How Far We’ve Come –
A Characterization Study of
Standalone WebAssembly Runtimes

Wenwen Wang

School of Computing
UNIVERSITY OF GEORGIA
What is WebAssembly?

A low-level and portable binary code format

**Design Goal:** facilitating the deployment of *non-web* applications on web platforms
Why WebAssembly?

**Advantages:**

- **Portability**
  - Independent on both programming languages and hardware platforms
- **Safety**
  - Sandboxed execution environment
- **Near-native speed**
  - Fast to compile to native binary code
WebAssembly is increasingly adopted **beyond the web**

- Trusted runtime environments for embedded systems
- Lightweight serverless frameworks for edge computing
- Software-fault isolation for serverless computing
- Resource accounting in remote computation
- ...

“A new foundation for pervasive computing”

— David Bryant (Mozilla Fellow)
Standalone WebAssembly Runtimes

The **key enabler** of WebAssembly outside the web

- Developed *specifically* to run WebAssembly binary code on a host physical machine without the need of browsers
- Just-In-Time (JIT) compilation or interpretation
- Much more lightweight than traditional language runtimes
However, there is very limited study about them

The characteristics of standalone WebAssembly runtimes are not clear

- How is the performance efficiency when running WebAssembly binaries?
- What amount of extra memory resource is consumed?
- Do they have an observable architectural impact?
- …
This work fills up this knowledge gap!

A characterization study of standalone WebAssembly runtimes

- Covering five popular standalone WebAssembly runtimes
- Constructing a benchmark suite, WABench
- Discovering many interesting findings
Methodology

Five standalone WebAssembly runtimes

- Actively developed and maintained
- Sufficiently mature to run a broad range of applications

<table>
<thead>
<tr>
<th>Runtime</th>
<th>Language</th>
<th>LoC</th>
<th>Type</th>
<th>#Stars</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasmtime</td>
<td>Rust</td>
<td>314K</td>
<td>JIT</td>
<td>7.6K</td>
<td>2 years</td>
</tr>
<tr>
<td>WAVM</td>
<td>C/C++</td>
<td>98K</td>
<td>JIT</td>
<td>2.2K</td>
<td>2 years</td>
</tr>
<tr>
<td>Wasmer</td>
<td>Rust</td>
<td>154K</td>
<td>JIT</td>
<td>12.4K</td>
<td>3 years</td>
</tr>
<tr>
<td>Wasm3</td>
<td>C</td>
<td>120K</td>
<td>Interpretation</td>
<td>5K</td>
<td>2 years</td>
</tr>
<tr>
<td>WAMR</td>
<td>C</td>
<td>145K</td>
<td>Interpretation</td>
<td>2.8K</td>
<td>2 years</td>
</tr>
</tbody>
</table>
WABench (50 Benchmarks)

A benchmark suite for standalone WebAssembly runtimes

- Existing benchmark suites
  - JetStream2, MiBench, and PolyBench

- Whole applications from a broad range of domains
  - bzip2, espeak, facedetection, gnuchess, mnist, snappy, and whitedb

Available at:

→ https://github.com/wabench/wabench
Overall Performance

Baseline: execution time of native execution

<table>
<thead>
<tr>
<th></th>
<th>Performance Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasmtime</td>
<td>1.67X</td>
</tr>
<tr>
<td>WAVM</td>
<td>3.54X</td>
</tr>
<tr>
<td>Wasmer</td>
<td>1.59X</td>
</tr>
<tr>
<td>Wasm3</td>
<td>6.99X</td>
</tr>
<tr>
<td>WAMR</td>
<td>9.57X</td>
</tr>
</tbody>
</table>

**Finding 1:** All WebAssembly runtimes introduce extra performance overhead, compared to native execution.
Impact of JIT Compilers

Three JIT compilers of Wasmer

- SinglePass (baseline), Cranelift, and LLVM

Finding 2: Cranelift and LLVM can achieve better performance than SinglePass.
Impact of Ahead-Of-Time (AOT) Compilation

Three JIT-based runtimes, baseline: w/o AOT

- Wasmtime, WAVM, and Wasmer

Finding 3: AOT has substantial performance impact on WAVM, but not Wasmtime and Wasmer.
Finding 4: WebAssembly binaries compiled with higher optimization levels can achieve better performance.
Finding 5: WebAssembly runtimes consume extra memory resources, compared to native execution.
Using hardware performance counters to measure IPC

Finding 6: WebAssembly runtimes exhibit higher IPC values than native execution.
### Branch Prediction Misses

<table>
<thead>
<tr>
<th></th>
<th>Normalized Branch Prediction Misses</th>
<th>Branch Prediction Miss Ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>1X</td>
<td>1.01%</td>
</tr>
<tr>
<td>Wasmtime</td>
<td>1.52X</td>
<td>0.77%</td>
</tr>
<tr>
<td>WAVM</td>
<td>8.99X</td>
<td>1.69%</td>
</tr>
<tr>
<td>Wasmer</td>
<td>1.56X</td>
<td>0.92%</td>
</tr>
<tr>
<td>Wasm3</td>
<td>12.64X</td>
<td>0.76%</td>
</tr>
<tr>
<td>WAMR</td>
<td>8.14X</td>
<td>0.53%</td>
</tr>
</tbody>
</table>

**Finding 7:** WebAssembly runtimes have more branch misses, but the branch miss ratios are very close to native execution.
## Cache Misses

<table>
<thead>
<tr>
<th></th>
<th>Normalized Cache Misses</th>
<th>Cache Miss Ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>1X</td>
<td>11.13%</td>
</tr>
<tr>
<td>Wasmtime</td>
<td>1.91X</td>
<td>12.98%</td>
</tr>
<tr>
<td>WAVM</td>
<td>4.60X</td>
<td>5.57%</td>
</tr>
<tr>
<td>Wasmer</td>
<td>1.73X</td>
<td>13.26%</td>
</tr>
<tr>
<td>Wasm3</td>
<td>1.39X</td>
<td>7.97%</td>
</tr>
<tr>
<td>WAMR</td>
<td>1.60X</td>
<td>8.99%</td>
</tr>
</tbody>
</table>

**Finding 8:** WebAssembly runtimes have more cache misses, but the cache miss ratios are very close to native execution.
Discussion

**Insight 1:** WebAssembly users need to select an *appropriate* standalone WebAssembly runtime.

**Insight 2:** It is necessary for standalone WebAssembly runtime developers to carefully *tune* the performance.

**Insight 3:** Developing *WebAssembly*-specific optimizations may further improve the performance.
Summary

- This work makes an attempt towards characterizing standalone WebAssembly runtimes
  - Covering five popular standalone WebAssembly runtimes
  - Constructing a benchmark suite, namely WABench
  - Discovering many interesting findings

- More research work is required to further understand the rationales behind the characteristics

**WABench** is available at

→ [https://github.com/wabench/wabench](https://github.com/wabench/wabench)

Thank you!