Relaxed-Memory Concurrency

Joint teams MM Moscova Project-Team, INRIA & Computer Laboratory, U. of Cambridge

Quiz 1.

In the system below, the two threads run in parallel on an x86 multiprocessor and share the memory locations x and y, which initially hold 0:

SB

Thread 0	Thread 1
MOV [x]←1	MOV [y]←1
MOV EAX←[y]	MOV EBX←[x]

Can you guess all the possible final states? How many final states are there?

Quiz 2.

In the system below, the two threads run in parallel on an x86 multiprocessor and share the memory locations x and y, which initially hold 0:

n6

Thread 0	Thread 1
MOV [x]←1	MOV [y]←2
MOV EAX←[x]	MOV [x]←2
MOV EBX←[y]	

Can the system reach a final state where register 0:EAX holds 1, register 0:EBX holds 0, and the the memory location x holds 1?

Quiz 3.

In the system below, the two threads run in parallel on a Power multiprocessor and share the memory locations x and y, which initially hold 0:

LB

Thread 0	Thread 1
lwz r1←[x]	lwz r2←[y]
stw [y]←1	stw [x]←1

Can the system reach a final state where both registers 0:r1 and 1:r2 hold 1?

Is this behaviour possible on an x86 multiprocessor?

Quiz 4.

In the system below, the four threads run in parallel on a Power multiprocessor and share the memory locations x and y, which initially hold 0:

IRIW

Thread 0	Thread 1	Thread 2	Thread 3
stw [x]←1	stw [y]←1	lwz r1←[x]	lwz r3←[y]
		lwz r2←[y]	lwz r4←[x]

Can the system reach a final state where register 2:r1 holds 1 and 2:r2 holds 0 (that is, thread 2 sees the write to x but not the write to y) while register 3:r3 holds 1 and 3:r4 holds 0 (that is, thread 3 sees the write to y but not the write to x)?

Is this behaviour possible on an x86 multiprocessor?

Quiz 5.

The code below attempts to implement a common message passing strategy: Thread 0 updates a data structure (at memory location x, initially 0) and then sets a flag (at memory location y, initially 0), while Thread 1 checks the flag and then accesses the data structure:

MP

Thread 0	Thread 1
stw [x]←1	lwz r0←[y]
stw [y] ←1	lwz r1←[x]

Suppose that at the end of the execution Thread 1 has seen the update to the flag, that is, 1:r0 = 1. Are we guaranteed that 1:r1 holds 1?

If not, can you propose a way to insert memory barriers in the code so that the message passing idiom works implemented correctly? The available barrier instructions are sync, lwsync, and isync.

Quiz 6.

Consider the following Java program where x and y are global variables initially holding 0:

Java

Thread 0	Thread 1
r1 = x	r2 = y
y = r1	x = (r2==1)?y:1
	print r2

Can this program print 1? What if you compile it with HotSpot or Gcj?

Answers

1. There are 4 possible final states. In the unexpected final state, the memory locations hold 1 while the registers all hold 0 (that is, 0:EAX = 0 and 1:EBX = 0). This test highlights that x86 multiprocessors implement store buffering. See [3,5].

2. The answer is yes. However an accurate reading of the Intel 64 Architecture Memory Ordering White Paper suggests that such final state is forbidden. This example shows that the informal prose documentation should be trusted only to a limited extent, and rigorous models of weak memory models must rely on formal mathematics and actual testing of the processors. See [1,7].

3. Yes. Power multiprocessors exhibit a memory model much more relaxed than x86 multiprocessors: unless precise conditions are met, loads and stores can be rearranged arbitrarily by each thread. See [4,6].

4. Yes. In a Power multiprocessor stores are not propagated atomically to other processors (while x86 guarantees store-atomicity). Most accounts of weak memory models do not take into account this crucial aspect. See [4,6].

5. The behaviours illustrated by Quiz 3 and 4 imply that the given code does not implement correctly message passing. It is possible to recover a correct behaviour by inserting sync or lwsync barrier instructions between the two stores and the two loads (but isync barriers will not suffice). More efficient implementations can rely on an address dependency between the two reads and an lwsync between the two write. See [4,6]. A strategy to insert barriers to recover sequential consistency is given in [4].

6. The program should not print 1 (even according to the JSR-133 memory model), but it does when compiled with HotSpot or Gcj. We investigate concurrent high-level programming languages and compilation in [8,9].

Summary of scientific results

- A formal model of the x86 relaxed memory model. Joint publications [5] and [9];
- A formal model of the Power relaxed memory model. Joint publications [3], [6] and [8]. The tool `ppcmem` is available online: `http://www.cl.cam.ac.uk/~pes20/ppcmem`;
- Tools to explore the memory model of modern processors. Joint publications [4] and [6]. The `diy` tool suite is available as free software from `http://diy.inria.fr`;
- CompCertTSO, a verified compiler from ClightTSO to x86. Joint publication [2]. Compiler available from `http://www.cl.cam.ac.uk/~pes20/CompCertTSO`
- Tools for semantics: Ott and Lem. Joint publications [7] and [1]. Both vailable as free software from `http://moscova.inria.fr/~zappa/software/ott` and `http://www.cl.cam.ac.uk/~so294/lem`.

Students jointly supervised

1. Jade Alglave. PhD. U. Paris 7 — Denis Diderot. Thesis defended on 26 November 2010.
2. Thomas Braibant. M2 internship, 2008 (results published in 2009).
3. Kayvan Memarian. M2 internship, 2011.
4. Pankaj Pawan. MTech thesis, 2011.

Joint Publications

1. F. Zappa Nardelli, P. Sewell, J. Sevcik, S. Sarkar, S. Owens, L. Maranget, M. Batty, J. Alglave: *Relaxed memory models must be rigorous*, EC2, 2009.
2. J. Alglave, A. Fox, S. Ishtiaq, M. Myreen, S. Sarkar, P. Sewell, F. Zappa Nardelli: *The semantics of power and ARM multiprocessor machine code*, DAMP'09.
3. S. Sarkar, P. Sewell, F. Zappa Nardelli, S. Owens, T. Ridge, T. Braibant, M. Myreen, J. Alglave: *The semantics of x86-CC multiprocessor machine code*, POPL'09.
4. J. Alglave, L. Maranget, S. Sarkar, P. Sewell: *Fences in Weak Memory Models*, CAV'10.
5. P. Sewell, S. Sarkar, S. Owens, F. Zappa Nardelli, M. Myreen: *x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors*, Communications of the ACM, 53, 2010.
6. S. Sarkar, P. Sewell, J. Alglave, L. Maranget, D. Williams: *Understanding POWER Multiprocessors*, PLDI'11.
7. J. Alglave, L. Maranget, S. Sarkar, P. Sewell: *Litmus: Running Tests Against Hardware*, TACAS'11.
8. J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, P. Sewell: *Relaxed-memory concurrency and verified compilation*, POPL'11.
9. M. Batty, K. Memarian, S. Owens, S. Sarkar, P. Sewell: *Clarifying and compiling C/C++ concurrency: from C++0x to POWER*, POPL'12.
10. P. Sewell, F. Zappa Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, R. Strnisa: *Ott: Effective tool support for the working semanticist*, Journal of Functional Programming, 71, 2010.
11. S. Owens, P. Böhm, F. Zappa Nardelli, P. Sewell: *Lem: A Lightweight Tool for Heavyweight Semantics*, ITP'11.

People

INRIA: Jade Alglave (now at Oxford U.), Thomas Braibant (now at INRIA Grenoble), Luc Maranget, Kayvan Memarian (now at U. of Cambridge), Pankaj Pawan, Francesco Zappa Nardelli
University of Cambridge: Anthony Fox, Mark Batty, Peter Boehm, Magnus Myreen, Suresh Jagannathan (visiting from Purdue U.) Scott Owens, Tom Ridge (now at U. of Leicester) Susmit Sarkar, Peter Sewell, Jaroslav Sevcik (now at Microsoft), Viktor Vafeiadis (now at MPI-SWS).

Inria INVENTEURS DU MONDE NUMÉRIQUE

UNIVERSITY OF CAMBRIDGE