Evaluation of WebAssembly IoT Runtimes on a ESP32 Microcontroller

Evaluation von WebAssembly IoT Runtimes auf einem ESP32 Microcontroller

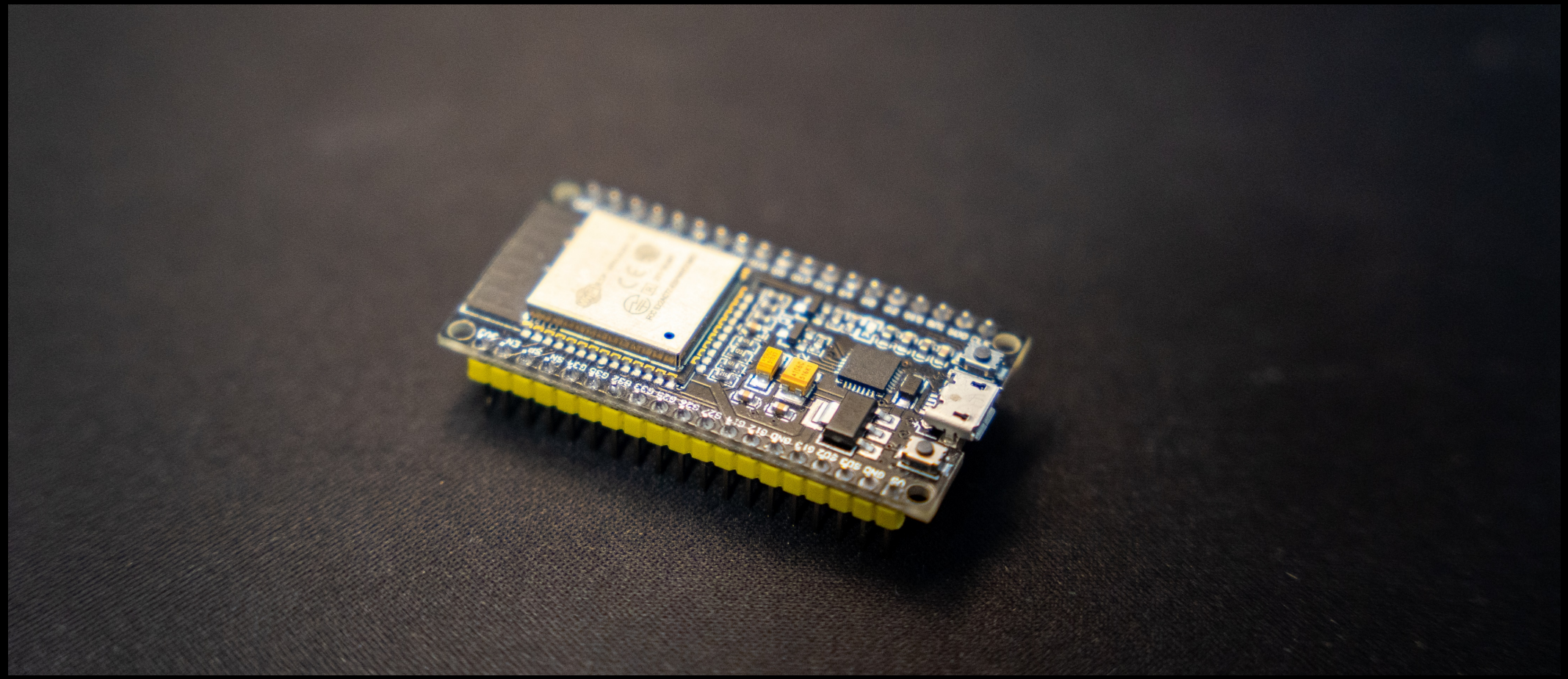
Abstract

- Run WebAssembly on the ESP32
- Found runtime
- WASM code runs on the ESP32 with decreased performance

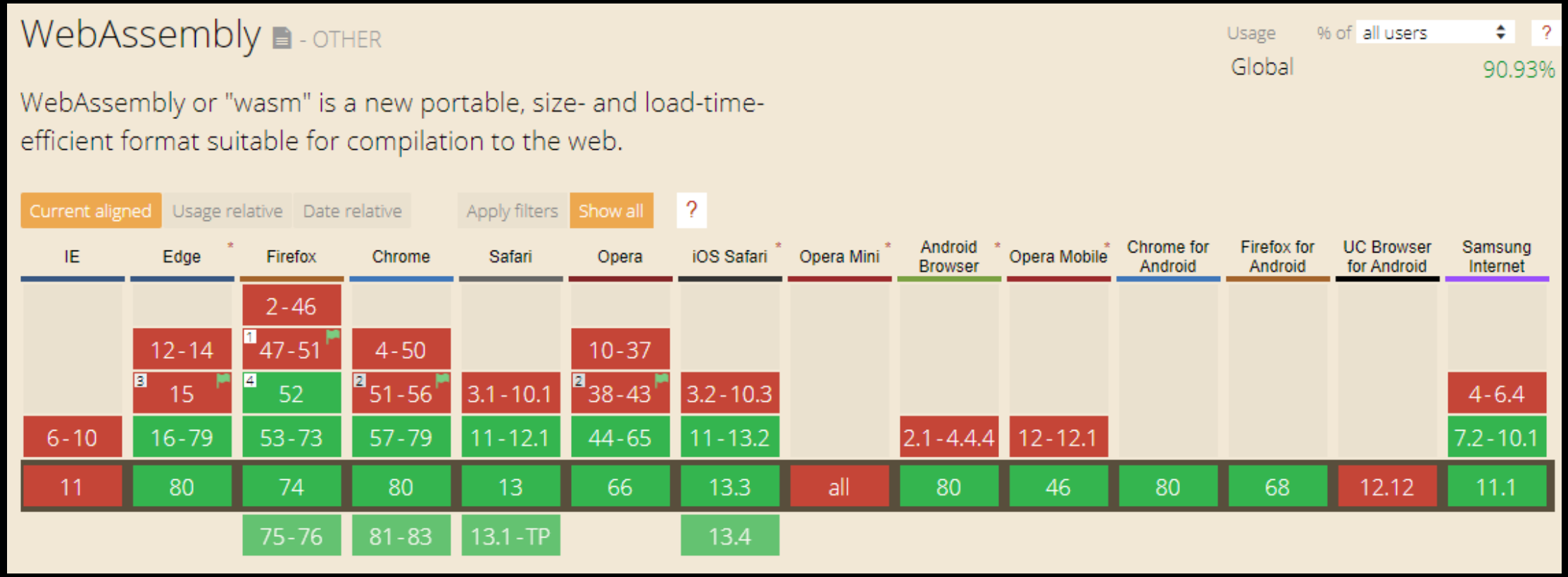
Outline

- Background: ESP32 and WebAssembly
- Methodology: Runtime and tests
- Evaluation: Rest results and learnings

The ESP32



WebAssembly adoption



WebAssembly

- Portable target for languages like C/C++ and Rust
- Shipped in self-contained modules
- Runs on stack machine
- No web dependencies
- Advantageous attributes for IoT
- Structured control flow

Structured control flow

instr ::= . . .

| nop
| unreachable
| block resulttype instr* end
| loop resulttype instr* end
| if resulttype instr* else instr* end
| br labelidx
| **br_if labelidx**
| br_table vec(labelidx) labelidx
| **return**
| **call funcidx**
| call_indirect typeidx

Methodology

Running WebAssmbly

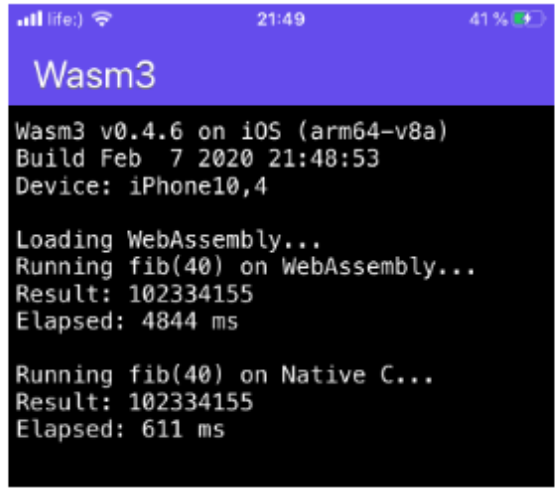
README.md

Wasm3

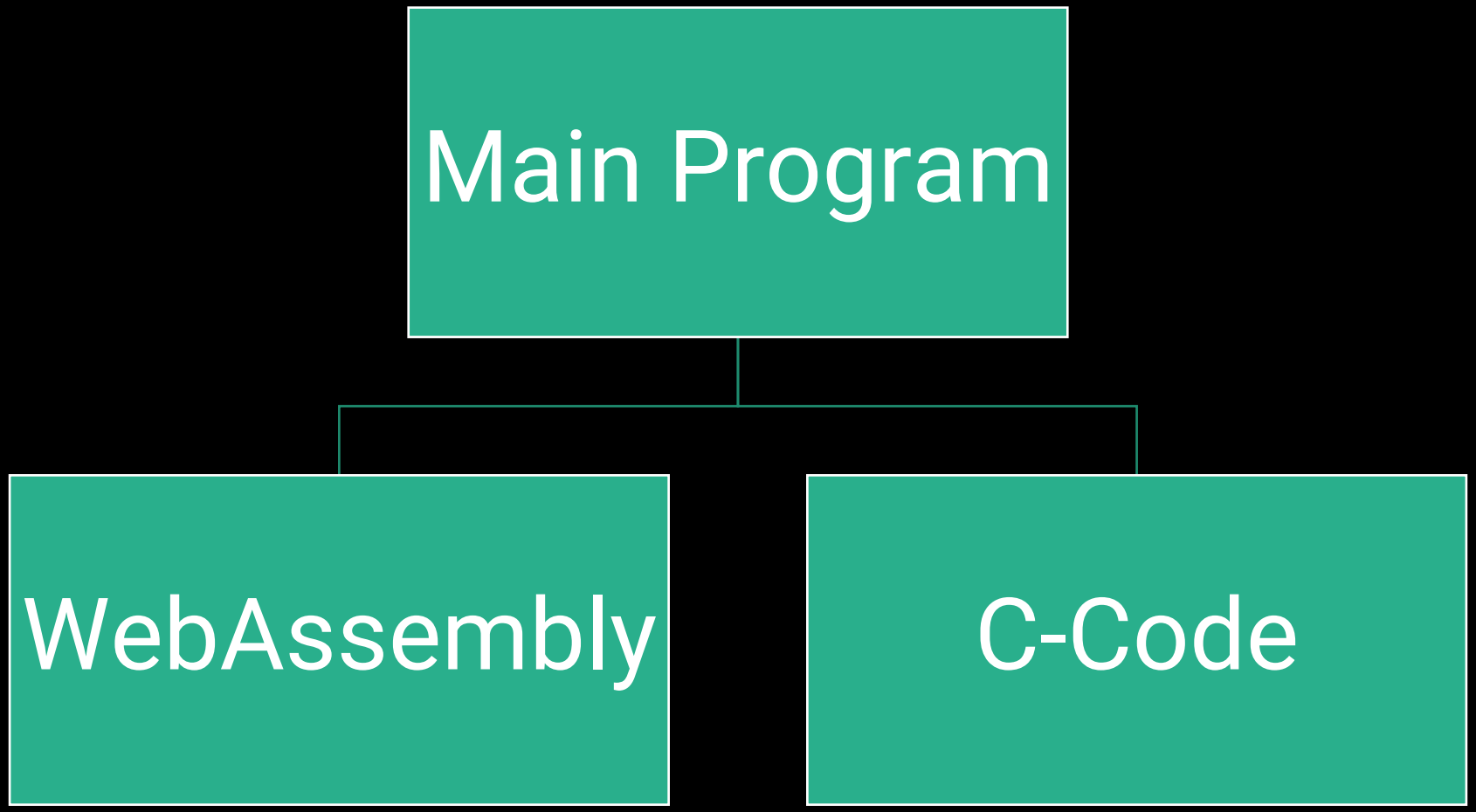
wapm v0.4.6 | issues 17 | tests failing | license MIT

A high performance WebAssembly interpreter written in C.

~ **8x faster** than other known wasm interpreters
~ **4-5x slower** than state of the art wasm `JIT` engines
~ **12x slower** than native execution



Testing setup



Recursive: C++

```
uint32_t run(uint32_t n) {
    if (n < 2) {
        return n;
    }
    return run(n - 1) + run(n - 2);
}
```

Recursive: WAT

```
(func $run (export "run") (type $t1) (param $p0 i32) (result i32)
  (block $B0
    (br_if $B0
      (i32.lt_u
        (local.get $p0)
        (i32.const 2)))
    (return
      (i32.add
        (call $run
          (i32.add
            (local.get $p0)
            (i32.const -1)))
        (call $run
          (i32.add
            (local.get $p0)
            (i32.const -2))))))
  (local.get $p0))
```

Recursive: Comparison

```
uint32_t run(uint32_t n) {
    if (n < 2) {
        return n;
    }
    return run(n - 1) + run(n - 2);
}
```

```
(block $B0
  (br_if $B0
    (i32.lt_u
      (local.get $p0)
      (i32.const 2)))
  (return
    (i32.add
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -1)))
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -2))))))
(local.get $p0)
```

Recursive: Comparison

```
uint32_t run(uint32_t n) {
    if (n < 2) {
        return n;
    }
    return run(n - 1) + run(n - 2);
}
```

```
(block $B0
  (br_if $B0
    (i32.lt_u
      (local.get $p0)
      (i32.const 2)))
  (return
    (i32.add
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -1)))
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -2))))))
(local.get $p0)
```

Recursive: Comparison

```
uint32_t run(uint32_t n) {
    if (n < 2) {
        return n;
    }
    return run(n - 1) + run(n - 2);
}
```

```
(block $B0
  (br_if $B0
    (i32.lt_u
      (local.get $p0)
      (i32.const 2)))
  (return
    (i32.add
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -1)))
      (call $run
        (i32.add
          (local.get $p0)
          (i32.const -2))))))
(local.get $p0)
```

Testcases

Recursive

Switch

Memory

Matrix Multiply

Native Calls

TypeScript

Evaluation

Measurements

Test case	WASM Execution	Native Exectuion	Slowdown factor
Recursive Calls	41,766 µs (σ=0.79)	1,000 µs (σ=0.28)	42
Switch statement	1,021 µs (σ=0.71)	0 µs (σ=0.00)	
Memory Access	1,802 µs (σ=0.74)	55 µs (σ=0.26)	33
Matrix Multiplication	26,277 µs (σ=0.60)	281 µs (σ=0.58)	93
Native calls*	33 µs (σ=5.15)	8 µs (σ=0.64)	4
TypeScript execution	41,493 µs (σ=0.85)	1,000 µs (σ=0.24)	41

Learnings

Drawbacks

- Performance loss
- Significant limitations

Potential

- Dynamic code
- Browser execution
- New languages

Conclusion