Virtual Machines on RISC-V

Jack Andrew[1]

[1]Imagination Technologies Ltd

Abstract

This extended abstract is submitted as a poster submission for the RISC-V Summit Europe, Barcelona, 5th-9th June 2023. It is assumed that the reader has a basic understanding of RISC-V and computer architecture. The poster aims to be a primer into virtual machines on RISC-V from a hardware/software co-design perspective. It introduces virtual machines and the associated software concepts. The poster closes by analysing two virtual machine topics that are critical to the commercial success of RISC-V and how they might be addressed in hardware and software.

Introduction

Virtualization is a critical technology for modern computing, allowing multiple operating systems and applications to run simultaneously on a single physical machine. Virtual machines have become increasingly popular due to their benefits, such as increased hardware utilization, enhanced security, and easier deployment of complex software stacks. RISC-V is an open-source Instruction Set Architecture (ISA) that has been gaining popularity in recent years due to its simplicity, modularity, and extensibility. This poster presents an overview of virtual machines, describes the current state of virtual machines on RISC-V and the critical challenges around implementing virtual machines on RISC-V. This poster focuses primarily on ISA emulation and there are undoubtedly opportunities to extend this to traps, OS emulation and multi-processor virtualization.

Overview of Virtual Machines

A virtual machine is a software emulation of a computer system, providing a virtual environment that behaves as if it were a physical machine. A virtual machine is a very broad topic and covers a wide range of implementations from interpreters (e.g. Perl), binary translators (e.g. Rosetta), high-level language virtual machines (e.g. JVM, Android) to full system virtual machines (e.g. Oracle VM Server). There are some common characteristics of all these virtual machines which will be used to discuss the state of virtual machines on RISC-V.

ISA emulation is a key aspect of many virtual machines in that they must support executing a program binary built for a different ISA than the host machine (e.g. an x86 binary running on an Apple MacBook M1).

Interpreting is the process of emulating the complete architected state of a source ISA or bytecode on a host machine. Interpreting is slow as it requires conversion of source ISA or bytecode at runtime but is still commonly use in virtual machines due to its fast start-up performance. Binary translation and dynamic binary optimization are used on subsequent passes of the code.

Binary Translation is the process of converting a source binary program into a target binary program. This can offer significant speedups over interpreters but requires increased startup time for initial translation and increased memory for storing translated code into a code cache. Therefore, it's common to perform a first pass interpretation while translating and optimizing the incrementally.

Dynamic Binary Optimization is the process of identifying frequently executed (*hot)* code regions via dynamic profiling. Code regions identified as hot will undergo further optimizations as the cost of optimization is likely to be amortized over frequent runs. This relies on both hardware and software profiling features at runtime.

State Mapping is the process of mapping source ISA registers to target ISA registers and/or memory. Depending on the similarity of the ISAs and the number of registers in each, the state mapping can be a cause significant overhead due to *spill and fill* of the registers into memory.

Virtual Machines Critical to Success

Virtual machines are going to be critical to the commercial success of RISC-V. Parallels can be drawn with Arm's success in the laptop and consumer markets via Apple and Android respectively.

Apple has reported 70% year-on-year grown in Q2 2021 MacBook sales which is largely correlated with the release of Arm based MacBooks, without emulation this would have been impossible as it eased the software / hardware transition from x86 based MacBooks. The majority of software written for MacBooks targets x86-64 and therefore to support this software, Apple developed a dynamic binary translator named Rosetta 2. In-order for RISC-V to be successful in laptops, a dynamic binary translator with equivalent performance and compatibility is likely required. This highlights the importance of emulating the x86 ISA on RISC-V hardware and hence will be focus of the following

section. At present there are a number of open-source and commercial efforts to produce an x86 binary translator for RISC-V and an analysis of this landscape would be an interesting exercise for future work.

If the move to Arm based laptops continues, then the emulation of aarch64 on RISC-V is likely to be of equal importance in the future but is not a focus of this poster.

Android on RISC-V has been much talked about in recent years, with Alibaba making a lot of progress in porting Android to RISC-V and Google announcing their desire to make RISC-V a "tier 1 platform" on Android. Given Android's increasing ubiquity in consumer devices such as digital TV, smarthubs and mobiles. RISC-V needs to port a performant and standardized version of Android to penetrate these markets. In-order to do this a solution to RISC-V's lack of instruction and data consistency is required.

Emulating x86 on RISC-V

Emulating a CISC ISA binary, such as x86, on a RISC ISA host machine, such as RISC-V, is much preferred than the inverse. Complex instructions can be broken down into their constituent simple RISC instructions. For example, a single x86 load and add instruction can be broken down into separate RISC-V load and add instructions.

The number of general-purpose registers is important in ISA emulation for determining the overhead of state mapping. RV64I implements 31 general purpose registers (x1-31), a zero register (x0) and a dedicated program counter (pc). x86-64 implements 16 general purpose registers (including stack pointers, link register) and a dedicated program counter (IP). Therefore, it's possible to map most x86-64 registers directly to RV64I registers with spare registers to be used by emulation code. This will minimize the *spills and fills* needed during emulation.

A notable omission from the RISC-V ISA are condition codes and these will present a challenge when emulating x86 code on RISC-V. x86 implements condition codes that are implicitly updated as the side effect of many instructions. The lack of condition codes in RISC-V means x86 condition codes will need to be emulated when performing arithmetic instructions which is a significant overhead. A common optimization that can reduce this overhead is lazy evaluation which exploits the fact that condition codes are rarely used and therefore only evaluated at the point they are consumed.

x86 supports total store ordering (TSO) where broadly speaking it implements a strongly ordered memory model with the allowance for local store buffers (giving significantly improved store performance). RISC-V (and Arm for that matter) implement a weak memory ordering model (WMO) which does not maintain the same guarantees as TSO but allows for even further optimizations in hardware. Emulating TSO on hardware supporting WMO comes at a significant performance penalty due to the number of memory fences needed to achieve strongly ordered guarantees. Therefore, Apple M1 has implemented TSO in hardware to achieve better emulated x86 performance. The extensibility of RISC-V allows for TSO to be natively implemented in hardware (Ztso) and implementations hoping to performantly emulate x86 should include this extension.

Fortunately for RISC-V, it supports the same endianness as x86 (little-endian) which can cause major headaches for emulation as was seen in Rosetta 1 when emulating PowerPC (big-endian) apps on x86 (little-endian).

Other notable areas that need to be considered but have not as part of this extended abstract are FP formats, atomics, memory management, memory protection, memory addressing and memory alignment.

Instruction & Data Consistency

There are countless reasons a virtual machine would desire to modify its instruction memory at runtime. For example, binary translations maintain a code cache for translated blocks of code in memory which cannot practically contain all translated code due to memory size limitations. These code caches are flushed, most commonly when full or in some form of FIFO mechanism.

Alternatively, on interrupts that cause transitions back to the host, the guest application must take the interrupt at a point where architectural state is as if it were running natively and service the interrupt in a reasonable time. The translated code, often executed as *superblocks*, may need to be modified to more quickly reach an interruptible state.

Therefore, processors must keep an up-to-date view of instruction memory in order to correctly execute the virtual machine.

The RISC-V ISA intends to support implementations with incoherent instruction caches, post I-cache buffering and execution pipe containing fetch instructions. Therefore, the lack of a ratified instruction / data consistency extension (e.g. Zjid) and it's omission from the RVA23 profile is a limitation for RISC-V implementations hoping to run virtual machines performantly without the overhead of full hardware instruction / data coherency, note that even on these machines a subset of the Zjid extensions are required.

The overhead of instruction/data coherency would likely be a significant area cost in the highly area / cost sensitive implementations in embedded consumer devices such as digital which Android is ever gaining popularity in.

Admittedly, as implementations scale to larger multi-core systems, e.g. server cores, the cost of instruction / data coherency hardware is likely preferred over tracking data to the point of coherency for all cores via the Zjid extension.

References

[1] J. E. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann Publishers, 2022.

[2] D. Williams, G. Magklis, M. Maas and P. Loewenstein. Instruction/Data (I/D)) consistency proposal for RISC-V.