Enabling RISC-V support on MaxineVM

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
Enabling RISC-V support on MaxineVM

Foivos S. Zakkak  
The University of Manchester  
United Kingdom, M13 9PL  
foivos.zakkak@manchester.ac.uk

Juan Fumero  
The University of Manchester  
United Kingdom, M13 9PL  
juan.fumero@manchester.ac.uk

Christos Kotselidis  
The University of Manchester  
United Kingdom, M13 9PL  
christos.kotselidis@manchester.ac.uk

ABSTRACT
In this paper we outline the current state of language Virtual Machines (VMs) running on RISC-V as well as our initiatives in augmenting the existing ecosystem with Maxine VM, a state-of-the-art open source research Virtual Machine (VM). Maxine VM is a metacircular VM for Java and is currently part of the Beehive ecosystem that provides a unified framework for hardware/software co-designed research on managed languages runtimes.

1 INTRODUCTION
Recently we are witnessing a trend towards providing open source modular language virtual machines. This trend is driven mainly by the need to reuse successful components across different VMs. Hence, numerous virtual machines such as Oracle’s HotSpot [5], IBM J9 [6], .NET [10], Google v8 [4], and RPython [2], have been recently open sourced. In addition, projects like Eclipse OMR [3], Graal VM [13], and Mu [11] provide a set of core components useful to VM implementers, where each component is self-contained and able to interface with others through well-defined APIs.

Regarding RISC-V support, only the Jikes research Virtual Machine [9] can currently execute some workloads with its baseline compiler. In this work we present our efforts to bring RISC-V support to another research virtual machine, namely the MaxineVM [7, 12].

2 MAXINE VM AND THE BEEHIVE ECOSYSTEM
MaxineVM features mature compilers that support both 64 and 32 bit execution modes for x86 and ARM architectures. In addition, MaxineVM is a core component of the Beehive [8] project that aims in developing a whole ecosystem of tools and runtimes for conducting research on managed runtimes on emerging heterogeneous architectures. As a result, porting MaxineVM to RISC-V will enable parts of the Beehive ecosystem (depicted in Figure 1) to be used on VM-related RISC-V research projects.

The goal of the Beehive Ecosystem is to provide a framework with the following characteristics regarding hardware/software co-design:

- Modular and easily extensible.
- Implemented with high level languages with good IDE support and low entry barrier.
- Realistic and diverse simulation infrastructures.
- Support of multiple hardware architectures.
- Support of heterogeneous systems.
- Capability of implementing multiple languages.
- Integration with popular research tools.

On the top layer of the stack are the groups of applications that Beehive aims to improve the state of the art for. These applications range from the standard benchmark suites, to applications running on top of Big Data frameworks, including domain specific applications and implementations of managed languages.

The runtime layer (second from top) incorporates all the runtime mechanisms necessary to execute a managed language. This layer unifies, under the same compilers and runtimes, high-quality polylong production and research VMs. It will feature two VMs, Maxine and OpenJDK HotSpot, that share a common optimizing compiler, Graal, and the Truffle runtime framework. OpenJDK HotSpot produces the production VMs, while MaxineVM [12] is a meta-circular research VM. MaxineVM will ultimately be compatible with the Java Virtual Machine Compiler Interface (JVMCI), and the JikesRVM’s Memory Management Toolkit (MMTk) [1], combining two powerful interfaces that will enable experimentation with different compilers and garbage collectors.

2.1 RISC-V support
Currently, MaxineVM supports ARMv7 and x86 architectures with the AArch64 support being underway. In addition, new initiatives to port MaxineVM and hence part of the Beehive ecosystem to RISC-V have been made. To that end, as of MaxineVM release 2.3, we have ported MaxineVM’s CrossISA toolkit [7] to RISC-V. The

Figure 1: The Beehive ecosystem
CrossISA toolkit enables the rapid prototyping of new ISA ports (assemblers and compilers) via an automated manner. The CrossISA toolkit utilizes the QEMU port to RISC-V to simulate bare-metal tests that are generated using the MaxineVM’s compilers code. We have also created a screencast\(^2\) to demonstrate the usage of the tools and showcase the process of porting the assembler to RISC-V. We anticipate that with strong community support we will manage to port and execute MaxineVM on RISC-V in a timely manner while the addition of JVMCI/Graal and MMTk [1] will enact research on RISC-V for a plethora of managed languages (supported by Truffle) and garbage collection algorithms (supported by MMTk). MaxineVM is fully open-sourced at: https://github.com/beehive-lab/Maxine-VM.

REFERENCES


\(^2\)https://youtu.be/K-BZpAX_dvY