# On the Future of Research VMs: A Hardware/Software Perspective

Foivos S. Zakkak, Andy Nisbet, John Mawer, Tim Hartley, Nikos Foutris, Orion Papadakis, Andreas Andronikakis, Iain Apreotesei, Christos Kotselidis
The University of Manchester
United Kingdom, M13 9PL
first.last@manchester.ac.uk

## ABSTRACT

In the recent years, we have witnessed an explosion of the usages of Virtual Machines (VMs) which are currently found in desktops, smartphones, and cloud deployments. These recent developments create new research opportunities in the VM domain extending from performance to energy efficiency, and scalability studies. Research into these directions necessitates research frameworks for VMs that provide full coverage of the execution domains and hardware platforms. Unfortunately, the *state of the art* on Research VMs does not live up to such expectations and lacks behind industrial-strength software, making it hard for the research community to provide valuable insights.

This paper presents our work in attempting to tackle those shortcomings by introducing *Beehive*, our vision towards a modular and seamlessly extensible ecosystem for research on virtual machines. Beehive unifies a number of existing state-of-the-art tools and components with novel ones providing a complete platform for hardware/software co-design of Virtual Machines.

## CCS CONCEPTS

• **Software and its engineering → Virtual machines**;

## KEYWORDS

Virtual Machines, Managed Runtime Systems, Heterogeneous Systems, Simulators

**ACM Reference Format:**
Foivos S. Zakkak, Andy Nisbet, John Mawer, Tim Hartley, Nikos Foutris, Orion Papadakis, Andreas Andronikakis, Iain Apreotesei, Christos Kotselidis. 2018. On the Future of Research VMs: A Hardware/Software Perspective. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3191697.3191729

## 1 INTRODUCTION

The last few years there is a movement towards providing open source modular *language virtual machines*. Mainly driven by the need to enable the reuse of successful components across different VMs, numerous virtual machines such as Oracle's HotSpot [11], IBM J9 [12], .NET [19], Google v8 [8], and RPython [5], have been recently open sourced. In addition, projects like Eclipse OMR [7] and Mu [24] 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 interfaces. This allows various combinations of different implementations of components to create a VM that matches the needs of each specific case. Graal VM [26], on the other hand, combines an efficient *just in time* (JIT) compiler with a language implementation framework to allow multiple languages to be efficiently implemented on top of a Java Virtual Machine (JVM).

These solutions focus on solving specific issues and often for specific environments; mainly targeting high peak performance. However, since today's VMs are exercising a wide range of devices ranging from cell phones to powerful clusters, diverse stacks that cover all environments are needed. Such solutions need to expand beyond a single node and/or hardware architecture enabling research on complex systems (e.g. clusters or cloud) with heterogeneous machines and enormous amounts of memory [1, 10].

## 2 THE BEEHIVE ECOSYSTEM

Our vision is to take advantage of the movement towards open source and modular VMs and form an ecosystem of tools that will enable the next generation of research on VMs. In particular, we envision an ecosystem with the following characteristics:

- 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.

We believe that such a research VM will provide a solid foundation for the research community. It will also provide a common baseline for comparing different works and it will bring the community closer, ultimately improving the quality of the conducted research.

Figure 1 visualizes the interconnection of tools that comprise the *Beehive* ecosystem [15]. On the top layer of the stack are the groups of applications that the ecosystem 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.

<Programming'18> Companion, April 9–12, 2018, Nice, France

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 polyglot 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 represents the production VMs, while MaxineVM [25] 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) [4], combining two powerful interfaces that will enable experimentation with different compilers and garbage collectors. The runtime layer will also include in-house software components and mechanisms [2, 13] that enable the acceleration of applications on specialized hardware, like SIMD hardware extensions, GPUs, FPGAs, etc. In order to support multiple ISAs, MaxineVM and its compilers are being ported to different architectures. Currently, Beehive executes on x86 and ARMv7 architectures while ongoing work extends the support for AArch64 and RISC-V via the CrossISA toolkit [14].

The layer below the runtime layer consists of the Operating System, the Services and Drivers, and the Virtualization components. The services and drivers component integrates a number of binary instrumentation tools to enable research and rapid prototyping of novel micro-architectures and ISA extensions [9, 17]. These tools will allow for code injection, binary translation and instrumentation, ISA extensions, and others.

Beehive integrates a variety of simulators providing trade-off selections between simulation speed (i.e. functional), simulation accuracy (i.e. cycle-accurate), and engineering efforts required to modify or implement new hardware timing models (i.e. software or hardware simulators). Therefore, Beehive integrates a software-based full-system, a software-based user-level, and a hardware-based user-level simulator; namely Gem5, ZSim, and APTSim respectively.

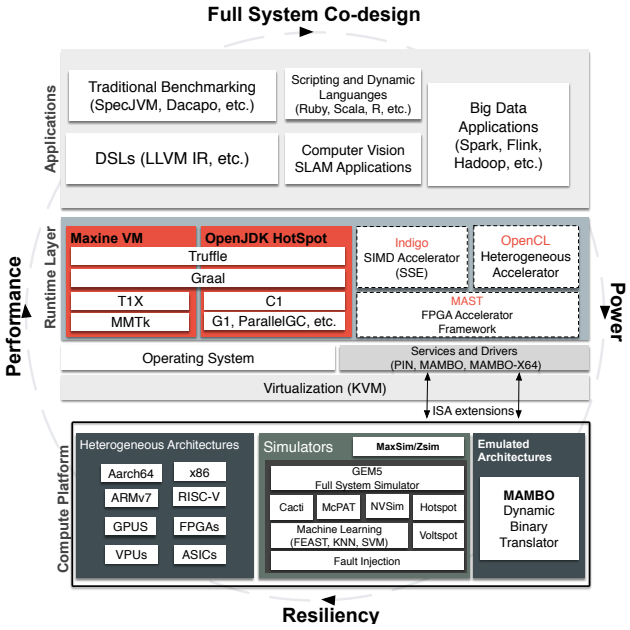


**Figure 1: The Beehive ecosystem**

Gem5 [3] is a software-based full-system simulation framework capable of modeling numerous hardware architectures. Beehive augments Gem5 with an interface that allows it to integrate with a variety of other simulators, such as McPAT [16] (power), HotSpot [23] (thermal), Voltspot [28] (power), and NVSim [6] (circuit-level). In addition, Beehive also incorporates machine-learning and data-analytics techniques in the Gem5 simulation framework enabling detailed analysis of the output results. Finally, Beehive augments Gem5 with a fault injection framework for reliability studies that is flexible, generic/portable, supports multi-cores, and allows for reproducible experiments.

ZSim [22] combines simulation accuracy with fast simulation times on x86 (about 10 MIPS in our experiments) and allows the execution of arbitrary managed workloads via lightweight user-level virtualization. MaxSim[1] [20] is a software-based simulation platform for x86_64 built on the Beehive ecosystem using the ZSim simulator. MaxSim can perform fast and accurate simulation of managed runtime workloads running on top of the Maxine VM and its capabilities include: 1) low-intrusive microarchitectural profiling via pointer tagging on x86-64 platforms, 2) modeling of hardware extensions related, but not limited to, tagged pointers, and 3) modeling of complex software changes via address-space morphing. Rodchenko et al. [21] use MaxSim to demonstrate that hardware software co-designed pointer tagging based optimizations can be used to eliminate the type information pointer from an object's storage.

APTSim [18] is a hardware-based simulator for ARM[2] architectures, implemented on Xilinx Zynq platforms [27]. Beehive through APTSim can directly interface unmodified application executables with FPGA hardware intellectual property (IP) and thus accelerate simulation. In APTSim functional simulation occurs natively on the ARM cores, and cycle-based timing is performed using FPGA hardware models of the memory system hierarchy and the CPU micro-architecture. The functional simulation, or an external tool must produce a trace of address loads/stores, and any program counter (PC) changes that are consumed by the FPGA timing models. In Beehive we achieve this through MaxineVM's optimizing compiler that instruments the application during JIT compilation.

Combining the above, the *Beehive* ecosystem will allow for *full system co-design* and exploration leading research towards improved performance, energy efficiency, and resiliency. The *Beehive* ecosystem is an ongoing project and its components are gradually being open-sourced at https://github.com/beehive-lab. Each individual component is being validated against standard methodologies and provide full coverage of the community-accepted benchmarks.

## ACKNOWLEDGMENTS

This work is partially supported by the EPSRC grant PAMELA EP/K008730/1, and the EU Horizon 2020 grant ACTiCLOUD 732366.

[1] https://github.com/beehive-lab/MaxSim
[2] It is equally applicable to any valid event trace and CPU ISA, as long as CPU microarchitecture models are available.

## REFERENCES


[1] 2017. Numascale NumaConnect. (2017). Retrieved Jan 10, 2018 from https://www.numascale.com/numa_pdfs/numaconnect-white-paper.pdf

[2] Colin Barrett, Christos Kotselidis, Foivos S. Zakkak, Nikos Foutris, and Mikel Luján. 2017. Experiences with Building Domain-Specific Compilation Plugins in Graal. In *Proceedings of the 14th International Conference on Managed Languages and Runtimes (ManLang 2017)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/3132190.3132207

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[4] Stephen M. Blackburn, Perry Cheng, and Kathryn S. McKinley. 2004. Oil and Water? High Performance Garbage Collection in Java with MMTk *(ICSE '04)*.

[5] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin Rigo. 2009. Tracing the Meta-level: PyPy's Tracing JIT Compiler. In *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems (ICOOOLPS '09)*. ACM, New York, NY, USA, 18–25. https://doi.org/10.1145/1565824.1565827

[6] Xiangyu Dong, Cong Xu, Norm Jouppi, and Yuan Xie. 2014. *NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Non-volatile Memory*. Springer New York, New York, NY, 15–50. https://doi.org/10.1007/978-1-4419-9551-3_2

[7] Eclipse Foundation. 2017. Eclipse OMR. (2017). Retrieved Jan 10, 2018 from https://www.eclipse.org/omr/

[8] Google. 2017. v8. (2017). Retrieved Jan 10, 2018 from https://developers.google.com/v8/

[9] Cosmin Gorgovan, Amanieu d'Antras, and Mikel Luján. 2016. MAMBO: A Low-Overhead Dynamic Binary Modification Tool for ARM. *ACM Trans. Archit. Code Optim.* 13, 1, Article 14 (April 2016), 26 pages. https://doi.org/10.1145/2896451

[10] Georgios Goumas, Konstantinos Nikas, Ewnetu Bayuh Lakew, Christos Kotselidis, Andrew Attwood, Erik Elmroth, Michail Flouris, Nikos Foutris, John Goodacre, Davide Grohmann, Vasileios Karakostas, Panagiotis Koutsourakis, Martin Kersten, Mikel Luján, Einar Rustad, John Thomson, Luis Tomás, Atle Vesterkjaer, Jim Webber, Ying Zhang, and Nectarios Koziris. 2017. ACTiCLOUD:Enabling the Next Generation of Cloud Applications. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*.

[11] The OpenJDK HotSpot Group. 2017. OpenJDK HotSpot. (2017). Retrieved Jan 10, 2018 from http://openjdk.java.net/projects/jdk/

[12] IBM and Eclipse Foundation. 2017. Eclipse OpenJ9. (2017). Retrieved Jan 10, 2018 from https://www.eclipse.org/openj9/

[13] Christos Kotselidis, James Clarkson, Andrey Rodchenko, Andy Nisbet, John Mawer, and Mikel Luján. 2017. Heterogeneous Managed Runtime Systems: A Computer Vision Case Study. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '17)*. ACM, New York, NY, USA, 74–82. https://doi.org/10.1145/3050748.3050764

[14] Christos Kotselidis, Andy Nisbet, Foivos S. Zakkak, and Nikos Foutris. 2017. Cross-ISA Debugging in Meta-circular VMs. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL 2017)*. ACM, New York, NY, USA, 1–9. https://doi.org/10.1145/3141871.3141872

[15] Christos Kotselidis, Andrey Rodchenko, Colin Barrett, Andy Nisbet, John Mawer, Will Toms, James Clarkson, et al. 2015. Project Beehive: A Hardware/Software Co-designed Stack for Runtime and Architectural Research. *In 9th International Workshop on Programmability and Architectures for Heterogeneous Multicores (MULTIPROG)* (2015). http://arxiv.org/abs/1509.04085

[16] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2013. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Trans. Archit. Code Optim.* 10, 1, Article 5 (April 2013), 29 pages. https://doi.org/10.1145/2445572.2445577

[17] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. ACM, New York, NY, USA, 190–200. https://doi.org/10.1145/1065010.1065034

[18] John Mawer, Oscar Palomar, Cosmin Gorgovan, Andy Nisbet, Will Toms, and Mikel Luján. 2017. The Potential of Dynamic Binary Modification and CPU-FPGA SoCs for Simulation. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 144–151.

[19] .NET. 2018. Microsoft .NET. (2018). Retrieved Jan 10, 2018 from https://github.com/Microsoft/dotnet

[20] Andrey Rodchenko, Christos Kotselidis, Andy Nisbet, Antoniu Pop, and Mikel Luján. 2017. MaxSim: A Simulator Platform for Managed Applications. In *ISPASS - IEEE International Symposium on Performance Analysis of Systems and Software*.

[21] Andrey Rodchenko, Christos Kotselidis, Andy Nisbet, Antoniu Pop, and Mikel Luján. 2018. Type Information Elimination from Objects on Architectures with Tagged Pointers Support. *IEEE Trans. Comput.* 67, 1 (Jan 2018), 130–143. https://doi.org/10.1109/TC.2017.2709739

[22] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 475–486. https://doi.org/10.1145/2485922.2485963

[23] Kevin Skadron, Mircea R Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. 2003. Temperature-aware microarchitecture. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.* 2–13. https://doi.org/10.1109/ISCA.2003.1206984

[24] Kunshan Wang, Yi Lin, Stephen M. Blackburn, Michael Norrish, and Antony L. Hosking. 2015. Draining the Swamp: Micro Virtual Machines as Solid Foundation for Language Development *(SNAPL 2015)*.

[25] Christian Wimmer, Michael Haupt, Michael L. Van De Vanter, Mick Jordan, Laurent Daynès, and Douglas Simon. 2013. Maxine: An Approachable Virtual Machine for, and in, Java. *ACM TACO* 9, 4, Article 30 (Jan. 2013).

[26] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. 2013. One VM to Rule Them All *(Onward! 2013)*.

[27] Xilinx. 2016. Zynq Ultrascale+ MPSoC Product Selection Guide. (2016). Retrieved January 17 2018 from https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf

[28] Runjie Zhang, Ke Wang, Brett H. Meyer, Mircea R. Stan, and Kevin Skadron. 2014. Architecture Implications of Pads As a Scarce Resource. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA '14)*. IEEE Press, Piscataway, NJ, USA, 373–384. http://dl.acm.org/citation.cfm?id=2665671.2665728