Understanding POWER Multiprocessors

Sushmit Sarkar, Peter Sewell, Jade Alglave, Luc Maranget, Derek Williams

Presented by
Akshay Gopalakrishnan

November 10, 2021

Introduction

- Much of the performance in hardware comes due to features such as Read/Write buffers, Speculation, Caches, etc.
- The behavior of execution of programs utilizing all such features in a hardware can be defined by a relaxed memory consistency model.
- ARM, x86, POWER.. all these hardware exhibit relaxed behaviors.
- Of these POWER is relatively well understood.
- Major reason is due to informal specification and behaviors described only via litmus tests.
- Continual hardware improvements in POWER also have resulted in more relaxed behaviors.

Goal

- This paper analyzes the POWER multiprocessor family for relaxed behaviors explain their discovered behaviors via an Abstract Machine.
- This helps in avoiding getting involved in the complexities of hardware itself while all the way understanding the weak beahviors exhibited by them.
- The abstract machine has only one storage subsystem while having read/write buffers for each thread.

Preliminary Definitions

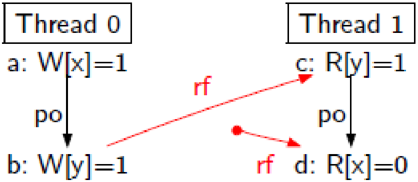
Axiomatic events

- Read - $R[x]$
- Write - $W[x]$

Binary Relations

- Program order - po (per thread syntactic order)
- Reads from - rf (from a write to a read whose value is the write)

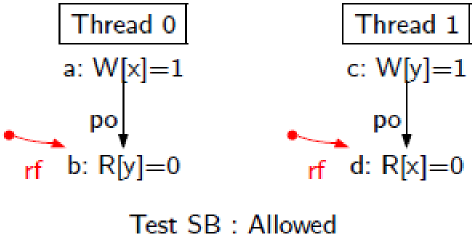
Example 1: Message Passing



Test MP : Allowed

The outcome in program is not allowed under sequential execution, but in POWER such an outcome is allowed. In real program, the read to y can be in a loop, which only ends once it reads 1, indicating that the value of x has been modified. But this behavior should then clearly not be possible. But it is.

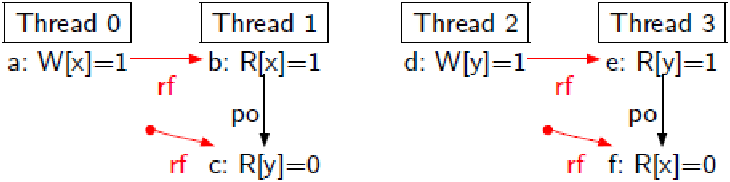
Example 2: Store Buffering



Similarly the outcome in the program above is also allowed under POWER. Such an outcome is also observable in x86-TSO.

Example 3: IRIW

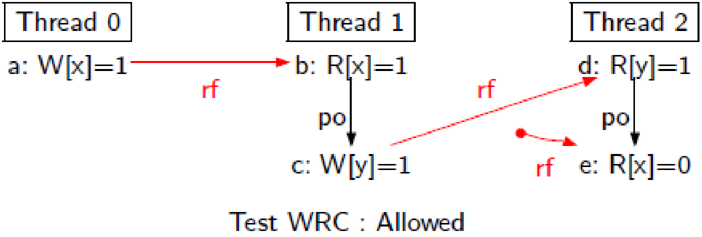
POWER also allows writes to be non-multi-copy-atomic (nmca). Meaning writes can be visible to processors in different orders. The following example showcases an outcome that should not be possible in multi-copy-atomic hardware.



Test IRIW : Allowed

Example 4: WRC

Another flavor of nmca is the following example



In this example, the read value of x is 1, meaning the write has been propagated to Thread 1. Subsequently, Thread 2 reads from a write *po* after the read to x in Thread 1. However, this does not ensure that Thread 2 has also received the updated value of x. The outcome above is allowed in POWER.

Use of Synchronization and other Barriers

The above behaviors of programs may not be desirable in all situations. To ensure this, POWER also comes with barrier instructions: mainly *sync* and *lwsync*. Let us see how these barriers can be used to ensure ordering constraints.

MP + sync