 state space was too large for full state space verification to complete, due to the intractable nature of state-space explosion. The numbers presented in the first two columns for these configurations are the minimum bounds collected from the periodic progress report after 80 days of execution.

[2] D. Lenoski, et al., "The directory-based cache coherence protocol for the DASH multiprocessor," SIGARCH Comput. Archit. News, vol. 18, pp. 148-159, 1990.

[3] E. Hagersten and M. Koster, "WildFire: A Scalable Path for SMPs," presented at the Proceedings of the 5th International Symposium on High Performance Computer Architecture, 1999.

[4] L. A. Barroso, et al., "Piranha: a scalable architecture based on single-chip multiprocessing," presented at the Proceedings of the 27th annual international symposium on Computer architecture, Vancouver, British Columbia, Canada, 2000.

[5] G. Gostin, et al., "The architecture of the HP Superdome shared-memory multiprocessor," presented at the Proceedings of the 19th annual international conference on Supercomputing, Cambridge, Massachusetts, 2005.

[6] M. M. K. Martin, et al., "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," SIGARCH Comput. Archit. News, vol. 33, pp. 92-99, 2005.

[7] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. 2005. X10: an object-oriented approach to non-uniform cluster computing. *SIGPLAN Not.* 40, 10 (October 2005), 519-538

[8] Meng Zhang, Alvin Lebeck, Daniel Sorin, "Fractal Coherence: Scalably Verifiable Cache Coherence," presented at the International Symposium on Microarchitecture, Atlanta, Georgia, 2010.

[9] M. M. K. Martin, et al., "Token coherence: decoupling performance and correctness," presented at the Proceedings of the 30th annual international symposium on Computer architecture, San Diego, California, 2003.

[10] C. Norris Ip and David L. Dill. 1996. Better verification through symmetry. *Form. Methods Syst. Des.* 9, 1-2 (August 1996), 41-75

[11] E. Ladan-Mozes and C. E. Leiserson, "A consistency architecture for hierarchical shared caches," presented at the Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures, Munich, Germany, 2008

[12] J. G. Beu, M. C. Rosier and T. M. Conte, "Manager-Client Pairing: A Framework for Implementing Coherence Hierarchies," *Proceedings of the 44th Annual International Symposium on Microarchitecture (MICRO-44),* (Porto Alegre, Brazil), Dec., 2011.

[13] Frans H. van Eemeren and Rob Grootendorst, "The Fallacies of Composition and Division", in "JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday", Amsterdam University Press 1999. http://www.illc.uva.nl/j50/contribs/eemeren/eemeren.pdf

[14] *"Protocol Verification as a Hardware Design Aid,"* David L. Dill, Andreas J. Drexler, Alan J. Hu and C. Han Yang, 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, IEEE Computer Society, pp. 522-525.

[15] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," ACM Comput. Surv., vol. 28, pp. 626-643, 1996.

[16] K. L. McMillan, "Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional

Model Checking," presented at the Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, 2001.

[17] S. Park and D. L. Dill, "Verification of FLASH cache coherence protocol by aggregation of distributed transactions," presented at the Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, Padua, Italy, 1996.

[18] U. Stern and D. L. Dill, "Improved probabilistic verification by hash compaction," presented at the Proceedings of the IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, 1995.

[19] D. A. Wood, et al., "Verifying a Multiprocessor Cache Controller Using Random Test Generation," IEEE Des. Test, vol. 7, pp. 13-25, 1990.

[20] F. Pong and M. Dubois, "Verification techniques for cache coherence protocols," ACM Comput. Surv., vol. 29, pp. 82-126, 1997.

[21] Laudon, J. and D. Lenoski (1997). "The SGI Origin: a ccNUMA highly scalable server." SIGARCH Comput. Archit. News 25(2): 241-251.

[22] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. 1992. The Stanford Dash Multiprocessor. *Computer* 25, 3 (March 1992)

[23] Tom Lovett and Russell Clapp. 1996. STiNG: a CC-NUMA computer system for the commercial marketplace. In *Proceedings of the 23rd annual international symposium on Computer architecture* (ISCA '96)

Appendix A: MCP Actions

Lower Tier Manager to Upper Paired Client Permission Query

HaveReadP	Return true if paired Client has read permission
HaveWriteP	Return true if paired Client has write permission
HaveEvictP	Return true if paired Client can be safely evicted

Lower Tier Manager to Upper Paired Client Permission *Get*

GetReadD	Paired Client begins data and read permission acquisition sequence within it's native coherence realm. L1/Lower Manager expects GetReadDAck upon completion.
GetExclusiveD	Paired Client begins data and write permission acquisition sequence within it's native coherence realm. L1/Lower Manager expects GetExclusiveDAck upon completion.
GetExclusive	Paired Client begins write permission acquisition sequence within it's native coherence realm. L1/Lower Manager expects GetExclusiveAck or GetExclusiveDAck upon completion.
	Used when data is already available in L1/Lower Manager (HaveData == true) and only a permission upgrade is required.
	May be satisfied by a GetExclusiveDAck if upper tier protocol demands a downgrade while GetExclusive is in flight, causing HaveData to become false.
GetEvict	Paired Client begins eviction sequence within it's coherence realm. L1/Lower Manager expects GetEvictAck upon completion.
	Used when block ownership or most recent dirty version resides in L1/Lower Manager's realm.
	Needs to include data payload when data being evicted is dirty.

Upper Tier Client to Lower Paired Manager Permission Request Reply

GetReadDAck	Response by paired Client to complete previous GetReadD request. Supplies data packet and signifies paired Client (and thus lower Manager's realm) now has read permissions.
GetExclusiveDAck	Response by paired Client to complete previous GetExclusive/GetExclusiveD request. Supplies data packet and signifies paired Client (and thus lower Manager's realm) now has write permissions.
GetExclusiveAck	Response by paired Client to complete previous GetExclusive request. Signifies paired Client (and thus lower Manager's realm) now has write permissions.
GetEvictAck	Response by paired Client to complete previous GetEvict request. Signifies paired Client has become invalid. Therefore, Manager's realm can safely eliminate all local copies of the block.

Upper Tier Client to Lower Paired Manager *Demand*

Supply	Demand data supply from lower tier's paired Manager or L1. No additional actions required by lower tier.
	Used for data forwarding to satisfy remote read when Manager-Client pair permission levels already match.
Invalidate	Demand lower realm to forfeit write permissions and read permissions, invalidating all local copies of data.
	Used to satisfy remote write request which requires exclusive rights when remote realm already has a copy of the data.
SupplyDowngrade	Demand Data from lower realm's paired Manager. Additionally, lower realm must forfeit write permissions but can retain read permissions and data.
	Used for data forwarding to satisfy remote read when upper-tier paired Client state is forfeiting exclusive/write permissions.
SupplyInvalidate	Demand Data from lower realm's paired manager. Additionally, lower realm must forfeit write permissions AND read permissions, invalidating all local copies of data.
	Used for data forwarding to satisfy remote exclusive/write request when remote realm expects data supplied from this realm.

Lower Tier Manager to Upper Paired Client Demand Reply

SupplyAck	Response by paired Manager to complete previous Supply demand. Supplies data packet.
InvalidateAck	Response by paired Manager to complete previous Invalidate demand. Signifies realm invalidation has completed.
SupplyDowngradeAck	Response by paired Manager to complete previous SupplyDowngrade demand. Supplies data packet and signifies realm downgrade has completed.
SupplyInvalidateAck	Response by paired Manager to complete previous SupplyInvalidate demand. Supplies data packet and signifies realm invalidation has completed.