A Lightweight Method for Generating Multi-Tier JIT Compilation Virtual Machine in a Meta-Tracing Compiler Framework (Artifact)

Yusuke Izawa
Tokyo Metropolitan University, Japan

Hidehiko Masuhara
Institute of Science Tokyo, Japan

Carl Friedrich Bolz-Tereick
Heinrich-Heine-Universität Düsseldorf, Germany

Abstract

Meta-compiler frameworks, such as RPython and Graal/Truffle, generate high-performance virtual machines (VMs) from interpreter definitions. Although they generate VMs with high-quality just-in-time (JIT) compilers, they still lack an important feature that dedicated VMs (i.e., VMs that are developed for specific languages) have, namely *multi-tier compilation*. Multi-tier compilation uses light-weight compilers at early stages and highly optimizing compilers at later stages in order to balance between compilation overheads and code quality.

We propose a novel approach to enabling multi-tier compilation in the VMs generated by a meta-compiler framework. Instead of extending the JIT compiler backend of the framework, our approach drives an existing (heavyweight) compiler backend in the framework to quickly generate unoptimized native code by merely embedding directives and compile-time operations into interpreter definitions.

As a validation of the approach, we developed 2SOM, a Simple Object Machine with a two-tier JIT compiler based on RPython. 2SOM first applies the tier-1 threaded code generator that is generated by our proposed technique, then, to the loops that exceed a threshold, applies the tier-2 tracing JIT compiler that is generated by the original RPython framework. Our performance evaluation that runs a program with a realistic workload showed that 2SOM improved, when compared against an RPython-based VM, warm-up performance by 15%, with merely a 5% reduction in peak performance.

2012 ACM Subject Classification Software and its engineering → Just-in-time compilers; Software and its engineering → Interpreters

Keywords and phrases virtual machine, JIT compiler, multi-tier JIT compiler, meta-tracing JIT compiler, RPython

Digital Object Identifier 10.4230/DARTS.11.2.16

Funding This research was supported by KAKENHI grant nos. 25K21180, 24K23866, 23H03368, and 23K28058, and JST ACT-X grant no. JPMJAX2003.

Acknowledgements We would like to thank Stefan Marr for his valuable advice on the measurement methodology and the evaluation results of 2SOM.

Related Article Yusuke Izawa, Hidehiko Masuhara, and Carl Friedrich Bolz-Tereick, "A Lightweight Method for Generating Multi-Tier JIT Compilation Virtual Machine in a Meta-Tracing Compiler Framework", in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 16:1–16:29, 2025. `https://doi.org/10.4230/LIPIcs.ECOOP.2025.16`

Related Conference 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact provides the source code of 2SOM [4], a two-level Simple Object Machine [1], and its dependencies including a customized version of PyPy [2] and RPython extension for `monotonic_clock` [3]. In addition, the benchmark set used in the paper, TruffleSOM and its associated GraalVM community edition, are included to reproduce result data shown in the paper. Also, a benchmarking tool called ReBench [5] is included to measure and collect data.

2 Content

- type of artifact: code
- format: source code, and binary
- location in the container: `https://doi.org/10.5281/zenodo.15302566`

The artifact package includes:

- (under artifact/2SOM/) the source code of 2SOM.
- (under artifact/2SOM/Examples/Benchmarks/) benchmark sets used in this artifact for 2SOM.
- (under artifact/2SOM/Smalltalk/) the standard library of SOM.
- (under artifact/pypy) the source code of our customized version of PyPy for enabling threaded code generation.
- (under artifact/pyp2) the release version of PyPy 7.3.19.
- (under artifact/pypy-no-handler-opt) the source code of our customized version of PyPy, but disabled the handler optimization just for evaluation.
- (under artifact/2SOM/site-packages/rtime_ext/) RPython extension for using `clock_monotonic` in 2SOM.
- (under artifact/TruffleSOM/) the source code of TruffleSOM, which is used for reproducing the data used in Figure 9.
- (under artifact/TruffleSOM/Smalltalk) the standard library of SOM used in TruffleSOM.
- (under artifact/TruffleSOM/Examples/Benchmarks) benchmark sets used for TruffleSOM. They are the same as those used in 2SOM.
- (under artifact/mx) a build tool for GraalVM-hosted languages, including TruffleSOM.
- (under artifact/labs-openjdk) a directory that contains the JDK fork for building GraalVM CE.
- (under artifact/graal) the source code of the latest GraalVM.
- (under artifact/build.sh) a script to build binaries in this artifact.
- (under artifact/run.sh) a script to run evaluations.
- (under artifact/run_quick.sh) a script to run evaluations for the kick-the-tier phase.
- (under artifact/visualize.sh) a script to generate plots.

The rest of them are artifacts needed for collecting data and reproducing resulting data in the Appendix in the extended version of the associated paper:

- (under artifact/run_appendix.sh) a script to collect data.
- (under artifact/visualize_appendix.sh) a script to generate plots.
- (under artifact/pypybenchmarks) benchmark sets used for collecting data in the Appendix.
- (under artifact/appendix) directory that contains scripts for running evaluations and generating plots in the Appendix.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact can be downloaded from Zenodo: `https://doi.org/10.5281/zenodo.15302566`.

3.1 Setup

1. Download and install Docker.
2. Download `image.tar` from Zenodo.

3.2 Launch the Docker Container from the Terminal

1. Change your directory to /Downloads
2. Load the image by the following commands, depending on your system:
   - `docker load -i "image.tar"`
3. Verify that the image was successfully loaded.
   - `docker images`
4. Run the image in a container by the following command:
   - `docker run --name ecoop25-8 -it ecoop25-8-amd64:1.0 /bin/bash`
5. Now, you are inside the container.

3.3 Build binaries and Run the Experiments

1. Ensure that you're in the directory `/work`
2. Run the experiments:
   - `./run.sh`
     - This script may take at least 1 week, depending on the testing platform
     - For quick evaluation, you can run `./run_quick.sh`
3. Generate plots from results:
   - `./visualize.sh`
4. (Optional:)
   - Run evaluations and generate plots in the Appendix in the full version:
     - `./run_appendix.sh`
     - `./visualize_appendix.sh`

3.4 Correlation between Generated Plots and Figures

Generated plot	Figure
`rq1.pdf`	Figure 9
`rq1_num_trace_ops.pdf`	Figure 10
`rq2_norm.pdf`	Figure 11
`rq2_micro_norm.pdf`	Figure 12
`rq3.pdf`	Figure 13
`appendix/exp_rank.pdf`	Figure 15
`appendix/method_count_rank_selected.pdf`	Figure 16
`appendix/pypylog_method_number_trace.pdf`	Figure 17

- You can check the resulting plots in your local machine by `docker cp` command:
  - `docker cp ecoop25-8:/work/rq1.pdf .`
  - `docker cp ecoop25-8:/work/rq1.pdf .`
  - `docker cp ecoop25-8:/work/rq1_num_trace_ops.pdf .`
  - `docker cp ecoop25-8:/work/rq2_norm.pdf .`
  - `docker cp ecoop25-8:/rq2_micro_norm.pdf .`
  - `docker cp ecoop25-8:rq3.pdf .`
  - `docker cp ecoop25-8:/appendix/exp_rank.pdf .`
  - `docker cp ecoop25-8:/work/appendix/method_count_rank_selected.pdf .`
  - `docker cp ecoop25-8:/appendix/pypylog_method_number_trace.pdf .`

4 Tested platforms

We have tested the execution of 2SOM, TruffleSOM, and PyPy on the platform with following specifications:

- OS: Ubuntu 24.04 LTD with kernel 6.8.0-57-generic
- CPU: Intel(R) Core(TM) i7-6700
  - other x86-64 supported CPU, such as Ryzen, can be used, but the resulting data may vary more due to the inability to use the denoising mechanism provided by Rebench.
  - ARM-based CPU such as Apple M1 is not checked to run the artifact. Since the provided image is built on a x86-64-based CPU, please run `./build.sh` to create ARM-based binaries.
- Memory: DDR4 2133 MHz Memory

5 License

The artifact is available under Creative Commons license.

6 MD5 sum of the artifact

0e59c23365e19dfc9c2c672de8c2b157

7 Size of the artifact

11.2 GiB

References

1 Michael Haupt, Robert Hirschfeld, Tobias Pape, Gregor Gabrysiak, Stefan Marr, Arne Bergmann, Arvid Heise, Matthias Kleine, and Robert Krahn. The som family: Virtual machines for teaching and research. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2010, pages 18–22. Association for Computing Machinery, 2010. `doi:10.1145/1822090.1822098`.

2 Yusuke Izawa. Customized PyPy for Threaded Code Generation, April 2025. Version 1.0. `doi:10.5281/zenodo.15287001`.

3 Yusuke Izawa. RPython Extension that Enables a Monotonic Clock, April 2025. Version 1.0. `doi:10.5281/zenodo.15286954`.

4 Yusuke Izawa, Hidehiko Masuhara, and Carl Friedrich Bolz-Tereick. A Lightweight Method for Generating Multi-Tier JIT Compilation Virtual Machine in a Meta-Tracing JIT Compiler Framework (Extended Version). arXiv, 2025. `doi:10.48550/arXiv.2504.17460`.

5 Stefan Marr. ReBench: Execute and Document Benchmarks Reproducibly, August 2018. Version 1.0. `doi:10.5281/zenodo.1311762`.