# MaxineVM
# Enabling HW/SW Co-design of managed languages on RISC-V

Christos Kotselidis

Associate Professor at The University of Manchester

Senior Architect at KTM Innovation

# Why Java on RISC-V?

- Second on the TIOBE index (> 10%)
- Platform independent
- High-level, developer-friendly, managed
- Pervasive
  - From IoT/SoC to high-end server applications

# State of other JVM RISC-V ports

- OpenJDK Zero – only interpreter
- JikesRVM – not publicly available port
- OpenJ9 – port effort ongoing; not full JIT compilation yet
- OpenJDK11 – Initial port from Huawei
- MaxineVM – First implementation to support full JIT compilation and wide benchmark coverage

# MaxineVM History

- Project started in 2005 at Sun Microsystems
- Through the years, the MaxineVM compiler branched out and transformed to the Graal compiler
- Since 2017, the University of Manchester continues the development of MaxineVM
  - https://github.com/beehive-lab/Maxine-VM

# MaxineVM Characteristics

- Metacircular VM – VM for Java written in Java
- Mainly used for research and not production
  - Easy to change and experiment
  - ~2x slower than production quality JVMs
- Multiple plug-and-play “schemes”
  - HeapScheme, LockScheme, ObjectLayoutScheme, etc.
- Compilers, GCs
  - T1X: Template interpreter
  - C1X: Optimizing compiler (based on HotSpot’s C1 compiler)
  - SemiSpace and Generational GCs

# MaxineVM on RISC-V

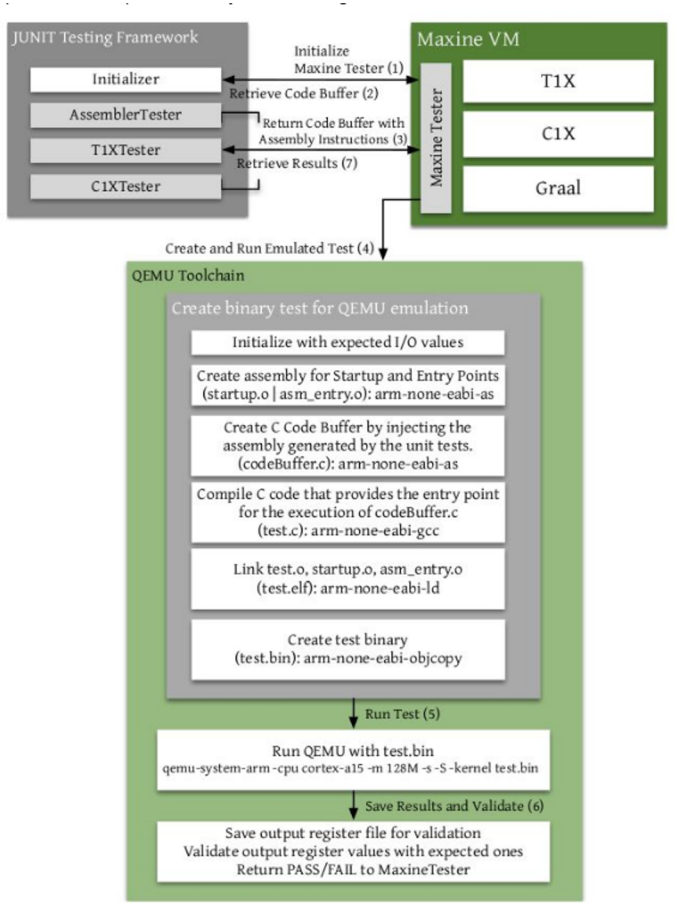
What had to be accomplished to run Maxine on RISC-V?

- Implement the RISC-V ISA instructions encodings used by the compilers in the VM assembler
- All RISC-V extensions which provide ISA instructions from spec v2.2 were used except:
  - V Vector Operations, P Packed SIMD, C Compressed Instructions, Q Quad Precision
- Port to RISC-V
  - T1X and C1X compilers
  - Adapters – make the transition between two calling conventions in use by the compilers
  - Stubs – hand crafted assembly that cannot be expressed as Java
  - Substrate – underlying C code which covers VM boot up, endianness, signal handlers registration, etc

# MaxineVM on RISC-V

## How the port was done?

- Testing infrastructure in place; allows for offline compiler testing




**Cross-ISA Debugging in Meta-circular VMs**


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

Andy Nisbet
The University of Manchester
Manchester, United Kingdom
andy.nisbet@manchester.ac.uk

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

Nikos Foutris
The University of Manchester
Manchester, United Kingdom
nikos.foutris@manchester.ac.uk



**Abstract**

Extending current Virtual Machine implementations to new Instruction Set Architectures entails a significant programming and debugging effort. Meta-circular VMs add another level of complexity towards this aim since they have to compile themselves with the same compiler that is being ex-



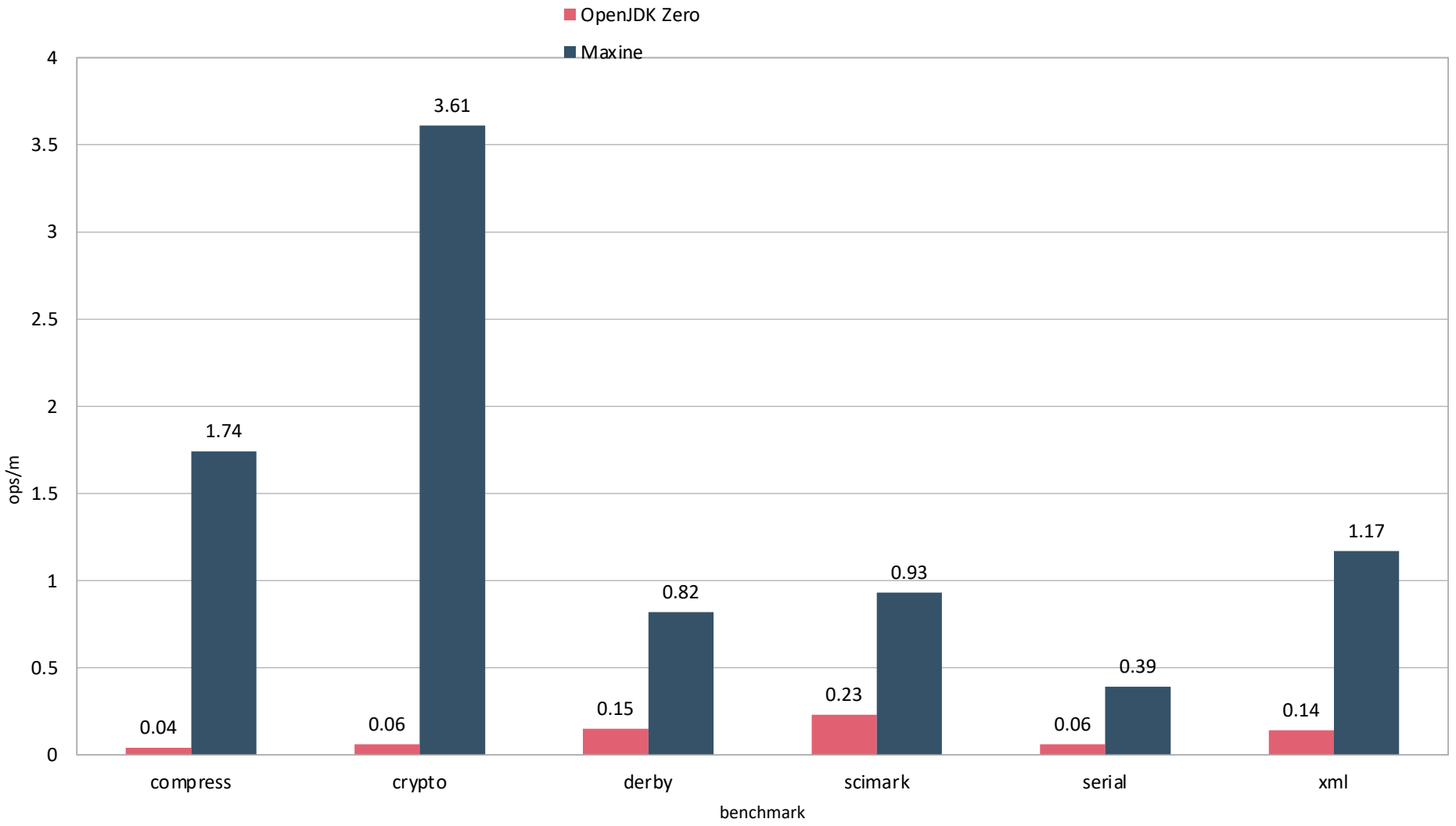
**ACM Reference Format:**
Christos Kotselidis, Andy Nisbet, Foivos S. Zakkak, and Nikos Foutris. 2017. Cross-ISA Debugging in Meta-circular VMs. In *Proceedings of ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3141871.3141872

# MaxineVM on RISC-V

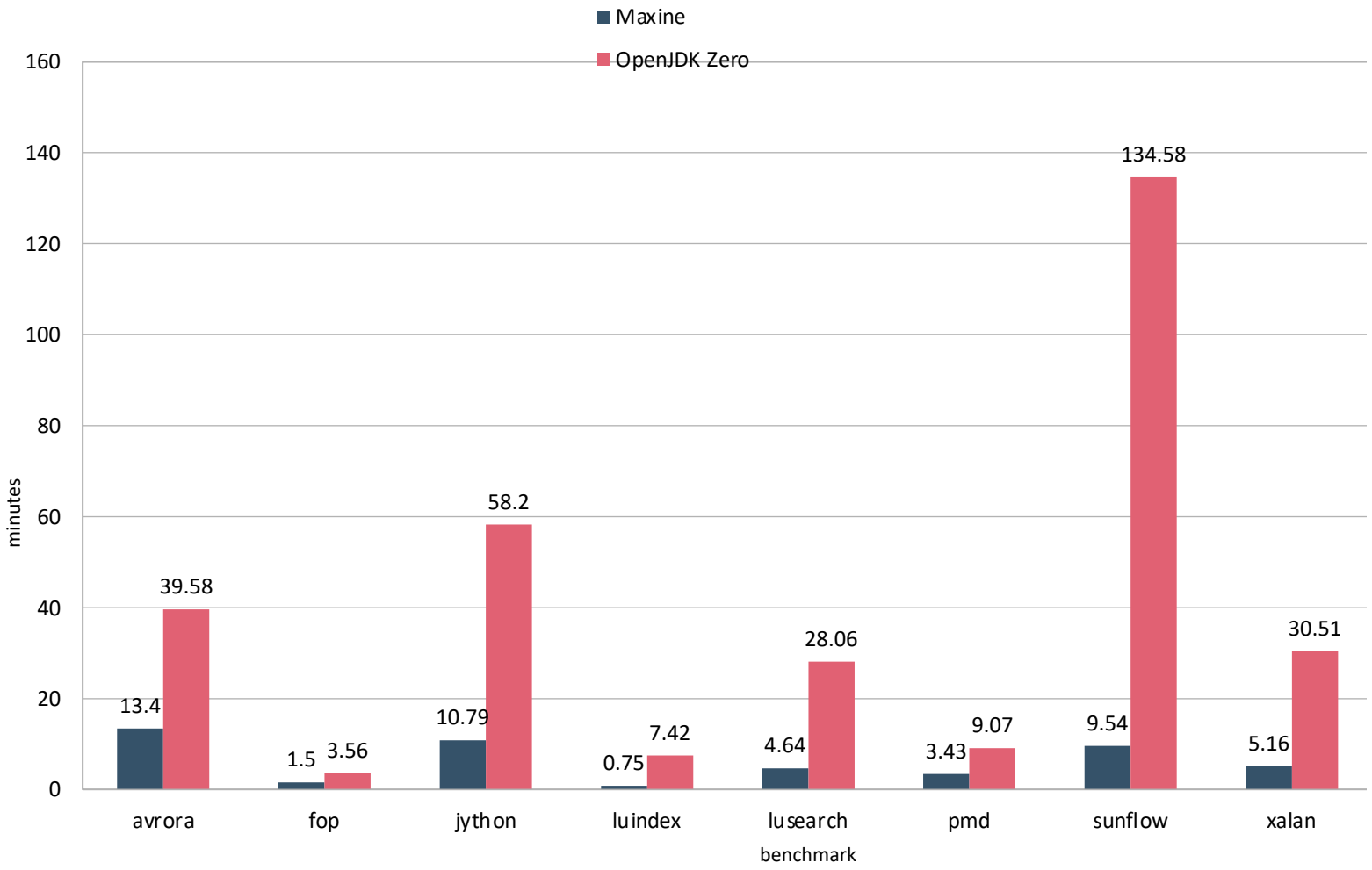
## Benchmarks

- Fedora 31 on top of Qemu
- SPECjvm2008 → 79% pass rate
- DaCapo → 60% pass rate
- Comparison against OpenJDK Zero

# Results – SpecJVM2008



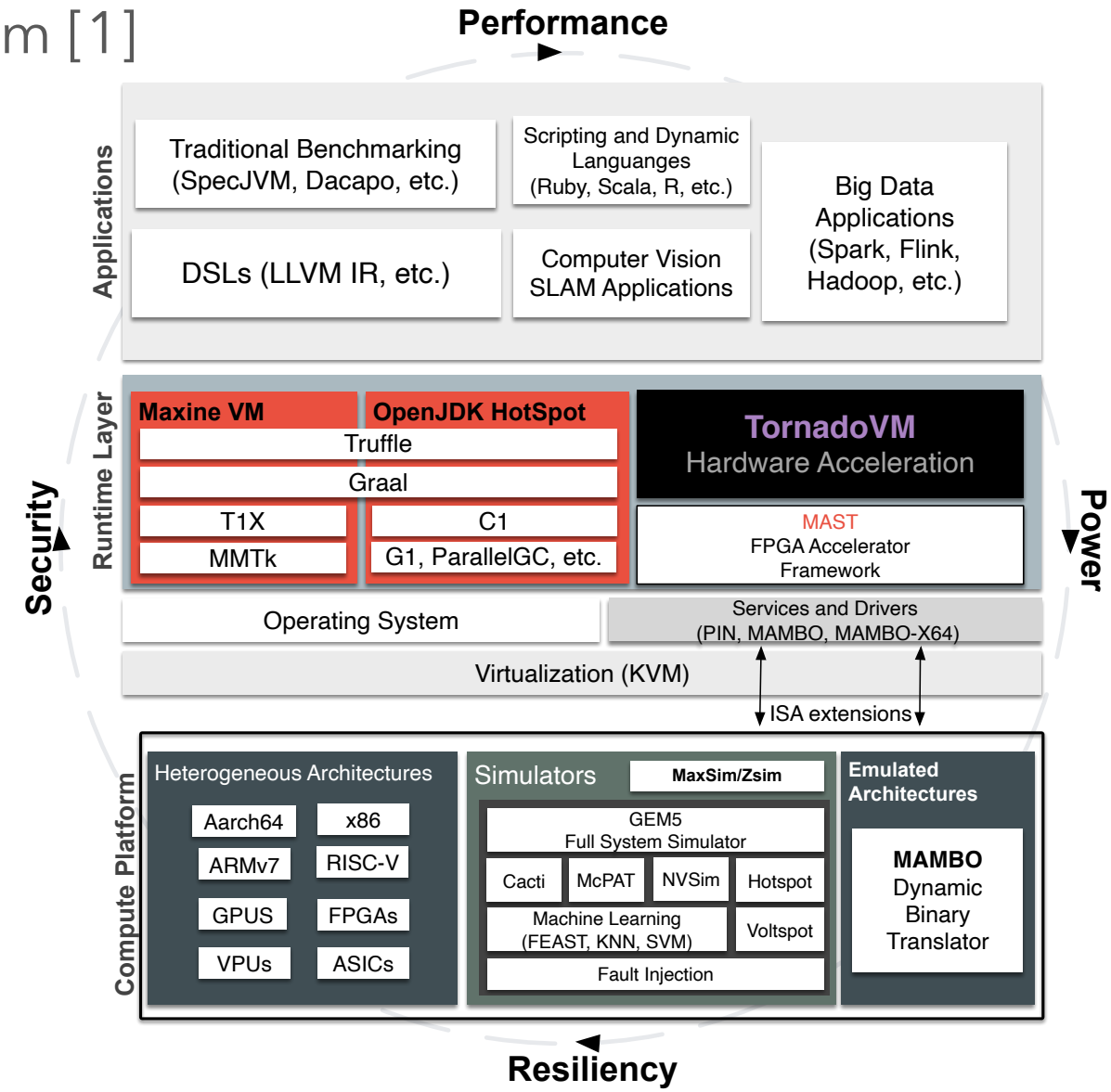
# Results – Dacapo



# Ongoing Research

- MaxineVM is part of the Beehive ecosystem [1]
- https://github.com/beehive-lab

*"Provide an extensible state-of-the-art infrastructure for hw/sw co-design research"*




- Safe code modification on architectures without HW support for SMC [2]
- Type-information elimination on archs with tagged pointers [3]
- Hardware acceleration of managed languages on GPUs and FPGAs [4, 5]
- Systems for co-simulation of managed applications [6]
- Memory characterization of managed languages on NUMA archs [7]

# References


[1] C. Kotselidis, et al., “Project Beehive: A HW/SW co-designed stack for runtime and architectural research”, https://arxiv.org/abs/1509.04085
[2] T. Hartley, et al., “An Analysis of Call-Site Patching without Strong Hardware Support for Self-Modifying-Code”, In MPLR 2019
[3] A. Rodchenko, et al., “Type Information Elimination from Objects on Architectures with Tagged Pointers Support”, In IEEE TOC 2018
[4] J. Clarkson, et al, “Exploiting High-Performance Heterogeneous Hardware for Java Programs using Graal”, In ManLang 2018
[5] C. Kotselidis, et al., “Heterogeneous managed runtime systems: A computer vision case study”, In VEE 2017
[6] A. Rodchenko, et al., “MaxSim: A Simulation Platform for Managed Applications”, In ISPASS 2017
[7] O. Papadakis, et al., “You can’t hide you can’t run: a performance assessment of managed applications on a NUMA machine”, In MPLR 2020

# Thank you!