Compilation of Scheme to WebAssembly

L3 internship

Hubert Gruniaux[1]
Superviser: Manuel Serrano[2]

[1]ENS-PSL

[2]Inria Sophia-Antipolis

September 2024

Scheme

- Based on Lisp
- Dynamically typed functional language
- Runtime polymorphism

```
(define (fib n)
  (if (< n 2)
    n
    (+
      (fib (- n 1))
      (fib (- n 2)))))
```

Bigloo

- An optimizing Scheme compiler
- A large project and old project (since the 90s)
  - The compiler: 100,000 lines of Scheme
  - The runtime: 70,000 lines of Scheme and 20,000 of C
- Two backends:
  - C code generation (*cgen* and *saw-c*)
  - Java bytecode generation (*saw-jvm*)
  - ... and our new WASM backend

WebAssembly

- Low-level virtual machine specification introduced in 2016
- High performance code execution inside web browsers
- Initially to port native code (C, Rust, etc.) to the Web
  - Modern alternative to `asm.js`
- New proposals useful for dynamic functional languages:
  - Garbage collector (october 2023 in Chrome),
  - First-class functions,
  - Tail calls (avril 2023 in Chrome), etc.
- Public presentation of Guile Hoot and Wasocaml at ICFT 2024
  - Scheme and ML Workshops

Representation of types

- Primitive types use WASM `i32`, `i64`, `f32` and `f64`
- Boxed types use WASM GC types
  - Eitheir GC structures:

    ```
    (type $bint (struct (field i64)))
    ```

  - Or GC arrays:

    ```
    (type $bstring (array (mut i8)))
    ```

  - The Scheme `obj` is WASM `eqref`

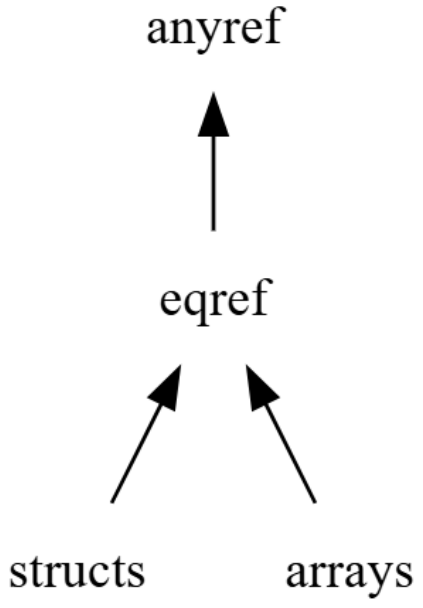

Figure: Hierarchy of GC types

Compilation of literals

- Boxed constants literals are globals
- Problem with non-trivial constants like strings or lists
  - Bigloo already supported lists manual initialization
- Globals initialization code must be *constant*,
  - BUT, `array.new_data` instruction is not constant

```
(data $my-str-data "Hello")
(global $my-str (ref null $bstring) (ref.null none))
(func $my-module-init
  (; ... ;)
  (global.set $my-str
    (array.new_data $bstring $my-str-data
      (i32.const 0)
      (i32.const 5))))
```

Implementation of classes

- Bigloo classes are lowered as GC structures
- We use WASM struct inheritance and subtyping

```
(type $my-parent-class
  (sub
    (struct (field i32))))
(type $my-class
  (sub $my-parent-class
    (struct (field i32 f32))))
```

Structural equivalence

- WASM uses structural equivalence
- Two classes with the same types are equivalent in WASM!
- The solution: using `rec` clauses

```
(rec (type $my-class (struct (; fields ;))))
```

Control flow in WebAssembly

- Use structured control flow based on a stack
- No arbitrary gotos
- BUT, the backend uses basic blocks implemented using gotos

```
(block $label
    (; code... ;)
    ;; Jumps to <TARGET>
    (br $label)
    (; code... ;))
(; <TARGET> ;)
```

```
(loop $label
    (; <TARGET> ;)
    (; code... ;)
    ;; Jumps to <TARGET>
    (br $label)
    (; code... ;))
```

Naive approach: dispatcher pattern

- We can use a big `switch` to emulate gotos

```
(local $label i32)
(local.set $label (i32.const 0))
(loop $dispatcher
  (block $bb_0
    (; ... ;)
    (br_table $bb_0 (; ... ;) $bb_n (local.get $label)))
  ;; Basic block 0 code...
  ;; Jumps to basic block 5
  (local.set $label (i32.const 5))
  (br $dispatcher))
```

Drawbacks with the dispatcher pattern

- Destroys the static analysis of the control flow graph
  - By the JIT compiler and WASM assembler
- Prevent use of non-nullable references in locals
  - WASM requires that *uses* are dominated by a *definition* for non-nullable references
  - We must use nullable references, possible slowdown
- Less readable code
- More difficult to debug

A better approach: Relooper

- Problem is not new!
  - B.S. Baker, "An Algorithm for Structuring Flowgraphs" (1977)
  - A. Zakai, "Emscripten: [...]" (2011)
- The idea: reconstruct the control flow with higher structures
  - We use *dominator trees* and *reverse postorder numbers*
  - N. Ramsey, "Beyond Relooper: [...]" (2022)
  - Same algorithm used by Guile Hoot and Wasocaml

Limits of Relooper

- Limited to reducible CFGs
- Almost all CFGs are reducibles in Bigloo
- But irreducible CFGs may happen in practice
  - Compilation of regular expressions
  - Bigloo BBV optimization
- In that case:
  - We can transform irreducible CFGs to reducible ones
  - In Bigloo, we simply fallback to the dispatcher pattern

Interfacing with JavaScript

- Not everything can be done in WASM
  - Like interfacing with the OS
  - Two possibilities: JavaScript or WASI
- JS can call WASM functions and vice-versa
- But not the same type representation
  - Strings must be copied to a common memory (costly)
  - GC references can not be passed to JS

OS interface using NodeJS API

- Bigloo runtime requires POSIX
  - We use the NodeJS API
  - Not compatible with web browsers
- Bigloo C have dependencies on external C libraries
  - PCRE for regular expressions
  - GMP for big numbers
  - pthread for threads
  - etc.
- Difficult to call C code compiled to WASM
  - Emscripten and C do not support GC
  - Use of a virtual stack (difficult to call C functions)

The current state of Bigloo-WASM

- Most of compilation work is done
  - Call/CC not supported
- Still work to be done for the runtime library
  - Threads
  - Regular expressions
  - Network (POSIX sockets)
- The project is usable but still experimental

Benchmarks I

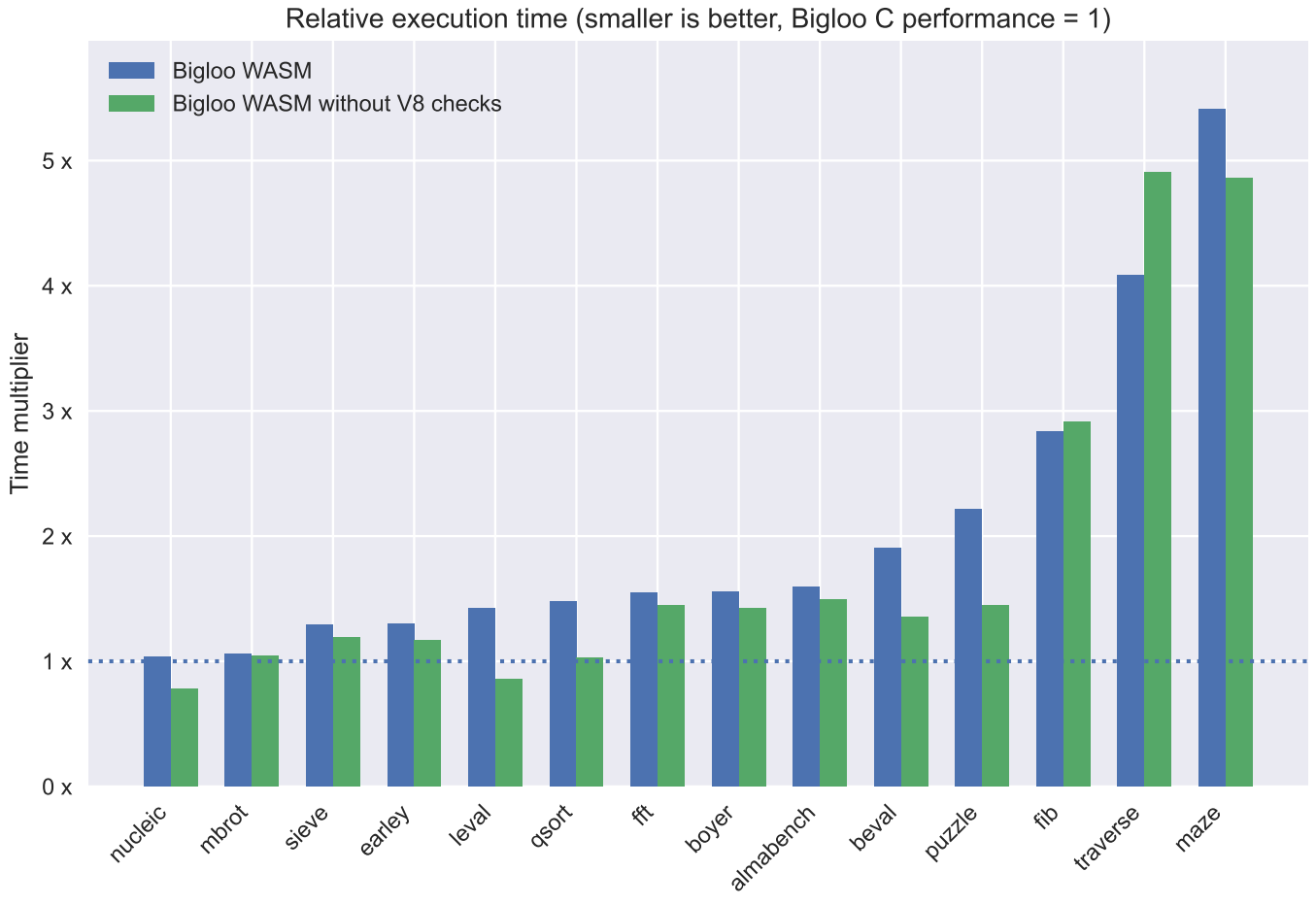

Figure: Benchmarks results

Benchmarks II

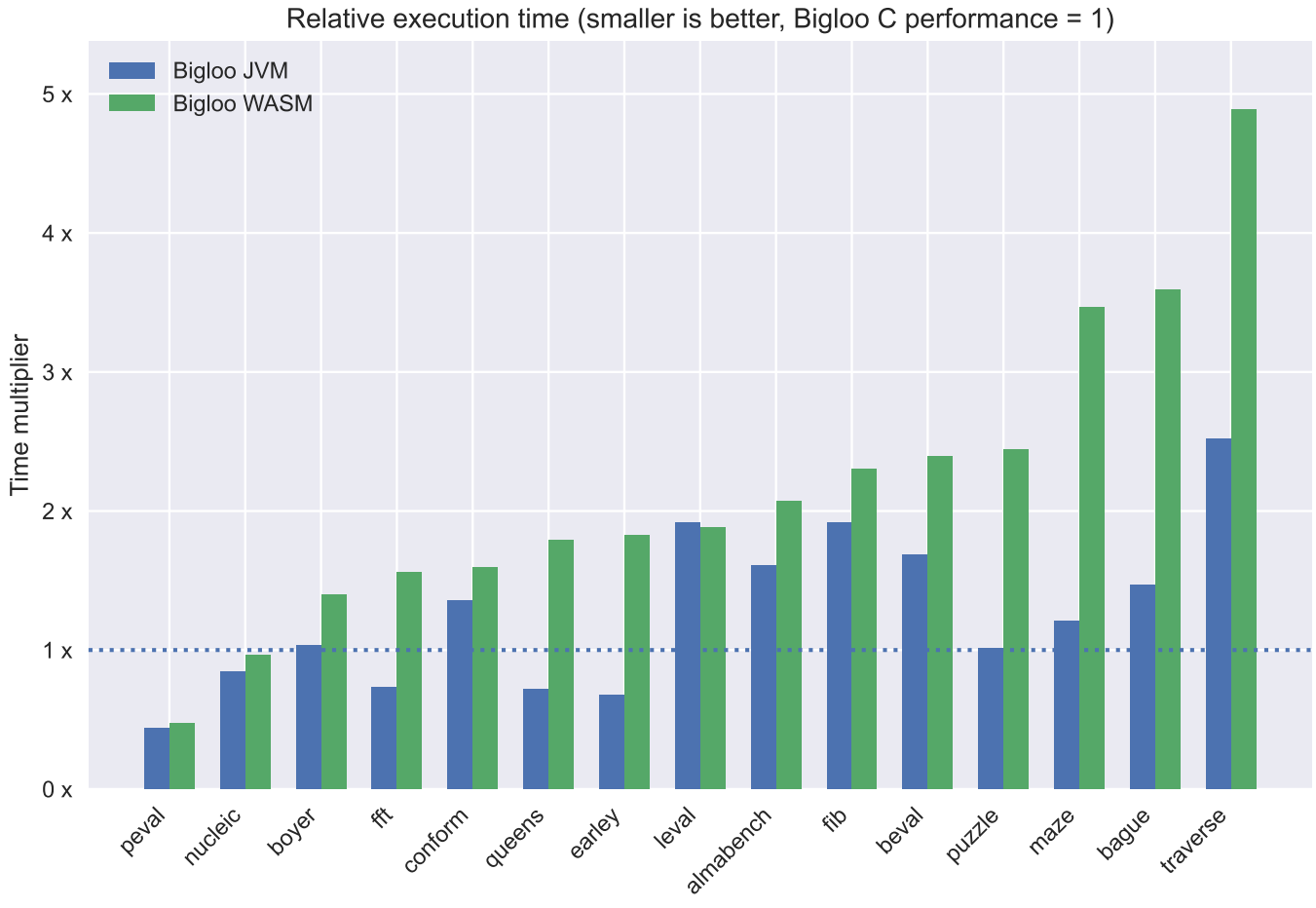

Figure: Benchmarks results

Conclusion

- WASM-GC is fast
- Ecosystem not mature enough
  - WASM-GC not supported by WebKit (will be in next version)
  - Support in Wasmtime, Wasmer lacking or experimental
  - Many bugs in toolchains (binaryen, Deno)
  - Multiple crashes with Chrome Devtools and NodeJS
- No WASM runtime and not enough libraries
- No standardized ABI between languages for WASM
- https://github.com/hgruniaux/bigloo-wasm