

# Hierarchical Cache Coherence Protocol Verification One Level at a Time Through Assume-Guarantee

Xiaofang Chen<sup>1</sup>, Yu Yang, Ganesh Gopalakrishnan Ching-Tsun Chou  
School of Computing, University of Utah Intel Corporation  
Salt Lake City, UT 84112 Santa Clara, CA 95054

**Abstract**—Due to the error-prone nature of modern cache coherence protocols, in all modern processor design flows these protocols are formally specified at the level of interleaving atomic transactions and model checked. Explicit state enumeration methods are almost always used for coherence protocol verification, as symbolic methods have failed to deliver advantages in this area. The move towards multicores implies that hierarchical organizations of several different cache coherence protocols will be employed in future. The product state space of all these protocols jointly operating in a multicore cache hierarchy is beyond the reach of all available explicit state model checkers. In [1], an assume guarantee technique that allowed these protocols to be handled for the first time was reported. In this approach, a method was proposed to create a set of initial abstract protocols  $\{\text{Abs } \#i\}$  where each  $\text{Abs } \#i$  simulates the given hierarchical protocol. Since the various  $\text{Abs } \#i$  depend on each other, verification consists of dealing with the set  $\{\text{Abs } \#i\}$  in an assume guarantee manner, refining  $\text{Abs } \#i$  in the process. The drawbacks of [1] were: (i) even one single  $\text{Abs } \#i$  modeled more than one cluster; in particular, portions of other clusters and directory structures were also modeled, thus still creating very large product spaces, (ii) details such as non-inclusive caching hierarchies could not be handled. This paper overcomes both these limitations, handling non-inclusive caching hierarchies, and bringing about a 95% reduction in the total state space encountered during any single explicit enumeration search, and requiring only a few such runs to finish verification.

## I. INTRODUCTION

Cache coherence protocols are central to all future advances in chip multiprocessors (CMPs), otherwise known as ‘multicores.’ Modern multiple chip-multiprocessors (M-CMP) employ extremely complex cache coherence protocols, to ensure high performance and to accommodate manufacturing realities such as unordered coherence message networks. In order to eliminate concurrency bugs from these protocols, modern industrial practice requires that finite state models of these protocols be exhaustively verified – not just debugged using semi-formal methods. This is achieved in most cases by modeling small instances of these protocols (e.g., three CPUs handling two addresses and one bit of data) in terms of interleaving atomic steps in guard/action languages such as Murphi [2] or TLA+ [3], and exploring the reachable states through explicit state enumeration. The implementation of the atomic steps is an important, but separate, problem, not addressed in this work (it may be addressed through a refinement verification [4] or synthesis [5]).

This work is supported in part by SRC Contract 2005-TJ-1318.

<sup>1</sup>Main contact: xiachen@cs.utah.edu.

Researchers have tried – but failed – to verify the interleaving models of cache coherence protocols using symbolic methods (BDD/SAT) methods. In [6], it is reported that a modest benchmark protocol with a 64-bit state vector and a few pages of description could not be verified, when the very same set of techniques could handle a CPU model with millions of state bits. The main reason for this failure is that it is very difficult to project away the state bits from cache coherence protocol models. What this means in terms of modern practice is that for industrial protocols that occupy 50 or more pages of description with hundreds of state bits, explicit state enumeration methods must cope with verification complexity.

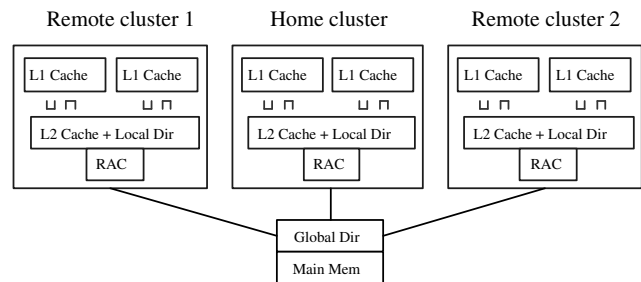


Fig. 1. A 2-level hierarchical cache coherence protocol.

Multicore chips will be organized in *hierarchies* (e.g., Figure 1). At present, it is common to see four quad-core chips forming a 16-way multiprocessor. These organizations will employ two levels of cache protocols called (i) the *intra* cluster level where the L1 caches of the CPUs will be kept coherent among themselves and with respect to a local L2 cache level, and (ii) the *inter* cluster level where the clusters themselves will be kept coherent. This is bad news, considering that the intra cluster protocol state space itself will be very large, and that any *product* of the states of three intra cluster protocols and one inter cluster protocol (Figure 1) would be unacceptably large. In our previous work [1], we presented a compositional approach for partly mitigating this problem. The workflow of the approach is shown in Figure 2.

Given an M-CMP protocol, we first built a set of abstract protocols  $\{\text{Abs } \#i\}$ . Each  $\text{Abs } \#i$  overapproximates, by construction, the original protocol (see Figure 3 which illustrates one such abstract protocol). However, it is also the case that each  $\text{Abs } \#i$  employs assumptions that will be validated only by verifying some number of  $\text{Abs } \#j$ 's that,

then, justifies these assumptions. In turn,  $\text{Abs } \#j$  employs assumptions that are justified by verifying some number of  $\text{Abs } \#i$ 's (further details of such meta-circular dependencies are discussed in Section II-A). In [1], we show that (i) each  $\text{Abs } \#i$  has far less states than the original hierarchical protocol, (ii) the additive complexity of verifying the  $\text{Abs } \#i$ 's in turn is also far less than the complexity of the original protocol. However, as will be seen from Figure 3, even one  $\text{Abs } \#i$  involves the *product* state space of one entire unit (such as ‘Home cluster’ in Figure 3) and two simplified units (such as ‘Remote clusters’ in Figure 3).

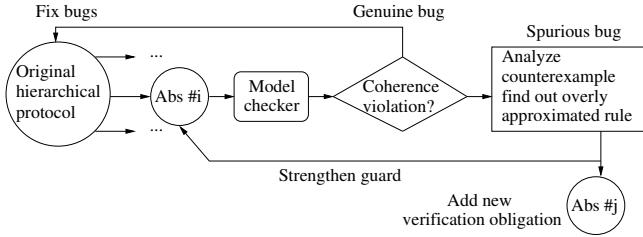


Fig. 2. The workflow of our approach.

This paper addresses the following questions:

- Can we contain the verification complexity so that we never incur more than the state space of one entire cluster? In other words, can we avoid incurring the product state space complexity, however small the “remote” unit state spaces are? This is because even if the remote units have just *two* states each, we would be dealing with *four* times the number of states in an entire unit – which is very high. In this paper we show that this is possible by *modeling the interfaces of the protocols in a particular way* (Section II-B). The result is that we are able to reduce the number of states examined in any one abstract protocol to under 5% of the number of states examined in [1].
- What other complications arise in realistic caching hierarchies and how do we handle it? Simpler M-CMP protocols use the *inclusive* caching hierarchy. That is, the content of an L1 cache is a subset of that of the L2 cache in the cluster. Protocols which use *non-inclusive* caching hierarchies are far more complex. We show that by *using history variables*, but in a novel way (Section III-C), we can extend the approach of [1].
- How can “standard tricks” be safely applied? One standard trick in most cache coherence exercises is to maintain the *latest copy* of a cache line in a book-keeping variable. In Section II-B, we show the perils of blindly extending this trick to now maintaining *intra\_copy*'s and an *inter\_copy*, and offer solutions.
- In Section II-B and III-B, we show, with examples, why the verification of a hierarchical protocol is simply not a matter of verifying each level of the hierarchy separately; we show how *imprecise records* arise, and how we deal with them.

**Related work:** A Term Rewriting method for reasoning about the correctness of hierarchical cache coherence protocols was presented in [7]. This work has not been demonstrated on protocols of realistic sizes as well as realistic features such as

the inclusion relationship between caches, unordered networks, etc. As explained in [1], our protocols were designed by consulting with an industrial expert who helped us include these features, and also additional features such as the silent dropping of cache lines which introduces additional corner cases. Thus, even in terms of the complexity of the protocols within one cluster, our protocols far exceed the complexity of popular benchmarks such as FLASH [8] and German [9]. We have already compared our work in this paper against our past work in [1]. Our paper in [1] and this paper derive their basic ideas from Chou et. al.'s work [10] which was a method for parameterized verification for non-hierarchical cache coherence protocols, and McMillan's work on compositional model checking [11]. While the use of *history variables* in program verification goes back several decades (e.g. [12], [13]), our usage of history variables is in the context of assume-guarantee reasoning, and it involves a meta-circular approach as discussed in Section III-C.

## II. VERIFYING M-CMP COHERENCE PROTOCOLS

### A. Background

In the M-CMP system verified in [1] (Figure 1), every cluster contains two processors of identical design, each with a private L1 cache. The L2 cache is shared by the two processors, and its content is a superset of that of the L1 caches, i.e. the *inclusive* caching organization is used. The remote access controller (RAC) is used by a cluster to communicate with other clusters. We model one address in our model, and associate the home cluster with the main memory for this address. In reality, the main memory is attached to every cluster; the fact there is only one memory is a consequence of the 1-address abstraction of our protocol. The two remote clusters are of identical design. The protocol used inside a cluster is a directory-based MESI protocol [14], [15], maintaining the coherence for the caches within a cluster. The local directory records which L1 cache(s) or whether the L2 cache has a valid copy, and in which specific state. The protocol used among clusters is also a directory-based MESI protocol. The global directory only has the high-level information of which cluster holds a valid copy, in which specific state. Other features of this protocol include: (i) silent-drops are supported on non-Modified cache lines, and (ii) network channels are modeled in non-FIFOed ordering, e.g. messages can arrive out of order.

In [1], we presented an approach for decomposing a hierarchical protocol into a set of abstract protocols, using abstraction and counter-example guided refinement in an assume-guarantee manner. For the protocol shown as in Figure 1, we decompose the protocol into three abstract protocols. In every abstract protocol, one detailed cluster is maintained, and the rest two are abstracted. Because the two remote clusters are of identical design, there are altogether two distinct abstract protocols. Figure 3 shows an abstract protocol where the details of the home cluster are maintained (contrast with Figure 1). “Local Dir’” represents part of the local directory “Local Dir”.

In the approach of [1], the abstract protocols are obtained by overapproximating the original protocol by projecting away

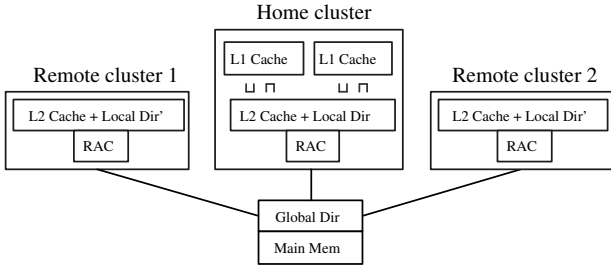


Fig. 3. One of the two abstract protocols in the previous approach.

intra-cluster details of different clusters.<sup>1</sup> The following figure shows an example of a transition before and after abstraction.

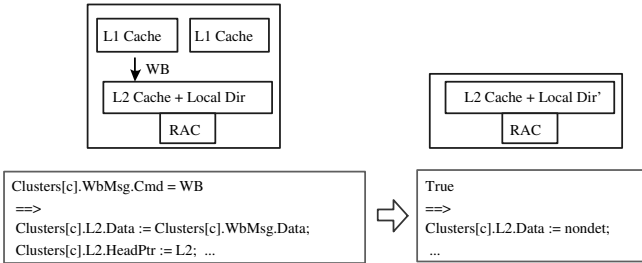


Fig. 4. A transition before and after abstraction.

In Figure 4, a writeback request from an L1 cache is received. With this request, the local directory updates the L2 cache copy, and records that the L2 cache has the latest copy in the cluster. After this cluster is abstracted, the L1 caches and the writeback network channel are removed. So for every transition which involves the details of these components, it is overapproximated. In this example, the guard is overapproximated to true, i.e. it becomes more permissive. The cache copy in the writeback request is replaced with a nondeterministic copy “nondet”, and the second statement in action is removed.

When we model check an abstract protocol, violations to coherence properties can be found, due to genuine bugs in the original protocol, or overapproximation of the abstraction. For genuine bugs, we fix them in the original protocol, and re-generate the abstract protocols. For spurious bugs, we do the refinement.

For the example in Figure 4, it is clear that the abstracted transition can easily introduce coherence violations, because the L2 cache can update its copy arbitrarily. For this example, we can strengthen the guard with the formula that  $\neg (P1)$  the L2 cache is in exclusive or modified state. Also, we add a new verification obligation to one of the abstracted protocols, ensuring that whenever there is a writeback request in a cluster,  $(P1)$  must hold. This is a characteristics of the inclusive caching organization, which does not necessarily hold for non-inclusive ones, as will be shown in Section III. The question of adding the verification obligation to which abstract protocol, depends on which one maintains the details for this cluster. Since each abstract protocol maintains the details of a different

<sup>1</sup>The visual analogy of how large mirror telescopes are made of mirror segments may help.

part of the original protocol, the abstract protocols depend on each other to justify the refinement.

The problem with the above decomposition approach is that, every abstract protocol still contains one intra- and one inter-cluster protocols. So its state space is still very large. For example, we show that (see Section II-E) one abstract protocol has about 0.6 billion of state space, with 40-bit of hash compaction. This somewhat reaches the upper limit of the number of states currently explored by an explicit state model checker in a day or two on a powerful machine. Also, there are duplicated behaviors among different abstract protocols. The proposed method slashes the states down to 5% of the original, and the runtimes come down from a day to under 10 minutes.

### B. A New Decomposition Approach

As the title of the paper suggests, our new approach involves abstract protocols that involve either the intra- or the inter-cluster protocol, but never both. The techniques used to build and verify the abstract protocols are similar to before: (i) every abstract protocol is obtained by overapproximating the original protocol, through projecting away different components; (ii) in the counter-example guided refinement for abstract protocols, the formula used to strengthen the guard of a transition, will be proved as an invariant with assume-guarantee reasoning; (iii) we use simulation to prove the soundness of our approach. The key difference with our previous approach lies in modeling the interfaces between the intra- and the inter-cluster protocols.

In [1], the interfaces were modeled in a “tightly-coupled” manner. More precisely, some Murphi transitions involve the details of *both* the intra- and the inter-cluster protocols. For instance, in one transition, the conditions under which the transition is enabled are that a local directory receives a request from an L1 cache asking for a cache copy, the L2 cache does not have the copy, and the local directory is not busy when the request is received. Once the transition is enabled, the local directory is set to busy, the RAC of the cluster is also set to busy, and a request asking for a cache copy is placed on the network channel used among the clusters.

If hierarchical protocols are modeled in this manner, it is not easy to see how to decompose the protocol into “per-level” abstract protocols. For example, in the transition we just described, after the cluster is abstracted, the guard becomes that the L2 cache does not have the copy. In the refinement process, we need to strengthen the guard using the fact that the RAC of the cluster must not be busy. To ensure that the guard strengthening is sound, we have to prove that whenever the local directory is not busy, the RAC of the cluster must not be busy. *This requires one abstract protocol to maintain the details of a whole cluster, including the L1 and L2 caches, the local directory and the RAC.* Such abstractions prevent us from decomposing the protocol “per-level.”

In our current approach, interfaces are modeled as “loosely-coupled.” In more detail, a transition is allowed to involve the details of either an intra- or the inter-cluster details, including the interface between them. It is not allowed to involve the

details of both levels. Writing verification models in this manner is not difficult, and does not mask any of the corner cases that were there before. For the example in the above, the transition can be divided into two transitions in our new protocol: (i) the first sets the local directory to busy, and puts the request on the interface; (ii) when the request on the interface is detected, and the RAC of the cluster is free, the second transition forwards the request to outside the cluster, clears the interface, and sets the RAC to busy.

Other than a better interface characterization, in developing the new decomposition approach, we also improved the way how coherence properties should be represented for hierarchical protocols. In the M-CMP protocol presented in [1], the intra- and inter-CMP protocols each uses an auxiliary variable to keep track of the latest copy inside itself. Let these variables be *intra\_copy* and *inter\_copy* respectively. For each cluster, *intra\_copy* is initially undefined. It is updated when a cluster receives a reply which grants a valid copy, receives an invalidate request from outside the cluster, or an L1 cache inside the cluster updates its copy in exclusive or modified state. On the other hand, *inter\_copy* is initially set to the copy in the main memory. It is updated when an L2 cache receives a writeback or shared writeback request, or a reply which grants a valid copy from inside a cluster. So the values of *intra\_copy* and *inter\_copy* depend on each other.

In developing the new decomposition approach, we discovered that these latest copies need to be constrained carefully on the interface. Figure 5 shows a simple scenario of a buggy protocol.

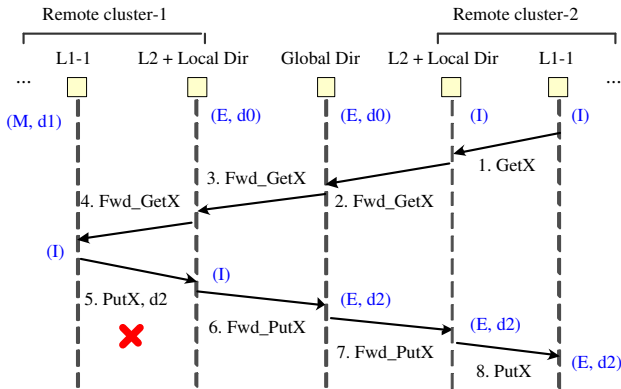


Fig. 5. A scenario of a non-coherent cache protocol.

In Figure 5, remote cluster-1 initially has a modified copy  $d1$  in L1 cache-1, and *intra\_copy* is just  $d1$ . In Step 1, an L1 cache of remote cluster-2 generates a GetX request, and this request is forwarded to the local directory of remote cluster-2, then to the global directory, and finally to L1 cache-1 of remote cluster-1 in Step 4. In Step 5, L1 cache-1 grants the request, sets itself to invalid, but mistakenly replies with some other data  $d2$ . On receiving the reply in Step 6, *inter\_copy* is set to  $d2$ , and *intra\_copy* is undefined. Thus, the latest copy in the system is lost.

For the example in Figure 5, if *intra\_copy* and *inter\_copy* are not constrained in an appropriate way, model checking cannot detect the bug. This also shows that when two protocols

are coupled hierarchically, more corner cases can come up due to the interfaces between the protocols. Specifically for this scenario, in the beginning of Step 6, a constraint that *inter\_copy* is the same as *intra\_copy* of remote cluster-1 should be asserted. Or, in summary, a unique latest copy should be used in the whole system. The value for this unique latest copy is initially set to that in the main memory, and it is updated only when an L1 cache updates its copy in exclusive or modified state.

### C. Details of the New Approach

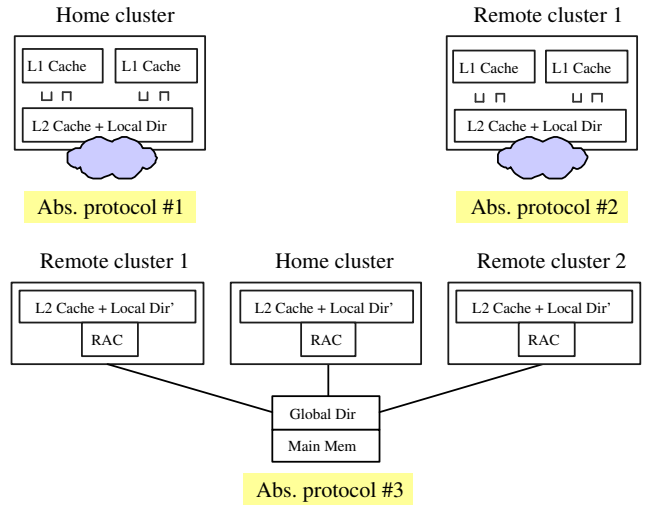


Fig. 6. The three abstract protocols from the new decomposition approach.

We still use Figure 1 as our current M-CMP protocol. For this protocol, our new approach will decompose it into four abstract protocols. Because the two remote clusters are of identical design, the two abstract intra-cluster protocols are the same. So Figure 6 only shows three distinct abstract protocols. The first one only contains the intra-cluster protocol of the home cluster. The second one is similar, except that it is for a remote cluster. The clouds in the figure represent the environment outside a cluster. The environment will nondeterministically generate requests or replies to the cluster. It can be automatically obtained from abstraction. Finally, the third one contains the inter-cluster protocol which is used among the clusters. In the following, we will illustrate how each abstract protocol is created.

1) *Abstraction*: As in [1], the derivation of each abstract protocol from the original protocol involves the variables abstraction and the transition relation and invariants abstraction. Abstraction for state variables is simply a projection. For example, to create the first abstract protocol, the RAC of the home cluster is projected away, and the rest of the system other than the home cluster is also dropped. For transition relation, given a transition in the original protocol, one or more corresponding transitions will be created. In more detail, consider a rule guarded transition in the form of *guard*  $\rightarrow$  *action* in a hierarchical protocol. If a sub-expression in *guard* contains a variable that has been projected away, we replace the sub-expression with true. For statements of the

form  $v := E$  in *action*, (i) if  $v$  has been dropped, then the assignment is dropped; (ii) if  $E$  contains variables that have been dropped, then  $E$  will be replaced with a nondeterministic value over the type of  $E$ . Other statements and invariants are processed similarly.

The above process ensures that every abstract protocol is an overapproximation of part of the original system. For example in Figure 6, the first one is an overapproximation of the intra-cluster protocol of the home cluster, and others are similar. This ensures that in our approach, by composing all the abstract protocols, it can cover every reachable state in the original protocol.

2) *Counter-Example Guided Refinement*: The refinement process is the same as before. For every spurious bug of an abstract protocol, we strengthen the guard of the overly approximated transition, and also add a new verification obligation to one of the abstracted protocols. For the abstracted protocols in Figure 6, the first two will add their verification obligations to the third one. On the other hand, the third one will add some of its verification obligations to the first, and add the rest to the second. So abstract protocols depend on each other to justify the refinement.

Now, consider one example of the refinement process. Suppose we model a unique latest copy in the system, denoted as *latest\_copy*. This variable is updated only when an L1 cache updates its copy in exclusive or modified state. After abstraction, it becomes that *latest\_copy* can be updated arbitrarily in the third abstract protocol, because the details involving L1 caches are projected away. Violations to data coherence properties can be easily detected due to the overapproximation. To do the refinement, we strengthen the rule to be: when the L2 cache of a cluster is exclusive or modified, can *latest\_copy* be updated. In the meanwhile, we add a verification obligation to the first and second abstract protocols, ensuring that when an L1 cache updates its copy, the L2 cache of the cluster must be exclusive or modified. This is the characteristics of the inclusive caching organization.

#### D. Soundness of The Approach

From the above, we can see that the refinement process is “circular”: the abstract intra-cluster protocol depends on the abstract inter-cluster protocol for the soundness of the guard strengthening, and vice versa. In fact, the soundness of the refinement can be justified by a temporal induction based assume guarantee reasoning. The intuitive reason is that all the verification obligations which are added, are checked from the initial states of the abstract protocols, step by step, and every abstract protocol is an overapproximation of part of the original protocol. More details of the formal proof can be found in [1].

Finally, we need to prove that if all the abstract protocols can be verified coherent, the hierarchical protocol must be coherent. This takes the same proof as in [10]. That is, given an M-CMP protocol  $M$ , and the set of abstract protocols  $M_i$ 's which are obtained using our approach, there exists a simulation relation between  $M$  and each  $M_i$ . Also, every reachable state of  $M$  is contained in the reachable states of the

composition of all the abstract protocols. Moreover, for every coherence property  $p$  in  $M$ , there is a corresponding coherence property  $p_i$  in each  $M_i$ . Every  $p_i$  is obtained by abstracting (see Section II-C.1)  $p$  in  $M_i$ , and they are strong enough to represent  $p$ . In our M-CMP example, these  $p_i$ 's fall into two categories. The first is that some  $p_i$  is exactly the same as  $p$ , and all the others are just true. The second is that  $p$  is exactly in the form of  $\wedge_i p_i$ , i.e. the conjunction of all the  $p_i$ 's. So for every  $p$ ,  $\wedge_i p_i \Rightarrow p$  holds.

#### E. Experimental Results

Figure 7 shows the experimental results of the M-CMP coherence protocol, using the traditional model checking, the previous approach, and the current approach. The first three experiments were performed on an Intel IA-64 machine, and the last three were performed on a PC with an Intel Pentium CPU of 3.0GHz. 40-bit hash compaction was used in all the experiments.

		# of states	Model check time (sec)	Use mem (GB)	Model check passed
Classical approach	Full model	> 438,120,000	> 125,410	18	Non conclusive
Previous approach	Abs. model 1	284,088,425	44,978	18	Yes
	Abs. model 2	636,613,051	66,249	18	Yes
Current approach	Abs. model 1	1,500,621	270	1.8	Yes
	Abs. model 2	574,198	50	1.8	Yes
	Abs. model 3	198,162	21	1.8	Yes

Fig. 7. Verification complexity using different approaches.

Here, we employed the Murphi model checker for the experiment. In the above table, model checking on the full model using the classical approach failed after more than 0.4 billion of states, due to state explosion. The previous decomposition approach was able to verify the protocol. However, the state space of each abstract protocol is still large. Using the new approach, the M-CMP protocol can be easily verified, with less than 2GB of memory. It can reduce more than 95 percent of the state space of the original protocol.

### III. NON-INCLUSIVE PROTOCOL VERIFICATION

Till now, the M-CMP protocols we have discussed use the *inclusive* caching hierarchy. That is, the content of the L1 cache is a subset of that of the L2 cache on the same cluster. Other than inclusive, there are two more caching hierarchies: *exclusive* and *non-inclusive*. Exclusive means that any block that is present in an L1 cache cannot be in the L2 cache on the cluster. Non-inclusive lies between inclusive and exclusive: overlaps and without containment are allowed. For illustration, some processors of the Intel Pentium family use non-inclusive caches, and processors of AMD Athlon and Operton use exclusive caches.

For inclusive caches, upon a cache miss in an L1 cache that hits in the L2, the cache controller only needs to copy the data to the missing L1 cache. On the other hand, when a block is replaced in the L2 cache due to a conflict or capacity miss, the same block must be evicted from all the L1 caches

of the cluster. For exclusive caches, the effective cache size of the system can be the sum of the L1 and L2 caches. As non-inclusive caches can cover both inclusive and exclusive caches, we will only focus on non-inclusive caches in the following.

### A. Verification Problems Due to Non-inclusion

For non-inclusive protocols, two categories of problems make the verification hard. First, cache coherence properties of non-inclusive protocols may have to involve L1 caches from different clusters. For example, consider an oftenly used coherence property: no two caches can write to the same address concurrently. For inclusive caches, we can represent this property using two verification obligations: (i) no two clusters can have their L2 caches both be exclusive or modified, and (ii) in every cluster, no two L1 caches can both be exclusive or modified. Each of these verification obligations can be model checked in abstract protocols using our new decomposition approach. In contrast, for non-inclusive caches, when two L1 caches from different clusters are both exclusive or modified, their corresponding L2 caches may not have the copy. Since every of our abstract protocols only maintains the details of at most one cluster, it is not straightforward how to represent the property that no two L1 caches from different clusters can be both exclusive or modified.

Second, for error traces corresponding to spurious bugs in abstract protocols, it is not straightforward how to refine the overapproximation. This is because the L2 cache may not contain a valid copy of the cache line which is present in some L1 cache of the cluster. For example, consider the following scenario. Initially, an L1 cache has an exclusive copy to writeback, and the L2 cache on the cluster does not contain this block. After the writeback request is received, the L2 cache transits to exclusive state with a valid copy. When this transition is abstracted in the inter-cluster protocol, it becomes that an L2 cache copy can change from invalid to exclusive arbitrarily. Clearly, this is a very coarse overapproximation and it can easily lead to coherence failures.

The above are problems which exist especially in non-inclusive hierarchical coherence protocols. In the next section, we will describe an approach for solving these problems. Our approach uses a variant of *history variables* [12], [13], the variation being that the value of the history variables is also determined in an assume-guarantee manner. For the experimental validation of this approach, we created a non-inclusive variant of the benchmark protocol. This new protocol is more complex than the inclusive protocol, and hence it can not be model checked using traditional approaches.

### B. Protocol Details

Our benchmark protocol [16] has the same configurations as the inclusive protocol presented in Section II-A. The intra- and inter-cluster all use an invalidation-based directory ESI protocol [15]. For the non-inclusive M-CMP protocol, we assume that when an L2 cache line is swapped out, the local directory of the cluster is also swapped out. This assumption may not be very practical for real coherence protocols. However, it makes

the verification problem harder, and forces us to come up with new verification techniques.

For the network channels used within a cluster, other than the ones used in the inclusive protocol, there also exists a set of broadcasting channels. These channels are used when there is a cache miss in the L2 and the local directory has no information of whether some L1 has a valid copy or not. After a broadcast, if a reply containing a valid copy is received, the reply will be forwarded to the requesting L1 cache. Otherwise, the request will be forwarded to the global directory.

The characteristics of non-inclusive of our protocol can also make the local directory have an imprecise record of a cache line. The following figure shows a simple scenario of how the imprecision can happen.

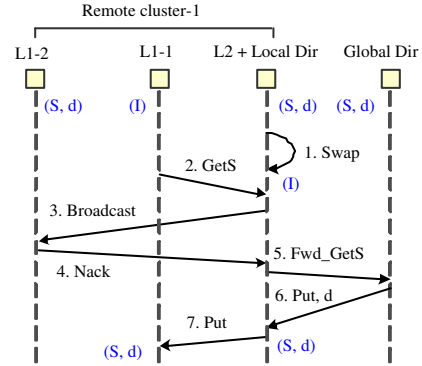


Fig. 8. Imprecise state record in the local directory.

In Figure 8, initially an L1 cache and the L2 cache in a remote cluster has a shared copy and it is recorded in the local directory. In Step 1, the L2 cache line is swapped out, and the record in the local directory is also dropped. In Step 2, another L1 cache in the same cluster requests a shared copy. As the local directory has no record about this line, the request is broadcast inside the cluster in Step 3. The L1 caches NACK this request in Step 4, as it is not safe for a shared copy to supply its data because the broadcast request can be interleaved with an invalidate request coming from outside the cluster. In Step 5, the request is forwarded to the global directory, and it is granted in Step 6. At this time, the local directory has lost the information that an L1 cache already has a shared copy. Such imprecision can lead to coherence violations for certain cache coherence properties, because subsequent invalidations will miss the shared copies in some L1 caches.

### C. Infer “Exclusive”

Now we will present how the verification problems due to non-inclusive can be solved. Given a cluster in which the L2 cache does not have a valid copy, we can infer if there is an exclusive or modified copy on the cluster in two ways. One is to infer from outside the cluster, i.e. the global directory and the network channels used among clusters. The other is to infer from inside the cluster, including the L1 caches and the network channels used within the cluster. These two approaches are similar. In the following, we will describe the second one in detail.

We still use Figure 1 to represent a non-inclusive M-CMP protocol, and use the new decomposition approach for the verification. The difference is that we will use the intra-cluster abstract protocols to infer whether there is an exclusive or modified copy in a cluster, and this inference will be used by the inter-cluster abstract protocol. In more detail, we add an auxiliary variable of boolean type for each cluster in the hierarchical protocol. Let this variable be  $IE$  (implicit exclusive). For every cluster,  $IE$  will be defined in the abstract intra-cluster protocol. Initially,  $IE$  is set to false. It is defined to be true if one of the following conditions holds:

- If an L1 cache has an exclusive or modified copy, or
- If the broadcast channel contains a valid reply, or
- If a network channel contains a reply with an exclusive copy, or
- If there is a writeback or shared writeback request, or
- If there is an exclusive ownership transfer request

When the value of an  $IE$  is true, it means that the cluster must have an exclusive or modified copy in the cluster, somewhere other than in the L2 cache.

Now with  $IE$ s, the first problem mentioned in Section III-A can be solved similarly as in inclusive protocols. For example, consider the coherence property that no two L1 caches in different clusters can write to the same address concurrently. Now this property can be represented as – no two  $IE$ s from different clusters can be both true, and no two L2 caches can be both exclusive or modified.

For the second problem, we can now constrain overly approximated transitions using  $IE$ s. For the example which can change an invalid L2 cache line to exclusive arbitrarily, it now can be strengthened as: the L2 cache can change from invalid to exclusive, if  $IE$  is true for the cluster. As usual, to ensure that the strengthening is sound, we add a verification obligation to one of the abstract intra-cluster protocols. In this case, the verification obligation requires that when a writeback request is on the way from an L1 cache to the L2 cache in a cluster, the  $IE$  must be true.

#### D. Experimental Results

For this non-inclusive hierarchical protocol, we have used the traditional and the current approaches for checking coherence. Figure 9 shows the experimental results using these approaches. As for the inclusive protocol, 40-bit hash compaction was used in all the experiments. The first experiment was performed on an Intel IA-64 machine, and the last three were performed on a PC with an Intel Pentium CPU of 3.0GHz. Again, the Murphi model checker was employed for the experiments.

From the table, we can see that the current approach can reduce the verification complexity, in terms of state space and runtime, by more than 95 percent.

## IV. CONCLUSION AND FUTURE WORK

Coherence is a particular challenge for M-CMP systems, as usually they are more complex and have more corner cases than non-hierarchical protocols. In our previous work,

		# of states	Model check time (sec)	Use mem (GB)	Model check passed
Classical approach	Full model	> 473,260,000	> 161,398	18	Non conclusive
	Abs. model 1	4,070,484	770	1.8	Yes
Current approach	Abs. model 2	2,424,719	250	1.8	Yes
	Abs. model 3	2,424,719	248	1.8	Yes

Fig. 9. Verification complexity using different approaches.

we presented a compositional approach for verifying an inclusive directory-based M-CMP protocol, using abstraction and assume-guarantee reasoning. However, the state space of abstract protocols is still very large. In this paper, we present a new decomposition approach which models hierarchical protocols using a better interface characterization. Every abstract protocol resulted from this approach only contains one instance of coherence protocols, and the approach is still sound. Furthermore, we also explored how our approach can be extended to verify non-inclusive M-CMP coherence protocols, using a variant of history variables. The experimental results show that our new approach is very effective in reducing the verification complexity.

Our method presented earlier is implemented as a manual form of abstraction and assume-guarantee reasoning. Currently, we are mechanizing the abstraction process. That is, given a hierarchical protocol and the variables to be projected away, we try to automatically create an abstract protocol. We plan to automate the spurious error trace recognition and counter-example guided refinement. It will be important to understand how to effectively combine our methods with automatic learning algorithms found in current model checkers.

## REFERENCES

- [1] X. Chen, Y. Yang, G. Gopalakrishnan, and C. T. Chou, “Reducing verification complexity of a multicore coherence protocol using assume/guarantee,” in *Formal Methods in Computer Aided Design*, 2006.
- [2] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, “Protocol verification as a hardware design aid,” in *IEEE Intl. Conference on Computer Design: VLSI in Computers and Processors*, 1992.
- [3] L. Lamport, “Specifying concurrent systems with  $tl_+^+$ ,” *Calculational System Design*, 1999.
- [4] X. Chen, S. German, and G. Gopalakrishnan, “Transaction based modeling and verification of hardware protocol implementations,” Under submission. Available upon request.
- [5] Arvind, “Bluespec: A language for hardware design, simulation, synthesis and verification,” in *MEMOCODE*, 2003.
- [6] K. McMillan and N. Amla, “Automatic abstraction without counterexamples,” in *Technical Report of Cadence*, 2003.
- [7] X. Shen and Arvind, “Specification of memory models and design of provably correct cache coherence protocols,” MIT, Tech. Rep., 1997.
- [8] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. G. J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, “The stanford flash multiprocessor,” in *Proceedings of the 21st Intl. Symposium on Computer Architecture*, 1994, pp. 302–313.
- [9] S. German, “Tutorial on verification of distributed cache memory protocols,” in *Formal Methods in Computer Aided Design*, 2004.
- [10] C. T. Chou, P. K. Mannava, and S. Park, “A simple method for parameterized verification of cache coherence protocols,” in *Formal Methods in Computer Aided Design*, 2004.
- [11] K. McMillan, “Verification of infinite state systems by compositional model checking,” in *Correct Hardware Design and Verification Methods*, 1999.
- [12] E. M. Clarke, “Proving the correctness of coroutines without history variables,” in *ACM Southeast Regional Conference*, 1978.

- [13] M. Clint, "Program proving: Coroutines," in *Acta Informatica*, 1973.
- [14] M. Papamarcos and J. Patel, "A low overhead coherence solution for multiprocessors with private cache memories," in *Proc. 11th Annual Int'l Symposium on Computer Architecture*, 1984.
- [15] D. Culler, J. Singh, and A.Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1998.
- [16] [Http://www.cs.utah.edu/~xiachen/hldvt07\\_submission](http://www.cs.utah.edu/~xiachen/hldvt07_submission).