The Complexity of Verifying Memory Coherence

Jason F. Cantin Mikko H. Lipasti James E. Smith
University of Wisconsin-Madison
1415 Engineering Drive
Madison, WI. 53706
{ jcantin, lipasti, jes} @ece.wisc.edu

ABSTRACT
The general problem of verifying coherence for shared-memory multiprocessor executions is NP-Complete. Verifying memory consistency models is therefore NP-Hard, because memory consistency models require coherence for some or all operations. However, verifying memory consistency remains NP-Complete for executions known to be coherent.

Categories and Subject Descriptors
B.8.1 [**Hardware**]: Performance and Reliability—*Reliability, Testing, and Fault-Tolerance*; F2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and Scheduling*.

General Terms
Verification, Reliability, Theory, Algorithms.

Keywords
Shared Memory, Coherence, Consistency Models.

1. INTRODUCTION
Memory coherence is an important feature of shared-memory systems. We analyze the problem of determining whether a shared-memory system provided coherence for an execution, and prove it is NP-Complete.

Verifying memory coherence remains NP-Complete under a number of restrictions. It remains NP-Complete when each process is restricted to a small number of memory operations, and when values are written only a small number of times. The problem becomes tractable if the number of processes is restricted, or additional information such as the order of writes or the mapping of reads to writes is provided.

The NP-Completeness of verifying coherence implies the NP-Hardness of verifying adherence to memory consistency models that require coherence, including sequential consistency. Further, our results extend to consistency models that do not require coherence for all operations, but allow the programmer to serialize memory operations with special instructions.

However, the complexity of verifying consistency is not a mere consequence of the complexity of verifying coherence. We find that verifying sequential consistency is NP-Complete for executions known to be coherent.

SPAA'03, June 7-9, 2003, San Diego, California, USA.
ACM 1-58113-661-7/03/0006.

2. RELATED WORK
Gibbons and Korach defined the VSC problem for verifying sequential consistency, and characterized its complexity [5,6,7]. Since sequential consistency and memory coherence are equivalent for executions that use only one shared variable, some of their results apply to memory coherence as well. However, the general problem of verifying sequential consistency for one location (memory coherence) was left as an open problem in [7].

3. PRELIMINARIES
Similar to prior work, reads are of the form "R(a, d)" and writes are denoted "W(a, d)", where *a* is the address and *d* is the value read/written. A *process history* is a sequence of these operations, as generated by the execution of a process. We assume that each location has *initial value* $d_I[a]$ prior to execution.

A *coherent schedule* is an interleaving of single-address process histories in which every read returns the value of the last write (except reads that precede the first write, which must return the initial value $d_I[a]$). A multiprocess execution is considered *coherent* if a coherent schedule exists for each location *a*.

To reason about the complexity of verifying coherence, we define a new decision problem:

DEFINITION 3.1: Verifying Memory Coherence (VMC)

INSTANCE:	Value set *D*, address *a*, and set *H* of process histories, each consisting of a finite sequence of memory operations.
QUESTION:	Is there a coherent schedule *S* for the operations of *H* with address *a*?

4. VERIFYING MEMORY COHERENCE
VMC is NP-Complete. Its NP-Hardness follows from a reduction from the known NP-Complete problem SAT [3]. The key idea is that the truth assignment of a variable *u* can be encoded by the order in which two writes appear in a schedule *S*.

$$T : U \mapsto \{True, False\}$$
$$W(a,d_u) \xrightarrow{S} W(a,d_{\bar{u}}) \quad \Leftrightarrow \quad T(u) = True \qquad (1)$$
$$W(a,d_{\bar{u}}) \xrightarrow{S} W(a,d_u) \quad \Leftrightarrow \quad T(\bar{u}) = True$$

Given an instance *Q* of SAT with set *U* of *m* variables and set *C* of *n* clauses, we define a set *H* of *2m+3* process histories for which a coherent schedule *S* exists if and only if *Q* is satisfiable. We first define two process histories, h_1 and h_2, each with a write

from the pair $\{W(a, d_u), W(a, d_{\bar{u}})\}$ for each variable u in U. We then define two process histories (h_u and $h_{\bar{u}}$) for each variable to test the literals, each reading the unique values in the order that corresponds to *true* for the literal (1). Once h_1 or h_2 have been interleaved in some way in S, only process histories representing literals that are *true* under the corresponding truth assignment may be interleaved with them in a coherent schedule S.

For each clause c (in C) satisfied by a literal l, a write with a special value d_c is appended to the process history h_l. Another process history, h_3, is defined with reads that return each of these values. This process history may be included in S only if each of these values was written previously (i.e., each clause satisfied).

After the reads in h_3, we rewrite the values d_u and $d_{\bar{u}}$ for each u such that these values appear in both orders in S. All remaining process histories may then be interleaved with these writes, completing the schedule. The complete proof may be found in [1].

5. RESTRICTED CASES

We find that VMC remains NP-Complete for as few as three simple operations (reads & writes) per process and values written only twice. VMC is also NP-Complete with only two read-modify-writes per process [6], and values written only three times.

If the memory system is augmented to provide the order in which writes were performed (i.e., the *write-order* [5,6,7]), VMC has an $O(n^2)$ time algorithm for n total operations. It follows from prior work that VMC is tractable if every process is allowed only one operation, or values are written only once (i.e., the *read-map*) [6]. The problem is also tractable when the number of processes is restricted. With n total memory operations, k processes and c addresses, VSC can be solved in $O(n^k k^c)$ time [7]. Since all instances of VMC are instances of VSC in which $c=1$, the complexity of VMC is $O(kn^k)$, which is polynomial for constant k. Similarly, all instances of VMC with only read-modify-writes and a constant number of processes have $O(n^k)$ time complexity [7].

For only two simple memory operations per process, the complexity is unknown. The case for read-modify-writes in which data values are written at most twice is also an open problem. Table 1 summarizes the results, proofs may be found in [1].

Table 1. Complexity results for coherence. New results shaded.

	Simple Reads/Writes	Read-Modify-Writes
1 Operation/Process	O(nlg(n))	$O(n^2)$
2 Operations/Process	**?**	NP-Complete
3+ Operations/Process	NP-Complete	NP-Complete
Constant Processes	$O(n^k)$	$O(n^k)$
1 Write/Value	O(n)	O(nlg(n))
2 Writes/Value	NP-Complete	**?**
3+ Writes/Value	NP-Complete	NP-Complete
Write-order Given	$O(n^2)$	O(n)

6. VERIFYING MEMORY CONSISTENCY

All hardware-implemented consistency models in the literature reduce to memory coherence for executions that access only one shared location [4]. Therefore, they are NP-Hard to verify.

Though there are models that relax the coherence requirement (e.g., Lazy Release Consistency [2]), these models provide special instructions with which the programmer can explicitly serialize memory operations. We can therefore modify our reductions for these models by adding these special instructions. For LRC, we place *acquire* and *release* operations around each operation in the original reduction. As long as there is a way to serialize memory operations, a SAT reduction should be possible.

However, the NP-Hardness of verifying consistency models is not a mere consequence of the NP-Completeness of verifying coherence. We find that verifying sequential consistency remains NP-Complete when it is known that coherence was provided. This follows from a reduction similar to the one used previously for VMC, modified such that H is *always* coherent for each individual location a.

Given an instance Q of SAT, we define H as follows: There are two process histories, h_1 and h_2, each with one of a pair of writes to a unique address a_u for each variable u in U. The truth of u corresponds to the order in which these writes appear in a schedule S (2). Two process histories (h_u and $h_{\bar{u}}$) are defined for the literals of each variable u, each reading the values d_X and d_Y in the order that corresponds to *true* for the literal. Once the writes have been ordered in some way, only one of these two histories may be interleaved with them in a coherent schedule S.

$$
\begin{aligned}
&T : U \mapsto \{True, False\} \\
&W(a_u, d_X) \xrightarrow{S} W(a_u, d_Y) \quad \Leftrightarrow \quad T(u) = True \\
&W(a_u, d_Y) \xrightarrow{S} W(a_u, d_X) \quad \Leftrightarrow \quad T(\bar{u}) = True
\end{aligned}
\tag{2}
$$

For each clause c (in C) satisfied by a literal l, a write to a special location a_c is appended to the process history h_l. Another process history, h_3, is defined with reads that return the value written to each location a_c. This process history may be included in a schedule S if and only if each of location has been written.

After a special location, a_Δ, is written at the end of h_3, each location a_u is rewritten by h_1 and h_2 such that the values d_X and d_Y appear in both orders in S. All remaining process histories may then be interleaved with this last set of writes, completing the schedule. The complete proof may be found in [1].

7. CONCLUDING REMARKS

The results in this paper and prior work [5,6,7], suggest that verifying coherence and consistency are inherently difficult problems. Further, though verifying coherence may itself be of practical utility, it may not simply the verification of consistency.

8. ACKNOWLEDGEMENTS

This research was supported by NSF grant CCR-0083126, IBM, and fellowships from the NSF and the UW Foundation.

9. REFERENCES

[1] Cantin, J. The Complexity of Verifying Memory Coherence. Technical Report ECE-03-1, UW-Madison, 2003.

[2] Keleher, P., Cox, A., and Zwaenepoel, W. Lazy Release Consistency for Software Distributed Shared Memory. Proc. of the 19th Int'l Symp. on Computer Architecture, 1992.

[3] Garey, M., and Johnson, D. Computers and Intractability: A Guide to the Theory of NP-Completeness. NY, W.H. Freeman and Co., 1979: 38-39, 95-107, 259.

[4] Gharachorloo, K. Memory Consistency Models for Shared-Memory Multiprocessors. WRL Research Report. 1995.

[5] Gibbons, P., and Korach, E. The Complexity of Sequential Consistency. Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing, 1992:317-325.

[6] Gibbons, P., and Korach, E. Testing Shared Memories. SIAM Journal of Computing, Aug. 1997:1208-1244.

[7] Gibbons, P., and Korach, E. New Results on the Complexity of Sequential Consistency, Tech. Report, AT&T Bell Labs, Murray Hill, NJ. Sep. 1993.