A General Compositional Approach to Verifying Hierarchical Cache Coherence Protocols

Xiaofang Chen and Ganesh Gopalakrishnan

UUCS-06-014

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

November 27, 2006

Abstract

Modern chip multiprocessor (CMP) cache coherence protocols are extremely complex and error prone to design. Modern symbolic methods are unable to provide much leverage for this class of examples. In [1], we presented a method to verify *hierarchical* and *inclusive* versions of these protocols using explicit state enumeration tools. We circumvented state explosion by employing a meta-circular assume/guarantee technique in which a designer can model check *abstracted* versions of the original protocol and claim that the real protocol is correct. The abstractions were justified in the same framework (hence the meta-circular approach). In this paper, we present how our work can be extended to hierarchical *non-inclusive* protocols which are inherently much harder to verify, both from the point of having more corner cases, and having insufficient information in higher levels of the protocol hierarchy to imply the sharing states of cache lines at lower levels. Two methods are proposed. The first requires more manual effort, but allows our technique in [1] to be applied unchanged, barring a guard strengthening expression that is computed based on state residing outside the cluster being abstracted. The second requires less manual effort, can scale to deeper hierarchies of protocol implementations, and uses history variables which are computed much more modularly. This method also relies on the meta-circular definition framework. A non-inclusive protocol that could not be completely model checked

even after visiting 1.5 billion states was verified using two model checks of roughly 0.25 billion states each.

1 Introduction

Modern chip multiprocessors (CMP, or "multicores") employ extremely complex cache coherence protocols which must not contain concurrency bugs. Modern symbolic methods are unable to provide much leverage for this class of examples, as the state bits in these protocols cannot be easily projected away. There are examples where SAT-based methods could handle ordinary circuits with millions of bits, but could not finish on simple cache coherence protocols [2]. Explicit state methods, when applied to industrial scale protocols, do not finish in realistic amounts of time (e.g., a day or two) even for protocol instances modeling about 3 CPUs, and are known to exceed realistic storage limits (e.g., 4GB of memory) even when running with reasonable degrees of lossy state compression enabled (e.g., with 40-bit state signatures stored instead of full state vectors).

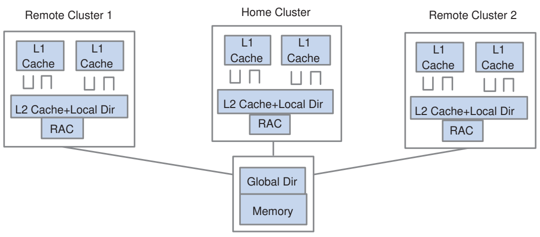

Figure 1: A 2-level cache coherence protocol for MCMP systems.

We now introduce some terminology. Referring to Figure 1, the term *Inclusive* means that the content of the L1 cache is a subset of that of the L2 cache on the same cluster. *Exclusive* means that any block that is present in an L1 cache cannot be present in the L2 cache in the same cluster. *Non-inclusive* lies between Inclusive and Exclusive: overlaps, without containment, are allowed. For illustration, some processors of the Intel Pentium family use non-inclusive caches, and processors of AMD Atholon and Operton use exclusive caches.

For inclusive caches, the local directory at the L2 cache knows which L1 cache(s) have valid copies. 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. However, 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. For non-inclusive caches, the L2 controller often has to "snoop" across all L1s, as we shall see. If we can effectively verify hierarchical cache coherence protocols with the non-inclusive caching policy, we can in fact verify hierarchical protocols with *any* caching policy (inclusive, exclusive, or non-inclusive).

In [1], we considered a complex *inclusive* cache coherence protocol benchmark that we had developed with the help of an industrial collaborator. Even though this protocol instance only had three clusters (or CPU sockets) with each cluster containing two CPUs, and we modeled only one cache line (a typical approach that we also follow here), we could not model check the protocol directly within reasonable space/time limits. The meta-circular assume/guarantee method that we proposed in [1] did succeed in proving this inclusive protocol correct. At that time, we did not know whether the technique would extend to *non-inclusive* protocols in a straightforward manner.

In this paper, we show that while the technique would extend, it is not straightforward to do so, especially when the non-inclusive protocol hierarchy gets any deeper. We present a new technique that is much more modular, and promises to extend to arbitrary depths of protocol hierarchy, and arbitrary variations of inclusive versus non-inclusive across protocol hierarchy levels. The modularity is achieved by using a variant of *history variables* [3, 4], the variation being that the value of the history variables must also be determined in a meta-circular manner.

For the experimental validation of our method, we created a non-inclusive variant of the benchmark protocol in [1]. This new protocol has even more states than the original protocol, and hence could not be model checked using traditional approaches: the search was aborted after visiting 1.5 billion states. Using our new technique, we could model check two abstract models that, each, have only roughly 0.25 billion states, and formally claim that the original protocol has no errors. Our contributions are: (i) evidence of verifying an example more complex than previously verified in the area of hierarchical cache coherence protocols (the example can serve as a valuable benchmark for others), and (ii) an understanding of *how* the meta-circular abstraction can be set up so that the user effort is minimized, and the verification complexity is contained.

Related work: Other than our own past work [1] and this paper, nobody else, to our knowledge, has verified hierarchical directory based protocols of non-trivial complexity. The details addressed in our benchmark protocol (Section 2) are important to point out, as the success of a method is directly measured by them; our protocol far exceeds the complexity of popular benchmarks such as FLASH [5]. Our paper in [1] and this paper derive their basic ideas from Chou et.al.'s work [6] which was a method for parametrized verification for non-hierarchical cache coherence protocols. McMillan's work on compositional model checking [7] also takes a similar approach. While the use of *history variables* in program verification goes back several decades (e.g., [3,4]), our particular usage of history variables in the context of our abstraction, and how it helps dramatically simplify the verification of hierarchical protocols has not been pointed out before.

Roadmap: Section 2 presents our non-inclusive protocol to some detail. Section 3 presents the challenges due to non-inclusive protocols. Section 4 presents our two approaches to refinement. Section 5 presents the assume/guarantee verification argument embodied in the new approach, and the conclusion follows.

2 A 2-level non-inclusive cache coherence protocol[1]

Our benchmark protocol [8] is composed of two levels. The level-1 protocol is *intra cluster*. It is a directory-based MSI protocol [9], maintaining cache lines in three states: modified, shared, and invalid. The level-2 protocols is *inter cluster*. It is also a directory-based MSI protocol. The global directory always records which specific cluster has a valid copy in what state. As is typical in model-checking based verification for coherence, only one address is modeled. With respect to this address, there are three NUMA (Non-Uniform Memory Access) clusters: one home cluster and two identical remote clusters. Each cluster has two symmetric L1 caches, an L2 cache and a local directory. "RAC" stands for the controller to communicate with other clusters and the global directory. The main memory in reality can be attached to every cluster. The fact there is only one memory is a consequence of the 1-address abstraction of our protocol.

Three types of network channels are available between each L1 and the L2 cache pair in level-1. One is the set of request/reply channels, represented as "ReqMsg[]" in the following. The second is the set of invalidation channels, used for invalidation and the corresponding acknowledgments, represented as "InvMsg[]." The third is the set of broadcasting channels, represented as "SnoopMsg[]". They are used when there is a cache miss in the L2 but the local directory has no record of which L1 cache has a valid copy. These three types of channels are separated out, because all channels for the same L1-L2 pair can transfer messages at the same time, and these messages do not have to maintain FIFO ordering. (In hardware, these channels can share one set of physical channels.)

As an example, when a request (from inside or from outside a cluster) is received by the local directory which has no record for the line, the level-1 directory protocol will first broadcast the request to both the L1 caches. If a reply containing a valid copy is received, the reply is forwarded to the requesting L1 (or to the outside requester). If no reply contains

[1]This section may be skimmed on first reading.

a valid copy, the request will either be forwarded to the global directory or be NACKed, according to where the request is generated.

Other than these channels used for each L1-L2 pair, there are two other types of messages which could only be used by one of the L1 caches at any time. "WbMsg" is for the write-back request, i.e. an L1 cache writes back its exclusive copy to the L2. "ShWbMsg" is for the shared-writeback request, i.e. an L1 cache with an exclusive copy supplies the data to a forwarded request, modifies itself to be either shared or invalid (dependent on the requesting type), and at the same time notifies the local directory about these modifications.

The non-inclusiveness of the L2 cache allows silent dropping of L2 cache lines under certain conditions, and at the same time removing the corresponding records in the local directory. For exclusive lines in L2, silent-drop is allowed when the most recent copy of the line is in one L1 cache in the cluster. This could happen for example, when an L1 cache initially requests an exclusive copy, the local directory gets the copy from the global directory and then replies to the requesting L1 cache. For shared lines in L2, silent-drop is allowed if they are not "the dirty copies from the perspective of the whole system" [2]. For all the other cases when a valid line is in L2, writeback to the main memory is required before the data can be dropped.

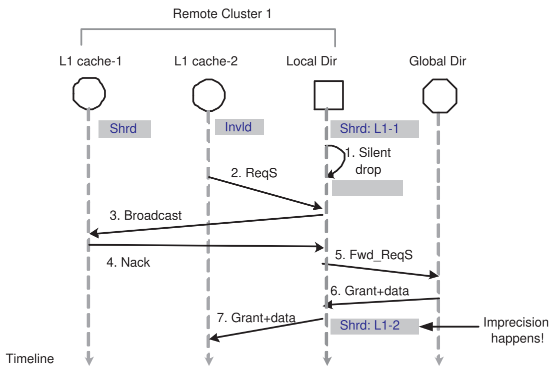

Figure 2: Imprecise state record in the local directory.

The characteristics of non-inclusion could make the local directory have an imprecise record of a cache line. Figure 2 shows a simple scenario of how the imprecision could happen. Initially, an L1 cache and the L2 cache in remote cluster-1 has a shared copy and it is recorded in the local directory. In Step 1, L2 silently drops the cache line and the record in the local directory is also swapped out. In Step 2, another L1 cache requests a shared

[2]This could happen for example, after the cluster obtains an exclusive copy, an L1 cache inside the cluster requests a shared copy and it is directly granted. At this time, the state of the local directory becomes shared.

copy. As the local directory has no record about this line, the request is broadcasted inside the cluster in Step 3. The L1 cache-1 NACKs this request in Step 4, as it only has a shared copy. The reason is that the broadcasting messages "SnoopMsg[]" could be interleaved with an invalidation messages "InvMsg[]" coming from outside, in any order. So it is *not safe for a shared copy to supply its data when receiving a broadcasting request*. These are all subtle corner cases that do not arise with inclusive protocols.

Continuing the scenario, in Step 5, the request is then forwarded to the global directory, and it is granted in Step 6. At this time, because the local directory has lost the information that L1 cache-1 has a shared copy, it can only record that the L1 cache-2 has a valid copy for this line. Such imprecision could lead to coherence violations for certain cache coherence properties, as subsequent invalidations will forget to invalidate the copy in L1 cache-1.

Such bugs are avoided in our protocol through a conservative assumption. When the global directory receives the forwarded request in Step 5, it can realize that remote cluster-1 already has a shared copy. So in addition to the reply and the data supplied in Step 6, a tag is also attached in the message indicating possible imprecision. When the local directory of remote cluster-1 receives this message, it will record that all the L1 caches inside the cluster have a valid shared copy, thus avoiding the imprecision.

3 Non-inclusive Coherence Protocol Verification

In our previous work [1], we developed a compositional approach to verify a 2-level inclusive coherence protocol in a hierarchical system similar to that shown in Figure 1. Essentially, three abstract protocols are built from the overall hierarchical protocol M; they are $\tilde{M}_1$, $\tilde{M}_2$, and $\tilde{M}_3$. ($\tilde{M}_2$ and $\tilde{M}_3$ happen to be identical due to the symmetry between the two remote clusters). The *home* cluster of $\tilde{M}_1$ is identical to that of M. Likewise, the Global Dir and Memory at the root node are also identical to that of M. However, the remote clusters of $\tilde{M}_1$ have their L1 caches and part of their local directories removed. This means that all inputs coming into L2 Cache + the retained part of the Local Dir of these units now comes completely unconstrained by the state of the removed pieces. $\tilde{M}_2$ is similar to $\tilde{M}_1$, except that the *remote* cluster-1 is kept unabstracted, while the home and remote cluster-2 are abstracted. In effect, each $\tilde{M}_i$ is constructed from M by simply projecting out (unconstraining) selected global variables, and correspondingly overapproximating the protocol transitions. Different variables are projected out for each $\tilde{M}_i$, and therefore, each $\tilde{M}_i$ presents a different overapproximated view of M. Thereafter, counterexample guided refinement is used for verifying each $\tilde{M}_i$.

The underlying logic is that we first overly approximate the protocol, and then reduce the degree of overapproximation and recover some semblance of proper behavior through guard strengthenings. In particular, we strengthen the transition guards in $\tilde{M}_i$ and at the same time, add verification obligations to one of $\tilde{M}_1$, $\tilde{M}_2$ or $\tilde{M}_3$, depending on where those strengthenings can be evaluated! For example, we may strengthen a guard g in $\tilde{M}_1$ to become $g \wedge p$, but add the verification obligation $g_0 \Rightarrow p$ to $\tilde{M}_2$, because p involves variables present in $\tilde{M}_2$, but not $\tilde{M}_1$. Here, g_0 is the guard of the corresponding rule of g in M. This causes mutual (apparent) circular dependencies between the systems. We formally proved in [1], through induction, that the product of $\tilde{M}_i$ simulate M; thus, once the three simpler protocols are verified, the original protocol can be concluded to be correct (with respect to coherence, in our case).

Difficulties due to Non-inclusive Protocols: For non-inclusive protocols, a valid cache line at level L1 may not exist at level L2. Also, the local directory may not have a record of the presence of the cache line in L1. Two categories of problems make the verification of the non-inclusive case hard. First, most ordinary formulations of the cache coherence property in non-inclusive protocols involve the states of L1 cache lines in *different* clusters (the property cannot be directly formulated in terms of L2 states, as is possible with inclusive protocols). For example, one such property could be that no two lines with the same address can be in the exclusive state concurrently. Formally, it can be represented as $StateCohProp$ in the following:

d	:	The number of data bits in a cache line
m	:	The number of clusters in a MCMP system
n	:	The number of L1 caches in one cluster
Basic cache line	:	$CL = \{$ state: {M,S,I}; data: array $1..d$ of bits;$\}$
Clusters	:	$P = \{p_1, \ldots, p_m\}$
L1 caches	:	$\forall i \in [1..m],\ L1(p_i) = \{l1_{i1}, \ldots, l1_{in}\}$
L2 caches	:	$\forall i \in [1..m],\ L2(p_i) = \{l2_i\}$
$StateCohProp$	:	$\forall i, j \in [1..m]:$ $(i \neq j \implies \neg(l2_i.state = Excl \wedge l2_j.state = Excl)) \wedge$ $(\forall k, l \in [1..n] : (k \neq l \implies \neg(l1_{ik}.state = Excl \wedge l1_{jl}.state = Excl)))$

To verify this property, we need to check the states of *all* the cache lines in the L1 and L2 caches in the system. For inclusive coherence protocols, the corresponding property can simply be represented in each abstracted protocol $\tilde{M}_p$ as

$$\begin{aligned}\forall i, j \in [1..m]: \quad & (i \neq j \implies \neg(l2_i.state = Excl \wedge l2_j.state = Excl)) \wedge \\ & \forall k, l \in [1..n]: \ k \neq l \implies \neg(l1_{pk}.state = Excl \wedge l1_{pl}.state = Excl)\end{aligned}$$

In this property, the first subexpression states that no two L2 caches can have exclusive lines concurrently in the system, and the second subexpression states that no two L1 caches can have exclusive lines concurrently, in the same cluster. There is no need to check if two L1 cache lines in different clusters can be exclusive at the same. As our abstraction always retains the L2 cache in each cluster and all the details of cluster p in $\widetilde{M}_p$, this property can be checked in each abstracted protocol. This does not work so directly for a noninclusive coherence protocol, as its $StateCohProp$ requires the state information of all the L1 and L2 caches.

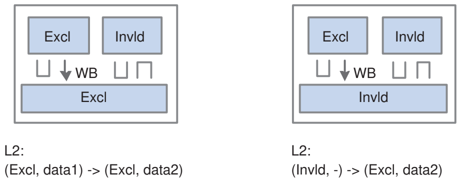

Figure 3: The L2 cache line state of an inclusive (left) and a non-inclusive (right) coherence protocol, on a writeback from an L1 cache.

The second difficulty is that the spurious counterexamples become extremely convoluted and prove to be difficult to eliminate. Figure 3 shows such an example. The left half of this figure shows the situation of an L2 cache receiving a writeback message from an L1 cache in an inclusive protocol. After receiving this message, the L2 line will change from the state of "(Excl, data1)" to "(Excl, data2)". If this cluster is abstracted in an $\widetilde{M}_i$, the state modification of the L2 line will become a transition which, if described in the rule-based language of Murphi [10], reads as "(l2$_i$.state = Excl) $\rightarrow$ l2$_i$.data := new_data;". This transition is a normal protocol behavior, as in any MSI protocol a cache line with an exclusive copy can update its data to a new value. On the other hand, the right half of Figure 3 shows the writeback in a non-inclusive protocol where the L2 cache initially does not have the cache line. After receiving the writeback, the L2 cache line will change from "(Invld, -)" to "(Excl, data2)". When this cluster is abstracted in an $\widetilde{M}_i$, the transition modeling the resulting state update reads $T \equiv$"(l2$_i$.state = Invld) $\rightarrow$ begin l2$_i$.state := Excl; l2$_i$.data := new_data; end;". Obviously, this transition is overly approximated, as the latter transition can easily lead to coherence violation.

In the next section, we will present two approaches to solving these problems. One requires much more manual effort, but allows our technique in [1] to be applied unchanged. The other requires less manual effort, is modular, and can scale to deeper hierarchies of protocol implementations, as are being proposed for many cluster architectures. It involves the use of history variables, but in the context of meta-circular reasoning.

4 Two Approaches to Verify Non-inclusive Protocols

Given a cluster in which the L2 cache does not have a valid line, we can infer if there is any exclusive copy inside the cluster in two ways. One approach is to infer with respect to the state elements situated outside the cluster, i.e. the global directory and the network channels in the level-2 protocol, and the L2 cache line state. The other approach is to infer with respect to the state elements inside the cluster, including the L1 cache states, and the network channels in the level-1 protocol.

4.1 Inferring $exclusive$ from outside the cluster

Take the protocol in Figure 1 as an example. If $\tilde{M_1}$ is the abstracted protocol where the two remote clusters are abstracted from M, the transition "p.l2.state $=$ Invld $\rightarrow$ begin p.l2.state $:=$ Excl; p.l2.data $:=$ new_data; end;" will be a transition on one remote cluster p. We need to ensure that this transition can only happen when there is indeed an exclusive copy inside p. The following expression $IsExcl$ describes how this inference can be done:

$$
\begin{array}{rcl}
IsExcl(p) & \equiv & \text{Dir.State = Excl} \wedge \\
& & \text{GUniMsg[p].Cmd} \neq (\text{ACK} \vee \text{IACK} \vee \text{ImACK}) \wedge \\
& & \text{GUniMsg[h].Cmd} \neq (\text{ACK} \vee \text{IACK} \vee \text{ImACK}) \wedge \\
& & \text{GWbMsg.Cmd = GWB\_None} \wedge \\
& & \text{((GShWbMsg.Cmd = GSHWB\_None} \wedge \\
& & \text{ Dir.HeadPtr = p)} \vee \\
& & \text{ (GShWbMsg.Cmd = DXFER} \wedge \\
& & \text{ GShWbMsg.Cluster = p))}
\end{array}
\qquad [4-1]
$$

$IsExcl$ states that for a given cluster p, if the global directory ($Dir.State$) shows there is an exclusive line in the system, and there is no granted message to p ($GUniMsg[p]$) and to the home cluster h ($GUniMsg[h]$), and there is no writeback ($GWbMsg$) messages, then if there is no shared writeback ($GShWbMsg$) messages or the shared writeback channel is containing a message which indicates that the exclusive copy is to be transferred to p, p must already contain an exclusive copy.

As $IsExcl$ only uses the variables in the level-2 protocol, which are retained in each $\tilde{M_i}$, we can simply use $IsExcl$ to strengthen the guard of the transaction T as described in Section 3. That is, "(l2$_i$.state $=$ Invld $\wedge$ IsExcl(i)) $\rightarrow$ begin l2$_i$.state $:=$ Excl; l2$_i$.data $:=$ new_data; end;" At the same time an additional verification obligation VO needs to be added in $\tilde{M_i}$, with an instance of $i = 1$ shown in the following. This expression is added to ensure that the guard strengthening is sound. That is, we know that the transition T was abstracted from the situation when an L2 cache line in invalid state receives a writeback. If VO is valid, it means that $IsExcl$ always holds under such situations. So we can safely use it to strengthen the guard of T.

$$
\begin{array}{rcl}
VO & \equiv & \\
& & \text{h.WbMsg.Cmd = WB} \rightarrow \text{IsExcl}(h)
\end{array}
$$

As for the example $StateCohProp$ in Section 3, we can use two obligations to indirectly verify it. The first obligation can be described as: $\forall p \in P,\ \forall i \in [1..n]:\ l1_{pi}.state = Excl \Rightarrow IsExcl(p)$. Based on this, we can then verify $\forall p, q \in P:\ (p \neq q) \Rightarrow \neg(IsExcl(p) \wedge IsExcl(q))$ as the second obligation.

This approach of inferring whether a cluster has an exclusive copy, when viewed from outside the cluster, is already quite complex, and hence barely feasible in verifying 2-level protocols. The difficulty of constructing an expression such as $IsExcl$ would be much higher for realistic 2-level protocols. It would become extremely complex with higher numbers of levels. To illustrate this point, consider Figure 4 which shows a 3-level hierarchical protocol. In this example, each L2 cache is non-inclusive of the L1 caches, and the L3 cache is also non-inclusive of the L2 caches. The level-1 protocol involves two L1 caches, the L2 cache, the local directory-1, and a communication controller (RAC1). In this example, the four instances of the level-1 protocol are symmetry. Similarly, the level-2 protocol involves two L2 caches, the L3 cache, the local directory-2, and so on. In level-3, the global directory tracks which specific cluster has a valid copy and its cache line state. Similar with the protocol in Figure 1, when accessing the main memory, the home cluster has a set of different behaviors than that of the remote cluster.

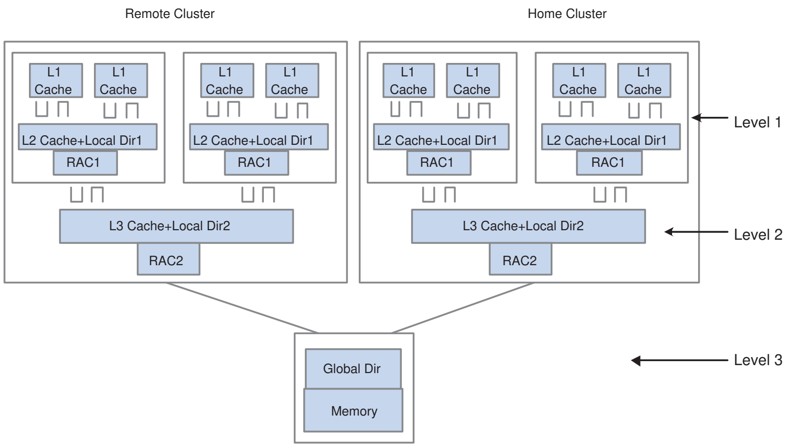

Figure 4: A 3-level non-inclusive protocol H.

Based on the approach in [1], verification of this hierarchical protocol H can be decomposed into verifying two abstracted protocols $\tilde{H}_1$ and $\tilde{H}_2$ shown in Figure 5. The underlying idea of this decomposition is that these two protocols are able to cover all the details of the level-1, level-2 and level-3 protocols, *albeit* with a suitably defined (and far more complex) definition for $IsExcl$.

More specifically, according to the abstraction, we know that in $\tilde{H}_1$ and $\tilde{H}_2$ there is an overly approximated transition U which is similar to T in Section 3, namely

$U \equiv$"(p.l2.state = Invld) $\rightarrow$ begin p.l2.state := Excl; p.l2.data := new_data; end;".

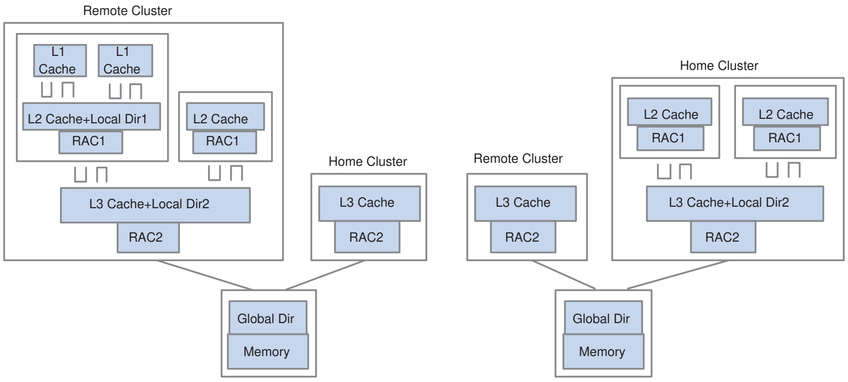

Figure 5: The two abstracted protocols $\tilde{H}_1$ (left) and $\tilde{H}_2$ (right) for the 3-level protocol H.

To strengthen this transition, inferring if there is an exclusive copy inside a cluster from outside this L2 cache may be very complex. The complexity of this expression can be estimated by looking at the expression $IsExcl(p)$ in $[4-1]$. The corresponding $IsExcl$ in the case of H will involve the information from the global directory, the network channels in level-3, the L3 cache, and the local directory and the network channels in level-2. Such a complex expression may be very error-prone and difficult to correctly formulate.

4.2 Inferring $exclusive$ from inside the cluster

We now try to constrain the overly approximated transitions in the abstraction using information inferred from inside a cluster. Consider our 2-level non-inclusive protocol in Figure 1. For the abstracted transition T, we want to use the L1 caches and the level-1 network channels to infer if there is an exclusive copy available in the cluster. This seems a contradiction, as during the over-approximation of T, information pertaining to the L1 caches and the level-1 network channels seems to be lost. How can we restore this information?

Recall, however, that in our approach, for a coherence protocol at each level, there is always an abstracted protocol $\tilde{M}_j$ which includes all the details for that level. So we can use the detailed instance, e.g. $\tilde{M}_k, j \neq k$, to provide and verify the value of the bit..

In more detail, we implemented this approach by adding an additional bit to the L2 cache line in the abstracted protocols. This bit is a function over the L1 caches and messages in the level-1 network channels. Figure 6 shows the data structures of a detailed cluster in M and an abstracted cluster in $\tilde{M}_1$ and $\tilde{M}_2$.

In Figure 6, we can see that an abstracted cluster projects away many details of the original cluster. The extra bit is represented as IE, as *implicit exclusive*. It has a corresponding function, shown as

```
ProcState: record
    -- B1 L1 caches
    L1:   array [NODE] of NODE_STATE;

    -- B2 network channels used inside a processor
    ReqMsg:  array [NODE_L2] of UNI_MSG;
    InvMsg:  array [NODE_L2] of INV_MSG;
    WbMsg:   WB_MSG;
    ShWbMsg: SHWB_MSG;
    NakcMsg: NAKC_MSG;
    SnoopMsg:  array [NODE] of Broadcast_MSG;

    -- local dir
    L2: record
        -- B3.1 used only by level-1 protocol
        pending:     boolean;
        ShrSet:        array [NODE] of boolean ;
        InvCnt:      CacheCnt;
        HeadPtr:     NODE_L2;
        ReqId:       NODE;
        ReqType:     boolean;
        ReqCluster:  Procss;
        ifHoldMsg:   boolean;
        ifBroadcast: boolean;
        ifReplied:   boolean;
        -- B3.2 used by both levels
        State:       L2State;
        Data:        DataSet;
        OnlyCopy:    boolean;
    end;

    -- B4 comm. controller with other clusters and
    -- global dir
    RAC: record
        State:    RACState;
        InvCnt:   ProcsCnt;
    end;
end;
```

```
ABSProcState: record
    L2: record
        -- extra bit
        IE:       boolean;

         -- B3.2 local dir
        State:    L2State;
        Data:     DataSet;
        OnlyCopy: boolean;
    end;

     -- B4 comm. controller
    RAC: record
        State:    RACState;
        InvCnt:   ProcsCnt;
    end;
end;
```

Figure 6: Data structures of a detailed cluster in M, and an abstracted cluster in $\tilde{M}_1$ and $\tilde{M}_1$.

"ImplicitExcl" in the following.

$$
\begin{aligned}
\forall p \in P,\ ImplicitExcl(p) \ \equiv \quad & \exists i \in [1..m] : (\text{p.l1}_i\text{.State=Excl} \vee \\
& \qquad\qquad \text{p.SnoopMsg[i].Cmd = (Put} \vee \text{PutX)} \vee \\
& \qquad\qquad \text{p.ReqMsg[i].Cmd = PutX)} \vee \\
& \text{p.WbMsg.Cmd = WB} \vee \\
& \text{p.ShWbMsg.Cmd = ShWb} \vee \\
& \text{p.ShWbMsg.Cmd = FAck}
\end{aligned}
$$

Here, $ImplicitExcl$ says if a cluster has an L1 cache line with an exclusive copy, or the broadcast channel contains a grant reply, or a request channel contains a grant exclusive reply, or a valid writeback or an exclusive ownership transfer message, the cluster must have an exclusive copy available somewhere other than in the L2 cache. So this bit can be regarded as restoring some information from the details projected away in the abstraction. More importantly, it changes an abstracted protocol into a pseudo-inclusive protocols where, for example, the L2 cache can always know if it can supply an exclusive copy from inside the cluster. Note that such transformation from "non-inclusion" to "inclusion" only happens in building the abstracted protocols. We do not need to modify the original hierarchical protocol.

With these extra bits in the system, the challenges discussed in Section 3 can be solved fairly easily. Firstly, for the coherence property $StateCohProp$, it could now be stated and checked in each $\tilde{M}_i$. The following shows this property in $\tilde{M}_1$, where the home cluster h keeps all the details while the remote clusters are abstracted.

$$
\begin{array}{l}
StateCohProp' \ = \\
\quad \forall r1, r2 \in Remoteclusters : \\
\qquad \neg(\exists k, l \in [1..n]; h.l1_k.state = Excl \wedge h.l1_l.state = Excl) \wedge \\
\qquad \neg((h.l2.IE \vee h.l2.State = Excl) \wedge (r1.l2.IE \vee r1.l2.State = Excl)) \wedge \\
\qquad \neg((r1 \neq r2) \ \wedge \ ((r1.l2.IE \vee r1.l2.State = Excl) \wedge (r2.l2.IE \vee r2.l2.State = Excl)))
\end{array}
$$

Secondly, for the overly approximated transitions like T in Section 3, T can be simply strengthened as "(p.l2.state = Invld$\wedge$ p.l2.IE) $\rightarrow$ begin p.l2.state := Excl; p.l2.data := new_data; end;". The additional verification obligation can now be stated as: $VO \equiv$ "(p.WbMsg.Cmd = WB $\wedge$ p.l2.State = Invld) $\rightarrow$ p.l2.IE". For all the other spurious counterexamples, they can be eliminated similarly using these extra bits. Note that the values of these bits only need *local* reasoning of one level in a hierarchical protocol, as contrast with the approach of inferring "exclusive" from all the outside information. Therefore, it is a more general method to verifying hierarchical protocols.

Now it comes to the question of how to update each extra bit in the abstracted protocols such that its value always equal to the definition. This can be implemented fairly easily in three steps. (1) As these bits do not exist in M, we construct another protocol M' which is the same with M except for the updates of the the extra bits. These bits initially are all $false$. When an exclusive copy is granted to an L1 cache in a message, the bit for that cluster is set to $true$ ("ReqMsg[]"). When an exclusive or shared writeback from an L1 cache is received by the L2 cache, the bit is set to $false$ ("SnoopMsg[], WBMsg, ShWbMsg"). (2) The procedure in the Appendix of [1] is applied to generate $\tilde{M}_1$ and $\tilde{M}_2$. (3) An additional verification obligation "ImplicitExcl" is added to both the $\tilde{M}_i$, stating that "$l2_i.IE = ImplicitExcl(i)$". This property ensures that the value of each bit is always consistent with the definition.

5 Verifying the 2-level non-inclusive protocol

For the 2-level non-inclusive protocol M described in Section 2, we use Murphi as an explicit model checker to verify it. After enumerating $1,521,900,000$ of states[3], model checking failed due to the state explosion problem. This is not surprising, considering the multiplicative effect of having three instances of complex coherence protocols running concurrently.

[3] We did all these experiments on an IA-64 machine, with a 1.4GHz Itanium-2 processor and 24GB of memory, and 40-bit hash compaction was used for the state representation.

Using the abstraction and the two refinement approaches described in Section 4, we were able to verify this complex protocol. In detail, three abstracted protocols $\tilde{M}_1$, $\tilde{M}_2$ and $\tilde{M}_3$ are built from M ($\tilde{M}_2$ and $\tilde{M}_3$ are the same due to the symmetry between two remote clusters). 17 iterations of refinement were applied in each $\tilde{M}_i$, and after that we were able to claim that the coherence property holds in M. Model checking of $\tilde{M}_1$ resulted in $234,478,105$ states, and $\tilde{M}_2$ in $283,124,383$ states. From these numbers, we can see that our abstraction and refinement approach is in fact very effective. It's also worthwhile to mention that, the second refinement approach of adding extra bits in fact do not introduce more states of the protocol. This is reasonable, as the value of each bit can be thought as a simple logical expression of other variables in the system.

In the following, we will formally prove that by introducing the extra bits in $\tilde{M}_1$ and $\tilde{M}_2$, our approach is still sound. That is, once $\tilde{M}_1$ and $\tilde{M}_2$ can be verified with respect to the coherence property ϕ_{coh} in M, M itself must also be coherent on ϕ_{coh}. We begin with a theorem and then apply it in the context of a particular protocol abstraction.

5.1 A theorem justifying metacircular reasoning

Theorem 1 A state transition system (STS) $\mathbb{M} = (\mathbb{S}, \mathbb{I}, \mathbb{T}, \mathbb{Z})$, where $\mathbb{S}$ is the set of states, $\mathbb{I} \subseteq \mathbb{S}$ is the set of initial states, $\mathbb{T} \subseteq \mathbb{S} \times \mathbb{S}$ is the set of transition relations, and $\mathbb{Z}$ is a set of properties to be checked in $\mathbb{M}$, i.e. $\forall z \in \mathbb{Z},\ z :\ \mathbb{S} \rightarrow Boolean$.

Suppose $M, M_1, \ldots, M_k$ are $(k+1)$ STSs with $M = (S, I, T, Z),\ M_i = (S_i, I_i, T_i, Z_i)$, and $\alpha_1, \ldots, \alpha_k$ are k functions over S, $\alpha_i : S \rightarrow S_i : i \in [1..k],\ k \in \mathbb{N}$, such that

$$\begin{aligned}\forall s \in S,\quad & \forall z \in Z,\ \exists z_1 \in Z_1, \ldots, z_k \in Z_k : \\ & z_1(\alpha_1(s)) \wedge \ldots \wedge z_k(\alpha_k(s)) \ \rightarrow\ z(s)\end{aligned}$$

Then if Z_i's are valid with respect to S_i's, i.e. $\forall z_i \in Z_i,\ \forall s_i \in S_i :\ z_i(s_i) = true,\ i \in [1..k]$, Z must be valid with respect to S. $\square$

Theorem 1 can be proved using a simple contradiction.

5.2 Applying the theorem

Now we prove that for our non-inclusive coherence protocol M, the abstracted protocols $\tilde{M}_1, \tilde{M}_2$ and $\tilde{M}_3$ ($\tilde{M}_3$ is the same with $\tilde{M}_2$) indeed satisfy the conditions of *Theorem 1*.

First of all, it is straightforward that $M, \tilde{M}_1$ and $\tilde{M}_2$ can be regarded as STSs, and cache coherence properties ϕ_{coh} are the $\mathbb{Z}$ of M. For each state $s \in S$ of M, s can be represented as a state vector containing components $v1, v2, \ldots, v7$, written as $s = \langle v1, v2, v3, v4, v5, v6, v7\rangle$. Here, $v1$ represents the state components corresponding to **B1**, **B2** and **B3.1** in the home cluster of M (see Figure 6). And $v2$ represents the state components of **B3.2** and **B4** in the home cluster. Similarly $v3, v4$ are used to represent remote cluster-1 and $v5, v6$ are for remote cluster-2. $v7$ is used to represent the rest of the components in the system. They include the global directory, the main memory, and the set of network channels used in level-2.

Secondly, the functions α_i for $\tilde{M}_i : i \in [1..3]$ can be represented simply as follows:

$\forall s \in S$ of M, $s = \langle v1, v2, v3, v4, v5, v6, v7\rangle$

(1) $\alpha_1(s) = \langle v1, v2, v4, v6, v7, f(s)\rangle$

(2) $\alpha_2(s) = \langle v2, v3, v4, v6, v7, f(s)\rangle$

(3) $\alpha_3(s) = \langle v2, v4, v5, v6, v7, f(s)\rangle$

In α_i, f is a function over S, $f : S \rightarrow array[1..3]\ of\ boolean$. In detail, for a given state s of M, f returns one bit of "IE" (implicit exclusive) for each of the cluster. That is, $f(s) = \langle ImplicitExcl(h), ImplicitExcl(r1), ImplicitExcl(r2)\rangle$, where h represents the home cluster, and $r1$ and $r2$ are for the remote clusters. We can see that $\forall s \in S,\ i \in [1..3] :\ \alpha_i(s) \in S_i$.

Thirdly, for each cache coherence property $z \in Z$ of M, according to the abstraction (the procedure is detailed in [1]), z has a corresponding property z_i in $\tilde{M}_i$. Each z_i is an over-approximation of z, i.e. $z \Rightarrow z_i$. It is obtained by replacing the least sub-expressions of z that contains a variable which is abstracted away in $\tilde{M}_i$ with $true$. Because $\forall s \in S$ of M, the union of $\alpha_1(s), \alpha_2(s)$ and $\alpha_3(s)$ are already able to cover every component of s, z can in most cases be directly represented using z_1, z_2 and z_3. For example, if $z \equiv \forall p \in P, \forall i \in [1..n],\ p.l1_i.State \neq ErrorState$, where $P = \{h, r1, r2\}$, then

z_1: $\forall i \in [1..n],\ h.l1_i.State \neq ErrorState$

z_2: $\forall i \in [1..n],\ r1.l1_i.State \neq ErrorState$

z_3: $\forall i \in [1..n],\ r2.l1_i.State \neq ErrorState$

It is straightforward that $z_1 \wedge z_2 \wedge z_3 \Rightarrow z$.

For our non-inclusive hierarchical protocol, with the extra bits IE and the functions $ImplicitExcl$ introduced, we were able to represent each $z \in Z$ of M fairly easily, using the corresponding z_1, z_2 and z_3. The example of $StateCohProp$ illustrated in Section 3 showed how it can be represented in each $\tilde{M}_i$.

In summary, our compositional approach ensures that for the 2-level non-inclusive protocol, once $\tilde{M_1}$, $\tilde{M_2}$ and $\tilde{M_3}$ are verified with respect to the coherence properties ϕ_{coh}, M most also be coherent with ϕ_{coh}.

6 Conclusion

Hierarchical cache coherence protocols are notoriously difficult to verify, as they usually have more corner cases than non-hierarchical protocols. And the multiplicative effect of having more than one instance of coherence protocols running concurrently can make the state space astronomial. Inclusion, exclusion and non-inclusion are the three caching policies two neighboring level of caches often use. In our previous work [1], we proposed a compositional approach to verify a 2-level inclusive hierarchical cache coherence protocol used in MCMP systems. However, hierarchical coherence protocols using exclusion and non-inclusion pose extra difficulty in the verification. In this paper, we propose a general approach which can be used to verify hierarchical protocols with any caching policy. By introducing extra bits in a non-inclusive hierarchical protocol, the verification becomes almost as easy as that of inclusive ones. And the soundness of this approach is ensured by adding an additional verification obligation, to constrain the behavior of these bits to be consistent with their definitions. We think this approach is lightweight, and it can be generally applied to the verification of hierarchical coherence protocols with more than two levels. Our success of verifying a complex 2-level non-inclusive coherence protocol shows that other hierarchical protocols could be verified in a similar way.

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] K.L.McMillan and N.Amla. Automatic abstraction without counterexamples. In *Technical Report of Cadence*, 2003.

[3] E.M.Clarke. Proving the correctness of coroutines without history variables. In *ACM Southeast Regional Conference*, 1978.

[4] M.Clint. Program proving: Coroutines. In *Acta Informatica*, 1973.

[5] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The stanford flash multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 302–313, 1994.

[6] 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.

[7] K.L. McMillan. Verification of infinite state systems by compositional model checking. In *Correct Hardware Design and Verification Methods*, pages 219–234, 1999.

[8] http://www.cs.utah.edu/formal_verification/noninclusive.tar.gz.

[9] D.E.Culler, J.P.Singh, and A.Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1998.

[10] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–5, 1992.