Remarkable Challenges of High-Performance Language Virtual Machines

Guillermo Polito, Stéphane Ducasse, Pablo Tesone, Luc Fabresse, G Thomas, M Bacou, Loïc Lagadec, Pascal Cotret

Remarkable Challenges of High-Performance Language Virtual Machines

G. Polito[1], S. Ducasse[1], P. Tesone[1], L. Fabresse[2], G. Thomas[3], M. Bacou[3], P. Cotret[4], and L. Lagadec[4]

[1] *Inria Lille – Nord Europe*
[2] *Telecom Nord Europe*
[3] *Telecom Paris*
[4] *ENSTA Bretagne*

1 Language Virtual Machines : Society Assets

Language Virtual Machines (VMs) are pervasive in every laptop, server, and smartphone, as is the case with Java or Javascript. They allow application portability between different platforms and better usage of resources. They are used in critical applications such as stock exchange, banking, insurance, and health [25]. Virtual machines are an important asset in companies because they allow the efficient execution of high-level programming languages. Nowadays, they even attract investments from large non-system companies, *e.g.,* Netflix [1], Meta [2], Shopify [3] and Amazon [4].

VMs achieve high-performance thanks to aggressive optimization techniques that observe and adapt the execution dynamically, either by doing just-in-time compilation [5] or by adapting the memory management strategies at runtime [90, 91]. For all these reasons Virtual Machines are highly-complex engineering pieces, often handcrafted by experts, that mix state-of-the-art compilation techniques with complex memory management that collaborate with the underlying operating systems and hardware. However, besides some well-known techniques that are published in research venues, most knowledge and technology around virtual machines are highly concentrated in large companies such as Microsoft, Google, and Oracle, making Virtual Machine construction difficult, and experiments difficult to reproduce and replicate.

Language VMs present many **multidisciplinary scientific challenges** that appear at the intersection of fields such as hardware, system software, compiler, and software language engineering. This document aims to give a brief overview of the current challenges the VM community faces. To keep this document short, we selected *remarkable challenges* in managed execution, managed memory, performance evaluation, software engineering and security.

2 Challenges in Managed Execution

VMs achieve balance speed with portability by having a mixed-mode execution that combines interpretation techniques with dynamic compilation (a.k.a. *JIT compilation*). Peter Deutsch introduced in his seminal work the idea of runtime binary translation for programming language implementations [22], where code is dynamically compiled at runtime to remove interpretation overhead and expensive operations of object-oriented languages such as message-sends are cached inline in machine code routines. It was not until a decade later that Hölzle extended this work with *adaptive optimizations* [31] where dynamic translation takes advantage of runtime information to apply standard compiler optimizations at runtime speculatively. Since then, adaptive and speculative optimizations have been an important subject of research with a notable appearance of different speculative compilation techniques such as runtime tracing [7, 75], speculative partial evaluation [100, 98, 99], and lazy block versioning [20]. Although state-of-the-art Virtual Machines such as Google V8 [94], Oracle HotSpot [67] and Microsoft CLR [60] achieve impressive execution performance for specific use cases, there

1. `https://www.netflix.com/`
2. `https://www.meta.com/`
3. `https://www.shopify.com/`
4. `https://www.amazon.com/`

are still open research questions that are fundamental for understanding the organization of such execution engines :

Minimizing warm-up times. Speculations and adaptive optimizations require a *warm-up phase* to extract runtime information and optimize accordingly. Current trends investigate how to optimize interpreters [45, 73, 77], how to minimize dynamic compilation times [46], and how to reuse speculative decisions on subsequent executions [9, 97].

Understanding speculative compilation. Speculative compilation is based on the strong assumption that runtime behavior stabilizes. Recently in 2017, Barrett et al. [8] showed that such stabilization does not necessarily happen in most applications, producing under-performant compilations. This led to the recent study of *phased behavior* : study the phases an application goes through during its execution and would require different compilation and speculation schemas [42]. Moreover, speculative optimization uses compilation heuristics that aim to be *one-size-fits-all* which are not suitable for all applications. Recent work proposes the usage of machine learning approaches to learning application-specific optimization heuristics online, considerably increasing warm-up times [59, 58].

Hardware and Operating System Integration. Several works explore the fusion between operating systems and high-level language implementation technology to lift the security and safety properties of high-level programming languages to operating system development [15, 82, 29]. Recent work has shown that making operating systems aware of language virtual machines yields important optimization opportunities by reusing JIT compiled code between applications, and reducing start-up and warm-up times [101, 43, 97]. In the same venue, recently researchers have explored how to lift hardware support to the programming language runtimes and vice-versa [47, 89, 64].

3 Challenges in Performance Evaluation

Evaluating programming language implementations has faced since its beginning the problem of **representativity**. Programs used to evaluate performance, often called *synthetic*, do not illustrate real work-loads [79, 63]. The DaCapo benchmark suite [11] proposed in 2006 a series of programs aiming at representing typical Java programs to evaluate language implementation (JVM) performance. These suites inspired others for languages such as Scala [80], and Javascript and WebAssembly [71, 14]. Although these synthetic benchmark suites are currently in use by language implementation researchers, they are still not considered representative of realistic workloads, and state-of-the-art VMs such as V8 now deprecate their usage in favor of real applications [2].

Besides the programs used to evaluate language implementations, other issues arise from the used methodologies : their **reproducibility** and **significance**. Indeed, benchmarking results suffer from non-determinisms from hardware, operating systems, and multi-threaded environments. Georges et al. [28] proposed in 2007 a statistically rigorous benchmarking methodology that has been in use for the last decade and a half. Recently in 2017 [8] the VM community discovered great flaws in such methodology and its derivates based on the wrong assumption that application performance stabilizes in a steady state.

Finally, the other side of the benchmarking coin is the runtime **profile** and **monitoring**. While benchmarking presents the execution results as a snapshot or a summary statistic, better understanding the performance behavior of language implementations requires tools to observe their execution *on-line*. Most profiling tools focus on application performance and provide little-to-none insights on language implementation behavior and its impact on performance [78, 39, 53, 70]. Recent work proposes to expose the internals of Virtual Machines to better understand the inter-relations of their different components [87], but mostly focus on their JIT compilers [41, 93].

It is worth noticing that these challenges have been explored in the past for speed reasons. It is not until recently that work started to evaluate also VM performance in terms of the consumption of energy [65] or memory [4, 72].

4 Challenges in Memory Management

Automated memory management has been a topic of increasing interest and study since McCarthy's seminal work in the 60s [56]. Whether automatic or manual, the agreement is that memory management is crucial to application performance due to its intricate relationship to hardware concerns, programming languages, application development patterns, and new data-centric applications. Yet, understanding such relationships remains an unresolved issue. The research community has recently barely been able to propose approximations of the cost of different memory management strategies [13] and shown that modern system performance is dominated by cache misses [34].

Besides performance, memory management is still nowadays one of the biggest causes of bugs and vulnerabilities due to memory unsafety and corruption [49]. Data-oriented programming attacks [32] do not only produce leaks of sensible data but also break program control flow integrity [3] allowing attacking disciplines such as the so-called return-oriented programming [81].

Moreover, memory management research needs to adapt to the new hardware technologies that appear at a great speed, such as disaggregated memory [10], GPUs [47], processor in memory [24], compressed memory [84], non-volatile memory [48] and hybrid solutions [83]. It is still an open question whether the current programming language abstractions adapt correctly to such technologies [55] or not.

5 Challenges in VM Software Engineering

Virtual Machines are complex pieces of engineering, used ubiquitously for personal, professional, and academic purposes. This ubiquity has raised the stakes in delivering quality implementations that achieve good speed and strong security, and are able to quickly evolve and adapt to a changing world.

Recent work acknowledges that Virtual Machines **construction incurs a high cost** in practice [95] because of the complexity and inter-dependence of its many components [54]. It is indeed a case where the whole is more than the sum of its parts. Several work address VM construction with modular approaches [27, 40], some targetting embedded systems [85] or dynamic reconfigurations [88, 51]. Recently it has been proposed to re-use existing compilers in different scenarios to build multi-tier execution engines with minimal effort [38, 37]. Wimmer et al. argue that some components can be automatically generated up to some degree *e.g.,* a compiler from an interpreter [12, 99], an interpreter from a compiler [86]. However, these specifications are partial and the generated artifacts present for example degradations in memory management in some cases [61]. Moreover, the underlying technology lies in some cases under the control of private companies (e.g., Oracle Graal).

A hidden cost in VM construction is the **verification** of their functional properties (correctness) and non-functional properties (speed, security). VM testing tasks have been traditionally approached with simulation environments [92, 36, 57, 74] and Metacircular VMs [96]. However, reproducing bugs still remains an expensive and time-consuming task because millions of instructions may need to be executed before hitting the actual problem. Several works have explored the path of reducing the cost of testing through program fuzzing [19, 18, 102, 68], test generation [35], and test transplantation [50]. These solutions are subject to VM nondeterminisms and require the availability of several execution engines as test oracles. Lately, interpreter-guided JIT compiler unit testing has shown that we can leverage the multiple execution engines inside a single VM for both test generation and testing oracle. However, it is still open how such an approach could be applied to coarse-grained integration tests. Similar techniques have been proposed also to perform automatic performance evaluations, either by the means of leveraging existing application test cases [23] or by doing microbenchmark generation [76].

6 Challenges in Security

VM complexity makes the VM runtime an interesting target for attackers, by either exploiting memory or JIT compiler vulnerabilities. The Chromium Project found out that about 70% of their identified security bugs were caused by memory safety violations [1]. Vulnerabilities in JIT compilers lead to code injection [16] or JIT spraying attacks [17], often needing no more than a carefully written input program and knowledge of the JIT compiler behavior.

Such security concerns raised recently awareness in the research community, which needs to respond with solutions that increase the security of language runtimes without sacrificing their performance benefits. Ahead of time verification of virtual machines has been explored for interpreters, sacrificing JIT compilation capabilities altogether with their speculative optimizations [21]. Software isolation-based solutions make a primary focus on fault-isolation [30, 6, 66, 33, 52], but largely restrain developers in the proposed programming model. Low-cost hardware-based enforcement of fine-grained memory isolation has been an important research focus as a countermeasure to the most advanced JIT attacks [62, 26, 69]. It comes in different forms, from memory protection keys to hardware-enforced environments. Memory isolation splits the components of an application with controlled communication and verified access to other resources. Some works based on the approach developed in [44] propose to extend the instruction set in order to counter previously cited JIT-based attacks.

7 Early Achievements

7.1 RMOD

Action Exploratoire AlaMVic. Since February 2021. Lead by G. Polito lead. AlaMVic explores new methods for Virtual Machines construction to solve the *high-cost* of VM construction, joining together programming language implementation, design, and engineering, using a holistic generative approach.

Interpreter-guided Differential JIT Compiler Unit Testing @ PLDI'22 . Novel automated testing approach for virtual machines combining concolic meta-interpretation and differential testing of interpreters and JIT compilers.

Interpreter Register Autolocalisation @ MoreVMs'22 . Automatic transformation to improve interpreter performance without sacrificing code quality.

Ahead-of-time JIT Compiler Generation - In progress . RQ : Can we generate baseline JIT compilers ahead-of-time using meta-interpretation techniques ?

JIT Compiler Static Code Reordering . RQ : Is code reordering worth it in a heuristic static setting without profiling information ? Alternative to Pettis-Hansen basic block reordering.

7.2 Benagil

J-NVM @ SOSP'21. Efficient integration of persistent memory in a Java Virtual Machine.

PrivaDSL - In progress. Use of hardware-enforced isolation such as Intel SGX in a Java virtual machine.

VM Disaggregated Memory - In progress. Study of a Java virtual machine for disaggregated memory.

Language Support for Serverless Applications - In progress. A shell language and runtime for serverless applications.

7.3 ENSTA Bretagne

Porting a JIT compiler to RISC-V : Challenges and Opportunities @ MPLR'22 . First works about Pharo port on RISC-V architecture. Works with RMOD colleagues.

Protections against JIT Attacks - In progress. How JIT compilers can be secured using an extended ISA ?

RISC-V cores - In progress . Developing RISC-V modules to study custom Instruction Set Architecture (ISA) extensions for JIT protection and other security related issues.

Références

[1] The chromium project. memory safety. Retrieved June 07 2022.

[2] Retiring octane. Retrieved June 07 2022.

[3] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.*, 13(1), nov 2009.

[4] I. Agadakos, D. Jin, D. Williams-King, V. P. Kemerlis, and G. Portokalidis. Nibbler : Debloating binary shared libraries. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 70–83, New York, NY, USA, 2019. Association for Computing Machinery.

[5] M. Arnold, S. J. Fink, D. Grove, M. Hind, and P. F. Sweeney. A survey of adaptive optimization in virtual machines. *Proceedings of the IEEE*, 93(2) :449–466, 2005.

[6] G. Back, W. Hsieh, and J. Lepreau. Processes in kaffeos : Isolation, resource management and sharing in java. In *4th USENIX International Symposium on Operating System Design and Implementation (OSDI)*, 2000.

[7] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo : A transparent dynamic optimization system. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, PLDI '00, pages 1–12, New York, NY, USA, 2000. Association for Computing Machinery.

[8] E. Barrett, C. F. Bolz-Tereick, R. Killick, V. Knight, S. Mount, and L. Tratt. Virtual machine warmup blows hot and cold. In *Proceedings of International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'17)*. ACM, Oct. 2017.

[9] C. Béra, E. Miranda, T. Felgentreff, M. Denker, and S. Ducasse. Sista : Saving optimized code in snapshots for fast start-up. In *Proceedings of the 14th International Conference on Managed Languages and Runtimes*, pages 1 – 11, Prague, Czech Republic, Sept. 2017. ACM.

[10] S. Bergman, P. Faldu, B. Grot, L. Vilanova, and M. Silberstein. Reconsidering os memory optimizations in the presence of disaggregated memory. In M. Lippautz and D. Chisnall, editors, *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management*, pages 1–14. ACM, June 2022. 2022 ACM SIGPLAN International Symposium on Memory Management, ISMMM 2022 ; Conference date : 14-06-2022 Through 14-06-2022.

[11] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The dacapo benchmarks : Java benchmarking development and analysis. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 169–190, New York, NY, USA, 2006. Association for Computing Machinery.

[12] C. F. Bolz, A. Cuni, M. Fijalkowski, and A. Rigo. Tracing the meta-level : Pypy's tracing jit compiler. In *ICOOOLPS '09 : Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pages 18–25, New York, NY, USA, 2009. ACM.

[13] Z. Cai, S. M. Blackburn, M. D. Bond, and M. Maas. Distilling the real cost of production garbage collectors. *CoRR*, abs/2112.07880, 2021.

[14] S. Cazzulani. Octane : The javascript benchmark suite for the modern web. Retrieved June 07 2022.

[15] G. Chari, J. Pimás, J. Vitek, and O. Flückiger. Self-contained development environments. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Dynamic Languages*, DLS 2018, page 76–87, New York, NY, USA, 2018. Association for Computing Machinery.

[16] P. Chen, Y. Fang, B. Mao, and L. Xie. Jitdefender : A defense against jit spraying attacks. In *IFIP International Information Security Conference*, pages 142–153. Springer, 2011.

[17] P. Chen, R. Wu, and B. Mao. Jitsafe : a framework against just-in-time spraying attacks. *IET Information Security*, 7(4) :283–292, 2013.

[18] Y. Chen, T. Su, and Z. Su. Deep differential testing of jvm implementations. In *International Conference on Software Engineering (ICSE'19)*, pages 1257–1268. IEEE Press, 2019.

[19] Y. Chen, T. Su, C. Sun, Z. Su, and J. Zhao. Coverage-directed differential testing of JVM implementations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 85–99. Association for Computing Machinery, June 2016.

[20] M. Chevalier-Boisvert and M. Feeley. Interprocedural type specialization of javascript programs without type analysis. In *European Conference on Object-Oriented Programming (ECOOP'16)*, pages 1–24, 2016.

[21] M. Desharnais and S. Brunthaler. Towards efficient and verified virtual machines for dynamic languages. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2021, page 61–75, New York, NY, USA, 2021. Association for Computing Machinery.

[22] L. P. Deutsch and A. M. Schiffman. Efficient implementation of the Smalltalk-80 system. In *Proceedings POPL '84*, Salt Lake City, Utah, Jan. 1984.

[23] Z. Ding, J. Chen, and W. Shang. Towards the use of the readily available tests from the release pipeline as performance tests : Are we there yet ? In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1435–1446, New York, NY, USA, 2020. Association for Computing Machinery.

[24] M. E. Fouda, H. E. Yantır, A. M. Eltawil, and F. Kurdahi. In-memory associative processors : Tutorial, potential, and challenges. *IEEE Transactions on Circuits and Systems II : Express Briefs*, 69(6) :2641–2647, 2022.

[25] O. I. Foundation. Openstack users in production, 2020.

[26] T. Frassetto, D. Gens, C. Liebchen, and A.-R. Sadeghi. Jitguard : hardening just-in-time compilers with sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2405–2419, 2017.

[27] N. Geoffray, G. Thomas, J. Lawall, G. Muller, and B. Folliot. Vmkit : a substrate for managed runtime environments. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '10, pages 51–62, New York, NY, USA, 2010. ACM.

[28] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA '07, pages 57–76, New York, NY, USA, 2007. Association for Computing Machinery.

[29] M. Golm, M. Felser, C. Wawersich, and J. Kleinöder. The jx operating system. In *ATEC '02 : Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, pages 45–58, Berkeley, CA, USA, 2002. USENIX Association.

[30] I. Gonzalez-Herrera, J. Bourcier, E. Daubert, W. Rudametkin, O. Barais, F. Fouquet, J. Jézéquel, and B. Baudry. Scapegoat : Spotting abnormal resource usage in component-based reconfigurable software systems. *Journal of Systems and Software*, 122 :398–415, 2016.

[31] U. Holzle. *Adaptive Optimization for Self : Reconciling High Performance with Exploratory Programming*. PhD thesis, Stanford University, Stanford, CA, USA, 1994.

[32] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang. Data-oriented programming : On the expressiveness of non-control data attacks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 969–986, 2016.

[33] G. Hunt, M. Aiken, M. Fähndrich, C. Hawblitzel, O. Hodson, James, S. Levi, B. Steensgaard, D. Tarditi, and T. Wobber. Sealing os processes to improve dependability and safety. In *In Proceedings of the ACM EuroSys Conference*, pages 341–354, 2007.

[34] A. Hunter, C. Kennelly, P. Turner, D. Gove, T. Moseley, and P. Ranganathan. Beyond malloc efficiency to fleet efficiency : a hugepage-aware memory allocator. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 257–273. USENIX Association, July 2021.

[35] S. Hwang, S. Lee, J. Kim, and S. Ryu. Justgen : Effective test generation for unspecified jni behaviors on jvms. In *International Conference on Software Engineering (ICSE'21)*, pages 1708–1718, 2021.

[36] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay. Back to the future : The story of Squeak, a practical Smalltalk written in itself. In *Proceedings of Object-Oriented Programming, Systems, Languages, and Applications conference (OOPSLA'97)*, pages 318–326. ACM Press, Nov. 1997.

[37] Y. Izawa and H. Masuhara. Amalgamating different JIT compilations in a meta-tracing JIT compiler framework. *CoRR*, abs/2011.03516, 2020.

[38] Y. Izawa, H. Masuhara, and C. F. Bolz-Tereick. Two-level just-in-time compilation with one interpreter and one engine. *CoRR*, abs/2201.09268, 2022.

[39] Visualvm : All-in-one java troubleshooting tool. `https://visualvm.github.io/`.

[40] The Jikes research virtual machine. http ://jikesrvm.sourceforge.net/.

[41] S. Kaleba, C. Béra, A. Bergel, and S. Ducasse. A detailed vm profiler for the cog vm. In *International Workshop on Smalltalk Technology IWST'17*, Maribor, Slovenia, Sept. 2017.

[42] S. Kaleba, O. Larose, S. Marr, and R. Jones. Who you gonna call ? a case study about the call-site behaviour in ruby-on-rails applications. mar 2022.

[43] K. Kawachiya, K. Ogata, D. Silva, T. Onodera, H. Komatsu, and T. Nakatani. Cloneable jvm : A new approach to start isolated java applications faster. In *International Conference on Virtual Execution Environments*, VEE '07, pages 1–11, New York, NY, USA, 2007. ACM.

[44] H. Kim, J. Lee, D. Pratama, A. M. Awaludin, H. Kim, and D. Kwon. Rimi : instruction-level memory isolation for embedded systems on risc-v. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[45] O. Larose, S. Kaleba, and S. Marr. Less is more : Merging ast nodes to optimize interpreters. February 2022.

[46] F. Latifi, D. Leopoldseder, C. Wimmer, and H. Mössenböck. Compgen : Generation of fast jit compilers in a multi-language vm. In *Proceedings of the 17th ACM SIGPLAN International Symposium on Dynamic Languages*, DLS 2021, page 35–47, New York, NY, USA, 2021. Association for Computing Machinery.

[47] T. D. Le, H. Imai, Y. Negishi, and K. Kawachiya. Automatic gpu memory management for large neural models in tensorflow. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on Memory Management*, ISMM 2019, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[48] A. Lefort, Y. Pipereau, K. Amponsem, P. Sutra, and G. Thomas. J-nvm : Off-heap persistent objects in java. In *Proceedings of the Symposium on Operating Systems Principles, SOSP'21*, page 16. ACM, 2021.

[49] H. Liljestrand, T. Nyman, K. Wang, C. C. Perez, J.-E. Ekberg, and N. Asokan. PAC it up : Towards pointer integrity using ARM pointer authentication. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 177–194, Santa Clara, CA, Aug. 2019. USENIX Association.

[50] I. Lima, J. Silva, B. Miranda, G. Pinto, and M. d'Amorim. Exposing Bugs in JavaScript Engines through Test Transplantation and Differential Testing. *arXiv :2012.03759 [cs]*, 2020.

[51] Y. Lin, S. M. Blackburn, and D. Frampton. Unpicking the knot : Teasing apart vm/application interdependencies. *SIGPLAN Not.*, 47(7) :181–190, Mar. 2012.

[52] D. Lohmann, J. Streicher, W. Hofer, O. Spinczyk, and W. Schröder-Preikschat. Configurable memory protection by aspects. In *Proceedings of the 4th Workshop on Programming Languages and Operating Systems*, PLOS '07, New York, NY, USA, 2007. Association for Computing Machinery.

[53] L. Marek, S. Kell, Y. Zheng, L. Bulej, W. Binder, P. Tŭma, D. Ansaloni, A. Sarimbekov, and A. Sewe. Shadowvm : Robust and comprehensive dynamic program analysis for the java platform. *ACM SIGPLAN Notices*, 49(3) :105–114, 2013.

[54] S. Marr. Modularisierung virtueller maschinen. Master's thesis, Hasso Plattner Institute, Germany, 2008.

[55] K. Matsumoto, T. Ugawa, and H. Iwasaki. Replication-based object persistence by reachability. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management*, ISMM 2022, page 43–56, New York, NY, USA, 2022. Association for Computing Machinery.

[56] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *CACM*, 3(4) :184–195, Apr. 1960.

[57] E. Miranda, C. Béra, E. G. Boix, and D. Ingalls. Two decades of smalltalk vm development : live vm development through simulation tools. In *Proceedings of International Workshop on Virtual Machines and Intermediate Languages (VMIL'18)*, pages 57–66. ACM, 2018.

[58] R. Mosaner, D. Leopoldseder, W. Kisling, L. Stadler, and H. Mössenböck. Compilation forking : A fast and flexible way of generating data for compiler-internal machine learning tasks. *The Art, Science, and Engineering of Programming*, 7(1), jun 2022.

[59] R. Mosaner, D. Leopoldseder, L. Stadler, and H. Mössenböck. Using machine learning to predict the code size impact of duplication heuristics in a dynamic compiler. In *Proceedings of the 18th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, MPLR 2021, page 127–135, New York, NY, USA, 2021. Association for Computing Machinery.

[60] .net common language runtime (clr) overview. `https://docs.microsoft.com/en-us/dotnet/standard/clr`.

[61] F. Niephaus, T. Felgentreff, and R. Hirschfeld. Graalsqueak : Toward a smalltalk-based tooling platform for polyglot programming. In *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes (MPLR'19)*, pages 14–26, New York, NY, USA, 2019. ACM.

[62] B. Niu and G. Tan. Rockjit : Securing just-in-time compilation using modular control-flow integrity. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1317–1328, 2014.

[63] T. Ogasawara. Workload characterization of server-side javascript. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 13–21, 2014.

[64] U. Okafor, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi. A methodology to transform an os-based application to a bare machine application. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1428–1433, 2013.

[65] Z. Ournani, M. C. Belgaid, R. Rouvoy, P. Rust, and J. Penhoat. Evaluating the Impact of Java Virtual Machines on Energy Consumption. In *15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bari, Italy, Oct. 2021.

[66] K. Palacz, G. Czajkowski, and L. Es. Incommunicado : Efficient communication for isolates. *none*, Apr. 2004.

[67] M. Paleczny, C. Vick, and C. Click. The java hotspotTM server compiler. In *Symposium on JavaTM Virtual Machine Research (JVM'01)*, JVM'01. USENIX Association, 2001.

[68] J. Park, S. An, D. Youn, G. Kim, and S. Ryu. Jest : N+1-version differential testing of both javascript engines and specification. In *International Conference on Software Engineering (ICSE'21)*, pages 13–24. IEEE Press, 2021.

[69] T. Park, K. Dhondt, D. Gens, Y. Na, S. Volckaert, and M. Franz. Nojitsu : Locking down javascript engines. In *NDSS*, 2020.

[70] X. Peng, B. Pernici, and M. Vitali. Virtual machine profiling for analyzing resource usage of applications. In J. E. Ferreira, G. Spanoudakis, Y. Ma, and L.-J. Zhang, editors, *Services Computing – SCC 2018*, pages 103–118, Cham, 2018. Springer International Publishing.

[71] F. Pizlo. Jetstream benchmark suite. Retrieved June 07 2022.

[72] G. Polito, L. Fabresse, N. Bouraqadi, and S. Ducasse. Run-fail-grow : Creating tailored object-oriented runtimes. *The Journal of Object Technology*, 16(3) :2 :1–36, 2017.

[73] G. Polito, N. Palumbo, P. Tesone, and S. Ducasse. Interpreter register autolocalisation : Improving the performance of efficient interpreters. In *Workshop on Modern Language Runtimes, Ecosystems, and VMs*, MoreVMs'22, June 2022.

[74] G. Polito, P. Tesone, S. Ducasse, L. Fabresse, T. Rogliano, P. Misse-Chanabier, and C. H. Phillips. Cross-ISA Testing of the Pharo VM : Lessons Learned While Porting to ARMv8. In *Proceedings of the 18th international conference on Managed Programming Languages and Runtimes (MPLR '21)*, Münster, Germany, Sept. 2021.

[75] A. Rigo and S. Pedroni. PyPy's approach to virtual machine construction. In *Proceedings of the 2006 conference on Dynamic languages symposium*, pages 944–953, New York, NY, USA, 2006. ACM.

[76] M. Rodriguez-Cancio, B. Combemale, and B. Baudry. Automatic Microbenchmark Generation to Prevent Dead Code Elimination and Constant Folding. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)* , ASE 2016 :Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, Singapore, Sept. 2016.

[77] E. Rohou, B. Narasimha Swamy, and A. Seznec. Branch Prediction and the Performance of Interpreters - Don't Trust Folklore. In *International Symposium on Code Generation and Optimization*, Burlingame, United States, Feb. 2015.

[78] E. Rosales, A. Rosà, and W. Binder. Fjprof : Profiling fork/join applications on the java virtual machine. In *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '20, pages 128–135, New York, NY, USA, 2020. Association for Computing Machinery.

[79] A. Sarimbekov, L. Stadler, L. Bulej, A. Sewe, A. Podzimek, Y. Zheng, and W. Binder. Workload characterization of jvm languages. *Software : Practice and Experience*, 46(8) :1053–1089, 2016.

[80] A. Sewe, M. Mezini, A. Sarimbekov, and W. Binder. Da capo con scala : Design and analysis of a scala benchmark suite for the java virtual machine. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, page 657–676, New York, NY, USA, 2011. Association for Computing Machinery.

[81] H. Shacham. The geometry of innocent flesh on the bone : Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 552–561, New York, NY, USA, 2007. Association for Computing Machinery.

[82] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White. Java on the bare metal of wireless sensor devices : the Squawk Java virtual machine. In *VEE '06 : Proceedings of the 2nd international conference on Virtual execution environments*, pages 78–88, New York, NY, USA, 2006. ACM Press.

[83] S. Song, A. Das, and N. Kandasamy. Exploiting inter- and intra-memory asymmetries for data mapping in hybrid tiered-memories. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*, ISMM 2020, page 100–114, New York, NY, USA, 2020. Association for Computing Machinery.

[84] T. Song, M. Kim, G. Lee, and Y. Kim. Prediction-guided performance improvement on compressed memory swap. In *2022 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, 2022.

[85] M. Stilkerich, I. Thomm, C. Wawersich, and W. Schröder-Preikschat. Tailor-made jvms for statically configured embedded systems. *Concurrency and Computation : Practice and Experience*, 24(8) :789–812, 2012.

[86] M. Stoodley. In pursuit of easy(er) jits (invited talk). In *Proceedings of the 12th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*, VMIL 2020, page 2, New York, NY, USA, 2020. Association for Computing Machinery.

[87] P. Tesone, G. Polito, and S. Ducasse. Profiling Code Cache Behaviour via Events. In *MPLR '21*, Münster, Germany, Sept. 2021.

[88] G. Thomas, N. Geoffray, C. Clément, and B. Folliot. Designing highly flexible virtual machines : The jnjvm experience. *Softw. Pract. Exper.*, 38(15) :1643–1675, dec 2008.

[89] C.-J. Tsai, C.-J. Lin, C.-Y. Chen, Y.-H. Lin, W.-J. Ji, and S.-D. Hong. Hardwiring the os kernel into a java application processor. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 53–60, 2017.

[90] D. Ungar. Generation scavenging : A non-disruptive high performance storage reclamation algorithm. *ACM SIGPLAN Notices*, 19(5) :157–167, 1984.

[91] D. Ungar and F. Jackson. Tenuring policies for generation-based storage reclamation. In *Proceedings OOPSLA '88*, volume 23, pages 1–17, Nov. 1988.

[92] D. Ungar, A. Spitz, and A. Ausch. Constructing a metacircular virtual machine in an exploratory programming environment. In *Companion to Object-Oriented Programming, Systems, Languages, and Applications conference (OOPSLA'05)*, pages 11–20, New York, NY, USA, 2005. ACM.

[93] Indicium : V8 runtime tracer tool. `https://v8.dev/blog/system-analyzer`.

[94] V8 : Google's open source high-performance javascript and webassembly engine. `https://v8.dev`.

[95] C. Wimmer, S. Brunthaler, P. Larsen, and M. Franz. Fine-grained modularity and reuse of virtual machine components. In *Proceedings of the 11th Annual International Conference on Aspect-Oriented Software Development*, AOSD '12, pages 203–214, New York, NY, USA, 2012. Association for Computing Machinery.

[96] C. Wimmer, M. Haupt, M. L. Van De Vanter, M. Jordan, L. Daynès, and D. Simon. Maxine : An approachable virtual machine for, and in, java. *ACM Transaction Architecture Code Optimization*, 9(4), Jan. 2013.

[97] C. Wimmer, C. Stancu, P. Hofer, V. Jovanovic, P. Wögerer, P. B. Kessler, O. Pliss, and T. Würthinger. Initialize once, start fast : application initialization at build time. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA) :1–29, 2019.

[98] C. Wimmer and T. Würthinger. Truffle : a self-optimizing runtime system. In *Proceedings of the 3rd annual conference on Systems, programming, and applications : software for humanity*, pages 13–14, 2012.

[99] T. Würthinger, C. Wimmer, C. Humer, A. Wöss, L. Stadler, C. Seaton, G. Duboscq, D. Simon, and M. Grimmer. Practical partial evaluation for high-performance dynamic language runtimes. In *PLDI'17*, 2017.

[100] T. Würthinger, A. Wöß, L. Stadler, G. Duboscq, D. Simon, and C. Wimmer. Self-optimizing ast interpreters. In *Symposium on Dynamic Languages*, DLS '12, 2012.

[101] X. Xu, K. Cooper, J. Brock, Y. Zhang, and H. Ye. Sharejit : Jit code cache sharing across processes and its practical implementation. *Proc. ACM Program. Lang.*, 2(OOPSLA), oct 2018.

[102] G. Ye, Z. Tang, S. H. Tan, S. Huang, D. Fang, X. Sun, L. Bian, H. Wang, and Z. Wang. Automated conformance testing for javascript engines via deep compiler fuzzing. In *Proceedings of Conference on Programming Language Design and Implementation (PLDI'21)*, pages 435–450, New York, NY, USA, 2021. ACM.