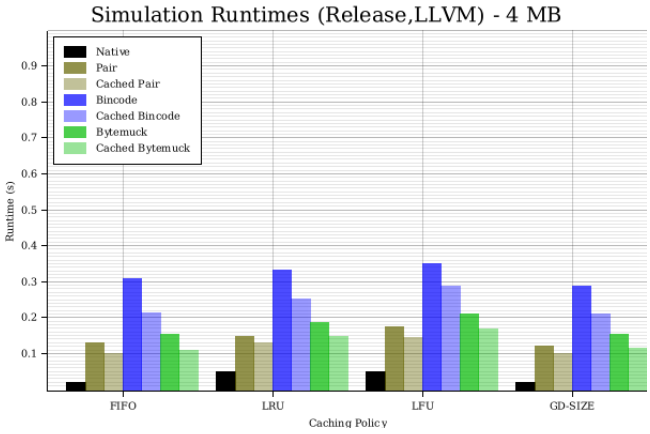

Figure 7: Execution-Time Performance of Simulator

the Wasmer references helps in all cases, so it should be done whenever possible. Among the three ABIs tested, they all have potential use cases. Pair is the fastest, but only works for data that can be represented as a fixed-length tuple of integers or floats. Bytemuck with static memory allocation is the next fastest for passing arbitrary data, however, this method might face challenges in the case of complex pointers or references within the data. Finally, Bincode is the slowest, but if it is possible to remove the extra call for dynamic memory allocation, it could be as fast as Bytemuck, while offering many more features.

When comparing the JIT backends, Singlepass is inferior unless extremely fast JIT times are desired. If the performance of the code generated by the JIT compiler is important, then the Cranelift or LLVM compilers are preferred, with the LLVM making a small performance gain in some programs, at the cost of a much slower JIT process than Cranelift, in addition to a more complex compilation process for the host program.

There are several potential directions for future work. These include exploring the differences between Bincode and Bytemuck on more complex data types. Additionally, these methods could be tested and verified using other languages that support WebAssembly, such as C, C++ and C#. Another avenue is investigating the performance of these methods with variable-length data, and batching the data to reduce the number of calls in the sandbox. Finally, there are forthcoming features of WASM called WebAssembly Interface Types, and WebAssembly Reference Types. Once available, their performance should be compared to these other ABI techniques, to see if a built-in solution in the runtime is better.

ACKNOWLEDGEMENTS

The authors thank the LTB 2022 reviewers for their constructive suggestions that helped to improve our paper. Financial support for this work was provided in part by Canada's Natural Sciences and Engineering Research Council (NSERC).

REFERENCES

[1] Bytecode Alliance. [n.d.]. Cranelift. https://github.com/bytecodealliance/wasmtime/tree/main/cranelift
[2] Bytecode Alliance. [n.d.]. wasmtime: Standalone JIT-style runtime for WebAssembly, using Cranelift. https://github.com/bytecodealliance/wasmtime
[3] Mudashiru Busari and Carey Williamson. 2002. ProWGen: A Synthetic Workload Generation Tool for the Simulation Evaluation of Web Proxy Caches. *Computer Networks* 38, 6 (June 2002), 779–794.
[4] Xiao-hui Cheng and Liang Zhang. 2011. "A Research of inter-process communication based on shared memory and address-mapping". In *Proceedings of International Conference on Computer Science and Network Technology* (San Jose CA USA), Vol. 1. 111–114. https://doi.org/10.1109/ICCSNT.2011.6181920
[5] Rust Foundation. [n.d.]. Rust Optimization Levels. https://doc.rust-lang.org/cargo/reference/profiles.html#default-profiles
[6] Rust Foundation. [n.d.]. Rust Programming Language. https://www.rust-lang.org/
[7] Rust Foundation. [n.d.]. Rust Time. https://doc.rust-lang.org/std/time/struct.Instant.html
[8] Google. [n.d.]. Go Programming Language. https://cloud.google.com/go
[9] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the Web up to Speed with WebAssembly. *SIGPLAN Not.* 52, 6 (June 2017), 185–200. https://doi.org/10.1145/3140587.3062363
[10] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. 2019. Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code. (July 2019), 107–120. https://www.usenix.org/conference/atc19/presentation/jangda
[11] Daniel Lehmann, Johannes Kinder, and Michael Pradel. 2020. Everything Old is New Again: Binary Security of WebAssembly. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 217–234. https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann
[12] LLVM. [n.d.]. LLVM. https://llvm.org/
[13] Lokathor. [n.d.]. Bytemuck. https://docs.rs/bytemuck/latest/bytemuck/
[14] lunatic.solutions. [n.d.]. Lunatic, Erlang inspired WASM runtime. https://lunatic.solutions/
[15] Nathan McCarty. [n.d.]. Bincode. https://github.com/bincode-org/bincode
[16] Nathan McCarty. [n.d.]. Bincode Specification. https://github.com/bincode-org/bincode#specification
[17] Microsoft. [n.d.]. ASP.NET Core Blazor hosting models. https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-5.0#blazor-webassembly
[18] Microsoft. [n.d.]. Query Performance Counter. https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter. https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter
[19] David J. Pearce. 2021. A Lightweight Formalism for Reference Lifetimes and Borrowing in Rust. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 43, 1 (April 2021), 1–73.
[20] Eric C. Reed. 2015. Patina : A Formalization of the Rust Programming Language.
[21] Veloren. [n.d.]. Veloren. https://gitlab.com/veloren/veloren
[22] W3C and Bytecode Alliance. [n.d.]. WASM (WebAssembly). https://webassembly.org/
[23] Wasmer. [n.d.]. Wasmer Backend Features. https://docs.wasmer.io/ecosystem/wasmer/wasmer-features
[24] Wasmer. [n.d.]. Wasmer Backend Performance. https://medium.com/wasmer/a-webassembly-compiler-tale-9ef37aa3b537
[25] Wasmer. [n.d.]. Wasmer, The Universal WebAssembly Runtime. https://wasmer.io/