FAST LITMUS TESTS ON RISC V

Contents

What is a Memory Model..?

A memory model is a set of rules and conventions that dictate how operations on memory (reads and writes) are performed and observed in a multi-threaded environment.

They help in managing the complexity of concurrent access to shared memory, ensuring both correctness and performance.

Axiomatic and Operational Models

The axiomatic approach is concerned with the theoretical correctness of memory operations. It focuses on the global properties and constraints that must hold true for the system to be considered consistent

The operational approach focuses on the practical execution of operations and how they manifest in a real system. It describes the behavior of threads and the order of operations from the perspective of actual hardware or software implementations.

Strong vs Weak(relaxed) memory model

In a strong memory model, operations are executed in a strict sequence that closely follows the program order. This means that the order of memory operations specified in the code is maintained during execution.

Changes made by one thread are immediately visible to other threads. When a thread writes to a memory location, that change is instantly reflected in all other threads view of memory. This means that all threads see memory operations in the same order and are able to perceive changes to memory in a predictable way.

In a weak memory model, the order of memory operations can be relaxed or reordered. This means that the sequence of operations seen by different threads might not strictly follow the program order.

Changes made by one thread might not be immediately visible to other threads. There could be delays or inconsistencies in updates to memory become visible to other threads.Delayed visibility can lead threads to observe stale or inconsistent data, making it necessary to use and apply synchronization mechanisms and processor optimizations to ensure correct behavior.

RISC V Weak Memory Model

The RVWMO memory model is defined in terms of the global memory order, a total ordering of the memory accesses produced by all harts. In general, a multithreaded program will have many different possible executions, and each execution will have its its own corresponding global memory order.

Tool Breakdown

Herdtools7 toolsuite => Tool suite to test memory models.

Diy7 => generator used to produce litmus tests for given configurations.

Litmus7 => Tool used to run litmus tests.

Herd7 => Simulator tool for memory model.

Mcompare7 => Comparison tool for logs.

What is a Litmus Test..?

Litmus tests are small, focused programs designed to test specific behaviors of a memory model. In the context of memory consistency models, litmus tests help illustrate how different memory operations might interact under various conditions.

A litmus test source has three main sections:

1. The initial state defines the initial values of registers and memory locations. Initialisation to zero may be omitted.

2. The code section defines the code to be run concurrently

3. The final condition applies to the final values of registers and memory locations.

Litmus Test Breakdown

```
RISCV MP                        ──────────→  Test Name
"PodWW Rfe PodRR Fre"           ──────────→  Axiom for thread program order
{
 0:x5=1; 0:x6=x; 0:x7=y;        ──────────→  Initialization
 1:x6=y; 1:x8=x;
}
 P0          | P1          ;    ──────────→  Two Harts namely P0 & P1 and their operations
 sw x5,0(x6) | lw x5,0(x6) ;
 sw x5,0(x7) | lw x7,0(x8) ;
exists (1:x5=1 /\ 1:x7=0)       ──────────→  Test Condition
```

Test Execution

The following command is used to execute a single test

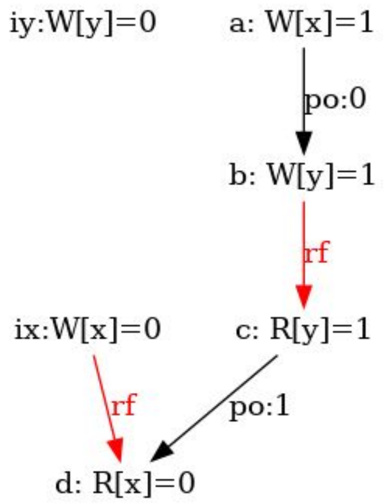
litmus7 MP.litmus (for hardware results)

herd7 MP.litmus (for herd simulator results)

Output from herd7

```
$ herd7 -I /home/vlab/.opam/default/share/herdtools7/herd/ -model ../../../model/rvwmo/riscv-ppo.cat -show prop -doshow prop
-graph free -showevents memory -squished true -showlegend false -o img MP.litmus
$ cd img
$ dot -T png MP.dot -o MP2.png
$ xdg-open MP2.png
```

```
Test MP Allowed
States 4
1:x5=0; 1:x7=0;
1:x5=0; 1:x7=1;
1:x5=1; 1:x7=0;
1:x5=1; 1:x7=1;
Ok
Witnesses
Positive: 1 Negative: 3
Condition exists (1:x5=1 /\ 1:x7=0)
Observation MP Sometimes 1 3
Time MP 0.01
Hash=2939da84098a543efdbb91e30585ab71
```



Output from litmus7

```
% Results for tests/origin/allow/MP.litmus %
RISCV MP
"PodWW Rfe PodRR Fre"
{
 0:x5=1; 0:x6=x; 0:x7=y;
 1:x6=y; 1:x8=x;
}
 P0           | P1       ;
 sw x5,0(x6) | lw x5,0(x6) ;
 sw x5,0(x7) | lw x7,0(x8) ;

exists (1:x5=1 /\ 1:x7=0)
Generated assembler
#START _litmus_P1
        lw a1,0(a5)
        lw a0,0(a4)
#START _litmus_P0
        sw a1,0(a5)
        sw a1,0(a4)
```

```
Test MP Allowed
Histogram (2 states)
997   :>1:x5=0; 1:x7=0;
1003  :>1:x5=1; 1:x7=1;
No

Witnesses
Positive: 0, Negative: 2000
Condition exists (1:x5=1 /\ 1:x7=0) is NOT validated
Hash=2939da84098a543efdbb91e30585ab71
Cycle=Rfe PodRR Fre PodWW
Relax MP No
Safe=Rfe Fre PodWW PodRR
Generator=diy7 (version 7.51+4(dev))
Com=Rf Fr
Orig=PodWW Rfe PodRR Fre
Observation MP Never 0 2000
Time MP 0.01
```

Fast Litmus Tests

```
git clone https://github.com/cd0912/fast-litmus-riscv.git

cd fast-litmus-riscv

make hw-tests CORES=4 GCC=riscv64-linux-gnu-gcc -j8
```

=>The above make uses 4 cores and 8 threads with risc v gcc toolchain.

```
cd hw-tests
```

Note : All the herdtools7 are preinstalled along with qemu

Obtaining and comparing of herd7 model results with litmus7 qemu results (user emulation)

qemu-riscv64-static hw-tests/run.exe > qemu.log

=>The litmus7 results will run and be saved in qemu.log.

herd7 -model ./model/riscv.cat -I ./model/rvwmo/ ./tests/@all > herd1.log

=>The herd7 model test results are generated natively for risc v memory model and saved in herd1.log.

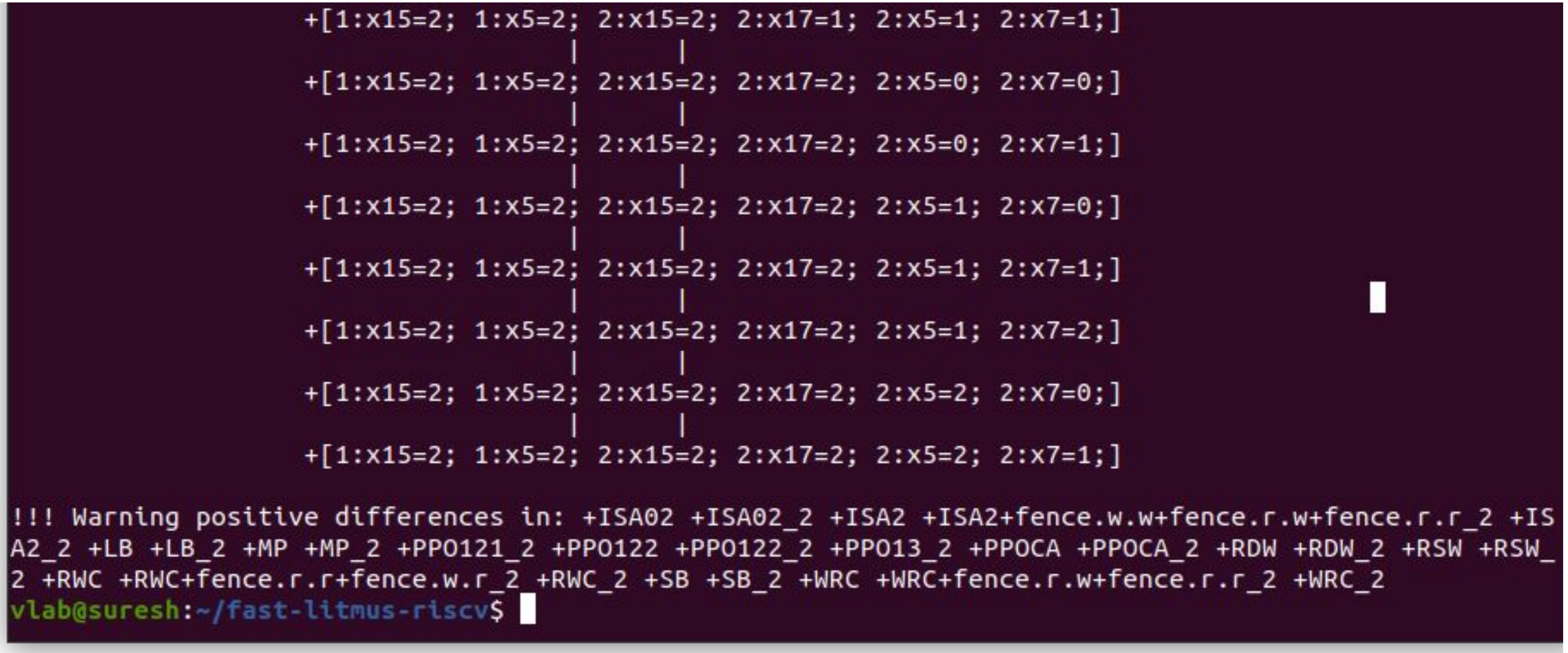
mcompare7 -nohash qemu.log herd1.log

=>This indicates that the model allows for more behaviour than was exhibited by the hardware. This is to be expected as implementations are unlikely to be as relaxed as the model permits them to be

=>Additionally the result from comparison can be saved using

mcompare7 -nohash qemu.log herd1.log > compare.log

Comparison from mcompare7 which is a prebuilt tool from herdtools7



```
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=1; 2:x5=1; 2:x7=1;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=0; 2:x7=0;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=0; 2:x7=1;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=1; 2:x7=0;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=1; 2:x7=1;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=1; 2:x7=2;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=2; 2:x7=0;]
                                   |      |
                  +[1:x15=2; 1:x5=2; 2:x15=2; 2:x17=2; 2:x5=2; 2:x7=1;]

!!! Warning positive differences in: +ISA02 +ISA02_2 +ISA2 +ISA2+fence.w.w+fence.r.w+fence.r.r_2 +IS
A2_2 +LB +LB_2 +MP +MP_2 +PPO121_2 +PPO122 +PPO122_2 +PPO13_2 +PPOCA +PPOCA_2 +RDW +RDW_2 +RSW +RSW_
2 +RWC +RWC+fence.r.r+fence.w.r_2 +RWC_2 +SB +SB_2 +WRC +WRC+fence.r.w+fence.r.r_2 +WRC_2
vlab@suresh:~/fast-litmus-riscv$
```

Logs comparison results through user mode emulation

https://drive.google.com/drive/folders/1Xtk2eZGUJ1No18OocTNNi_boxO7xJidT?usp=drive_link

Note :- These are only herd model and hardware results. Once when we setup the rmem tool we will get the flat model comparisons

No of outcomes

Running **$ litmus7 MP.litmus -a 4 -r 10 -s 100** generates 2000 outcomes. This is because:

a = 4 (number of cores or threads).

r = 10 (iterations per test).

s =100 (loop size, or the number of iterations within each thread).

N = a / t i.e 4 /2 = 2

The total number of outcomes is calculated as n × r × s. Here, n is the number of concurrent tests, r is the number of repetitions per test, and s is the size of each test.

Obtaining and comparing of herd7 model results with litmus7 qemu results (system emulation)

Running the below command generates the log files

```
$ ./run.sh
```

To compare the results we use mcompare7 tool

```
$ mcompare7 -nohash run.log herd.log
```

Logs comparison through system emulation

https://drive.google.com/file/d/1Kaqw14boULfc72hjVtNFGud_OXueG5bk/view?usp=sharing

Note :- These are only herd model and hardware results on system mode emulation. Once when we setup the rmem tool we will get the flat model comparisons

Observation

If you see at the end of the output the line "!!! Warning negative differences in: [...]", the hardware has exhibited some behaviour that the model does not allow. This indicates that the hardware is inconsistent with the RISC-V RVWMO memory model.

You are most likely to see the line "!!! Warning positive differences in: [...]". This indicates that the model allows for more behaviour than was exhibited by the hardware. This is to be expected as implementations are unlikely to be as relaxed as the model permits them to be. In addition, it is possible that the test harness just did not trigger the right conditions for a certain behaviour to be exhibited by the hardware.

Test Verification

```
$ make verify-allow-SB
```

```
$make verify-forbidden-PPO13
```

There is a verification tool written in python which verifies the test using herd7 tool which determines the test is allowed or forbidden by the risc v memory model

THANK YOU