Visualizing Execution Histories on Multiple Memory Consistency Models

Alba Cristina Magalhães de Melo, Simone Cintra Chagas
albamm@cic.unb.br, simonec@cic.unb.br
Departamento de Ciência da Computação/UnB
Campus Universitário - Asa Norte - CEP -70910-900 Brasília - Brazil
Fone: (55)(061)3482702

Abstract *In order to provide a better understanding on the semantics of the memory models, many researchers have proposed formalisms to define them. Unfortunately, many of the formal definitions are so complex that it is still difficult to say what kind of execution history can be produced on a particular memory consistency model. In this paper, we propose a visualization tool that shows what operations orderings could lead to user-defined execution histories on different memory consistency models. The memory consistency model defines order relations that restrict the executions that can be produced. We also present a prototype of this visualization tool that analyses two-processor execution histories for two memory consistency models: sequential consistency and PRAM consistency.*

1 Introduction

Using the shared memory programming paradigm in parallel architectures is quite complex. The main reason for this increased complexity is the difficulty to make the shared memory behave exactly as if it was a uniprocessor memory. This characteristic is important since it could make uniprocessor programs automatically portable to parallel architectures. Besides, programmers are already used to the uniprocessor shared memory programming model and using a similar model would make parallel programming easier.

In [Lam79], a memory model called Sequential Consistency was proposed. Using this memory model, all parallel processes are able to observe all shared memory accesses in the same order. However, this model slightly differs from the uniprocessor model because real time order is relaxed. Many hardware and software systems have implemented Sequential Consistency. Although Sequential Consistency provides a relatively easy programming model, there is a great overhead on coherence operations to guarantee that all processors see always the same access order [Mos93].

To reduce coherence overhead, researchers have proposed to relax some consistency conditions, thus creating new shared memory behaviors that are different from the traditional uniprocessor one. These memory models are called relaxed memory consistency models because they only guarantee that some of the shared memory accesses are seen by all processors on the same order.

Memory consistency models, strong or relaxed ones, have not been originally formally defined. The lack of a unique framework where memory models can be formally defined makes it difficult to compare and understand the memory model semantics.

In this article, we describe a visualization tool that assists DSM-system designers and multiprocessor hardware designers in the task of specifying and analyzing a memory consistency model. Particularly, this new tool analyses a particular execution history on a chosen memory model and shows if it is valid or not. Basically, we build an execution tree and traverse it, respecting the constraints imposed by the current memory model. If we arrive at the leaves, there is at least one valid execution path and, thus, the execution history is valid on the memory model. We describe also the prototype of this visualization tool that analyses execution histories on two memory models: Sequential Consistency and PRAM Consistency.

The rest of this paper is organized as follows. Section 2 describes our system model. Section 3 presents a denotational approach based on execution trees that is used to decide the validity of an execution history on a particular memory model. The prototype of a multiple memory consistency model visualization tool is presented in section 4. Related work in the area of DSM visualization is presented in section 5. Finally, conclusions are presented in section 6.

2 System Model

To describe memory models in a formal way, we propose a history-based system model that is related to the models described in [Adv93] and [KNA93]. This model was already described in [Bal97]. In the present paper, we will only review some definitions:

System	A finite set of processors
Processor p_i	Executes operations on the Shared Global Memory M
Shared Global Memory M	Contains all memory addresses
Local memory m_i	Caches all memory addresses of M
$Op_i(x)v$	Memory operation executed by processor p_i on address x with value v.
Basic types of operation on M	read(r) and write (w)
Syncronization operations on M	sync(x);acquire(x);release(x) or user-defined operations
$O_{pi}(x)v$ is issued	Processor p_i executes the instruction $o(x)v$.
$r_{pi}(x)v$ is performed	A write operation on x cannot modify the value returned to p_i
$w_{pi}(x)v$	$c = \sum_{a=0}^{n-1} w_{pa}(x)v$
$w_{pi}(x)v$ perfomed with respect to processor p_j	The value v is written to the address x on the local memory m_j of p_j
$w_{pi}(x)v$ performed	$w_{pi}(x)v$ is performed with respect to all processors.
Local execution history H_{pi}	An ordered sequence of memory operations issued by p_i
Execution history H	$U\ H_{pi}$
Memory Consistency Model	Defines an order relation $(\xrightarrow{R})$ on a set of shared memory accesses

In our definitions, we use the notion of *linear sequences*. If H is a history, a *linear sequence* of H contains all operations in H exactly once. A linear sequence is *legal* if all read operations *r(x)v* return the value written by the most recent write operation on the same address in the sequence. The decision of which orderings are valid in an execution history is made by the *memory consistency model*. One execution history is valid on a memory consistency model if it can be produced with the order relation defined by the model.

3 Validating Execution Histories

Computing orderings for execution histories is a problem that has been extensively studied in [NeMi90]. In this paper, this problem is shown to be NP-hard. This is basically because, in the most general case, we must examine all possible orderings before deciding which orderings could have produced a particular execution history.

Validating a given execution history on a particular memory model is thus a complex task that is decomposed in three main steps in our system: build the execution tree, extract constraints from the memory model and traverse the execution tree.

In order to build a complete execution tree, we must include all possible interleavings of shared memory operations that appear in an execution history. Figure 1 shows an execution history and its associated execution tree.

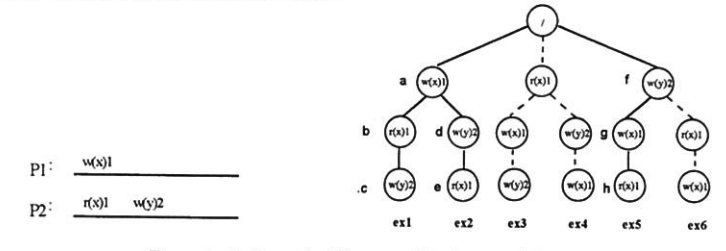

Figure 1 - An Execution History and Its Associated Execution Tree

In figure 1, there are six possible execution paths. Each path is an ordered sequence of nodes, from the root to the leaf. There are $p!$ linear execution orders that can be derived, where p is the number of memory operations to be considered. We observe thus an explosion of the number of possible paths. Although all the sequences generated in figure 1 are linear, some of them are not well-formed as defined by [HS92]. Removing non well-formed sequences from the execution tree will surely reduce the size of the tree and that is done with no side-effect, as there is no practical interest in allowing values to be read before they are written.

For every chosen memory consistency model, there are two decisions that must be taken. These decisions will be based on the formal definitions. As an illustration, we will consider the formal definitions of SC and PRAM presented in [Bal97]. First, we must identify the set of operations that will be validated. In the case of strong memory models, such as Sequential Consistency, all shared memory operations must be included. In the case of relaxed memory models, only a subset of the memory operations will be verified. For example, in PRAM consistency, H_{pl+w} is this subset. Second, we must define what order must be respected in this set of operations. In the particular case of Sequential Consistency and PRAM Consistency, the only order to be respected is program order. For the particular history in figure 1, the program order that must be respected is $r_{p2}(x)1 \overset{po}{\rightarrow} w_{p2}(y)2$. The last step in validating a history consists to examine all possible paths that belong to the execution tree and verify if they satisfy the constraints imposed by the model. We use a left-to-right in-depth algorithm to traverse the execution tree in search of valid execution paths.

When a node is visited, there are two decisions that can be taken. If the operation in the node is valid on the memory consistency model, the in-depth search continues and the next node is examined. If the operation is not valid, the current path is abandoned and the next left-to-right path is examined. If we arrive at one valid leaf, that means that the whole path is valid.

4 Prototype Implementation

We implemented a prototype of our visualization tool using the Delphi 3.0 programming environment. The functionality of the prototype is shown in figure 2.

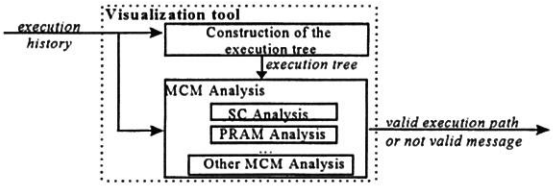

Figure 2 - Prototype of the visualization tool

In our visualization prototype, there are two main modules: *Construction of the Execution Tree* and *Memory Consistency Model Analisys*. The first module receives an execution history as input. This execution history can be selected by the user among predefined execution histories or can be provided by the user himself. In the first version of our prototype, we limited execution histories to have only two processors (P_1 and P_2) with at most three memory operations per processor. The basic function of this module is to construct a tree containing the possible execution paths that could have generated the chosen execution history.

After the tree construction, the user must choose which memory consistency model is to be used in the analysis. By now, we have only implemented Sequential Consistency [Lamp79] and PRAM Consistency[LS88]. The chosen MCM receives the execution history and its associated execution tree and searches the tree for valid paths, as explained in section 3.

Our visualization tool provides both a memory view and an ordering view. The memory view shows the values of each memory position x for every memory M_i. The ordering view shows the execution path that is being examined as well as the valid and invalid execution paths that have already been decided. The analysis of an execution history can be done automatically or in a step-by-step basis.

5 Related Work

[CBE95] presents a visualization tool called StormWatch. This tool is able to analyse the same application on different memory coherence protocols. StormWatch provides trace, source and communication views. The trace view is based on execution histories. Maya is a simulation plataform described in [ACL94]. It analyses the behavior of coherence protocols that implement multiple memory consistency models. Basically, these two systems provide visualization of the behavior of parallel applications on some specific implementations of memory consistency models. While this kind of visualization is very useful, it only shows that some of the constraints imposed by a particular implementation of a memory consistency model can lead to bad performance results.

Analysing feasible event orderings has been previously done in the domain of parallel and distributed debugging, specially to detect race conditions [AHM91]. [NeMi90] analysed formally the possible event orderings that can be generated by a parallel program and demonstrated the NP-hardness of this problem.

As far as we know, this is the first work that provides a visualization tool that can be used to analyse the effects and the potentialities of memory consistency models, in a way that is totally independent from coherence protocols that could implement it. We claim that this separation is necessary.

6 Conclusions and Future Work

In this article, we presented a visualization tool that analyses execution histories and shows graphically what valid execution paths could have led to their production. The analisys of execution histories can be done on a variety of memory consistency models. We claim that visualizing the possible interleavings of shared memory accesses is helpful for multiprocessor hardware designers and DSM-system designers in the task of choosing the most appropriate memory model. This can reduce considerably the number of unexpected results that are produced after the machine or the software is released.

As future work, we intend to incorporate more memory models to our visualization tool. Also, we intend to define a language for specifying memory consistency models based on formal definitions.

7 References

[ACL94] D. Agrawal, M. Choy, H. Leong, A. Singh, *Evaluating Weak Memories with Maya*, In Proceedings of the 8th Workshop on Parallel and Distributed Simulation, July, 1994, pages 151-155.

[Adv93] S. V. Adve, *Designing Multiple Memory Consistency Models for Shared-Memory Multiprocessors*, PhD thesis, University of Wisconsin-Madison, 1993.

[AHM91] S. V. Adve, M. Hill, B. Miller, H. Netzer, *Detecting Data Races on Weak Memory Systems*, In 18th Annual International Symposium on Computer Architecture, May, 1991, p.234-243.

[Bal97] A. Balaniuk, Supporting Multiple Memory Consistency Models on a Shared Virtual Memory Parallel Programming Environment, In SBAC-PAD 97, October, 1997, pages 365-379.

[CBE95] T. Chilimbi, T. Ball, S. Eick, J. Larus, *StormWatch: a Tool for Visualizing Memory System Protocols*, Supercomputing, 1995.

[HS92] A Heddaya, H. Sinha, *An Overview of Mermera: a System Formalism for Non-Coherent Distributed Parallel Memory*, Technical report BU-CS-92-009, Boston University, September, 1992, 21 pages.

[KNA93] P. Kohli, G. Neiger, M. Ahamad, *A Characterization of Scalable Shared Memories*. Technical Report GIT-CC-93/04, GIT, 1993.

[Lam79] L. Lamport, *How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs*, IEEE Transactions on Computers, pages 690-691, 1979.

[LS88] Lipton, R. J. and Sandberg, J. S. PRAM: A Scalable Shared Memory. Technical Report CS-TR-180-88, Princeton University, September 1988.

[Mos93] D. Mosberger, *Memory Consistency Models*, Operating Systems Review, pages 18-26, 1993.

[NeMi90] R. Netzer, B. Miller, *On the Complexity of Event Ordering for Shared-Memory Parallel Program Executions*, Technical Report TR-908, University of Wisconsin-Madison, January, 1990.