Benchmarking WasmFX

Daniel Hillerström

Computing Systems Laboratory
Zurich Research Center
Huawei Technologies, Switzerland
and
The University of Edinburgh, UK

May 20, 2024

WebAssembly Stacks Subgroup

`https://dhil.net`

Collaborators

Sam Lindley

Andreas Rossberg

Daan Leijen

KC Sivaramakrishnan

Matija Pretnar

Frank Emrich

Luna Phipps-Costin

Arjun Guha

`https://wasmfx.dev`

Collaborators

Sam Lindley

Andreas Rossberg

Daan Leijen

KC Sivaramakrishnan

Matija Pretnar

Frank Emrich

Luna Phipps-Costin

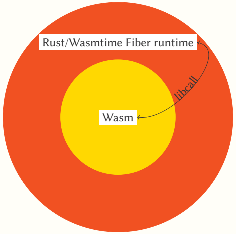
Arjun Guha

`https://wasmfx.dev`

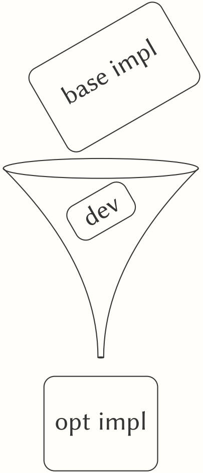
Status: WasmFX in Wasmtime

Implementation status

- Switching stacks via libcalls to Wasmtime Fiber



- Continuations and their stacks are safe
- Fiber stacks are pooled
- Continuation metadata is unpooled
- Continuation arguments are boxed



Benchmark definitions

Definition: *Microbenchmark*

A “small” program designed to measure the performance of a single operation of the system.

Definition: *Macrobenchmark*

A program that is representative of some “real” workload, where context switching is inherent.

What we (micro)benchmark

General setup

- Source language: C with a bespoke fiber library
  - Asyncify implementation
  - WasmFX implementation
- Requirement: all fibers gracefully terminate (i.e. successful return or cancellation)

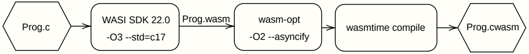
Fibers interface in C

```c
/** The signature of a fiber entry point. **/
typedef void* (*fiber_entry_point_t)(void*);
/** The abstract type of a fiber object. **/
typedef struct fiber* fiber_t;

/** Allocates a new fiber with the default stack size. **/
fiber_t fiber_alloc(fiber_entry_point_t entry);
/** Reclaims the memory occupied by a fiber object. **/
void fiber_free(fiber_t fiber);

/** Yields control to its parent context. **/
void* fiber_yield(void *arg);

/** Possible status codes for ‘fiber_resume‘. **/
typedef enum { FIBER_OK, FIBER_YIELD, FIBER_ERROR } fiber_result_t;

/** Resumes a given ‘fiber‘ with argument ‘arg‘. **/
void* fiber_resume(fiber_t fiber, void *arg, fiber_result_t *result);
```

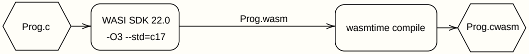
Experiments setup

Compilation pipelines

- Asyncify



- WasmFX



Apples & oranges

- Different storage
  - Asyncify-backed fibers in linear memory
  - WasmFX-backed fibers in tables
- Clang unwilling to generate function references

Microbenchmark: Prime sieve

Description

- Actor-based concurrency simulation
- Computes the first 8100 prime numbers
- 8100 coroutines, multiple yields
- Shallow call stack

	Run-time ratio	Binary size ratio
Asyncify	1.00	1.05 (41kb)
WasmFX (base)	5.31	1.0 (39kb)
WasmFX (dev)		1.0 (39kb)

Lower is better

Microbenchmark: Prime sieve

Description

- Actor-based concurrency simulation
- Computes the first 8100 prime numbers
- 8100 coroutines, multiple yields
- Shallow call stack

	Run-time ratio	Binary size ratio
Asyncify	1.00	1.05 (41kb)
WasmFX (base)	5.31	1.0 (39kb)
WasmFX (dev)	3.25	1.0 (39kb)

Lower is better

Microbenchmark: C10m

Description

- HTTP server workload simulation
- 10 million coroutines in total
- Sliding window: 10000 coroutines run concurrently, each yielding once
- Shallow call stack depth

	Run-time ratio	Binary size ratio
Asyncify	1.00	12.72 (9.1kb)
WasmFX (base)	3.87	1.0 (723b)
WasmFX (dev)		1.0 (723b)

Lower is better

Microbenchmark: C10m

Description

- HTTP server workload simulation
- 10 million coroutines in total
- Sliding window: 10000 coroutines run concurrently, each yielding once
- Shallow call stack depth

	Run-time ratio	Binary size ratio
Asyncify	1.00	12.72 (9.1kb)
WasmFX (base)	3.87	1.0 (723b)
WasmFX (dev)	2.76	1.0 (723b)

Lower is better

Microbenchmark: Skynet

Description

- Nested tree-structured concurrency simulation
- 10 million coroutines in total, 6 active, each yielding once
- Deep call stack

	Run-time ratio	Binary size ratio
Asyncify	1.00	27.52 (9kb)
WasmFX (base)	4.18	1.0 (327b)
WasmFX (dev)		1.0 (327b)

Lower is better

Microbenchmark: Skynet

Description

- Nested tree-structured concurrency simulation
- 10 million coroutines in total, 6 active, each yielding once
- Deep call stack

	Run-time ratio	Binary size ratio
Asyncify	1.00	27.52 (9kb)
WasmFX (base)	4.18	1.0 (327b)
WasmFX (dev)	3.25	1.0 (327b)

Lower is better

Microbenchmark: Hello World

Description

- Cooperatively printing of “`Hello World`”
- 2 coroutines, print one letter, yield
- Print operation and yield in loop

	Run-time ratio	Binary size ratio
Asyncify	2.95	1.4 (33kb)
WasmFX (base)	1.0	1.0 (24kb)
WasmFX (dev)	1.0	1.0 (24kb)

Lower is better

Microbenchmark: C10m revisited

Description

- HTTP server workload simulation
- 10 million coroutines in total
- Sliding window: 10000 coroutines run concurrently, each yielding once
- Shallow call stack depth
- I/O call in hot loop

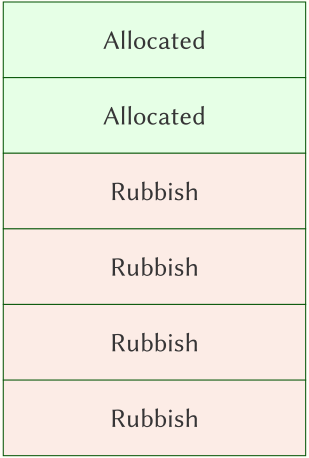
		Run-time ratio	Binary size ratio
No I/O	Asyncify	1.00	12.72 (9.1kb)
	WasmFX (base)	3.87	1.0 (723b)
	WasmFX (dev)	2.76	1.0 (723b)
I/O	Asyncify	1.00	12.15 (9.2kb)
	WasmFX (base)	1.41	1.0 (757b)
	WasmFX (dev)	1.38	1.0 (757b)

Lower is better

Unsafe stacks

Unsafe stacks

- Allocated via malloc
- On demand allocation



Allocated

Allocated

Rubbish

Rubbish

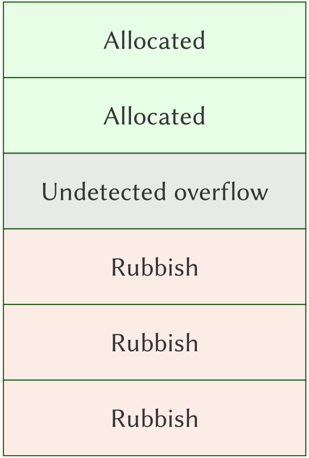
Rubbish

Rubbish

Unsafe stacks

Unsafe stacks

- Allocated via malloc
- On demand allocation



Allocated

Allocated

Undetected overflow

Rubbish

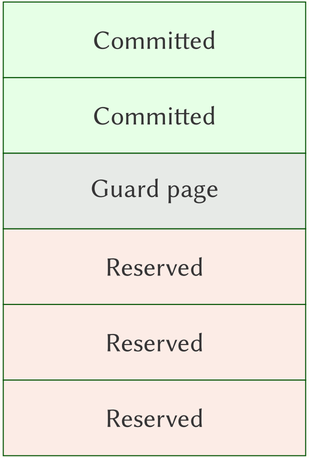
Rubbish

Rubbish

Safe stacks

Safe stacks

- Always allocated via mmap
- Guard pages delimit stacks
- Stack pools
- Suggestive scheme for stack growing



Committed

Committed

Guard page

Reserved

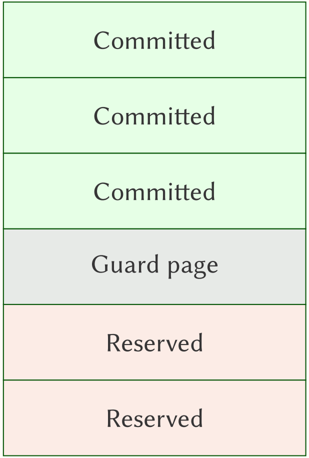
Reserved

Reserved

Safe stacks

Safe stacks

- Always allocated via mmap
- Guard pages delimit stacks
- Stack pools
- Suggestive scheme for stack growing



Committed

Committed

Committed

Guard page

Reserved

Reserved

Microbenchmark: C10m revisited, again

Description

- HTTP server workload simulation
- 10 million coroutines in total
- Sliding window: 10000 coroutines run concurrently, each yielding once
- Shallow call stack depth

	Run-time ratio	Binary size ratio
Asyncify	1.00	12.72 (9.1kb)
WasmFX (dev/pool)	2.76	1.0 (723b)
WasmFX (dev/no pool)	187.73	1.0 (723b)

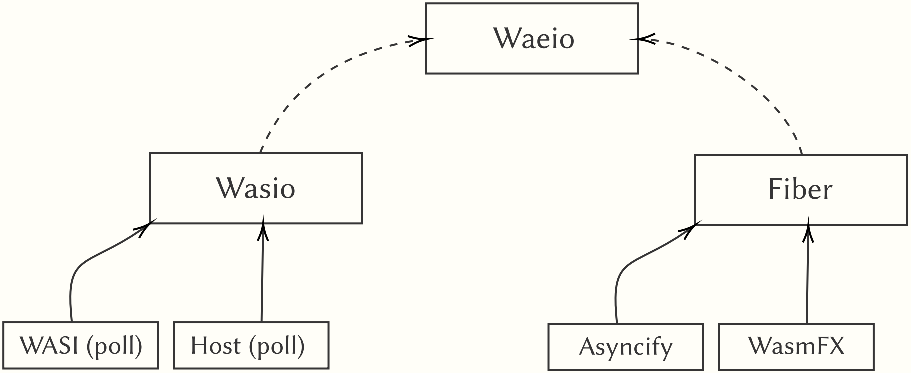
Lower is better

What we (macro)benchmark

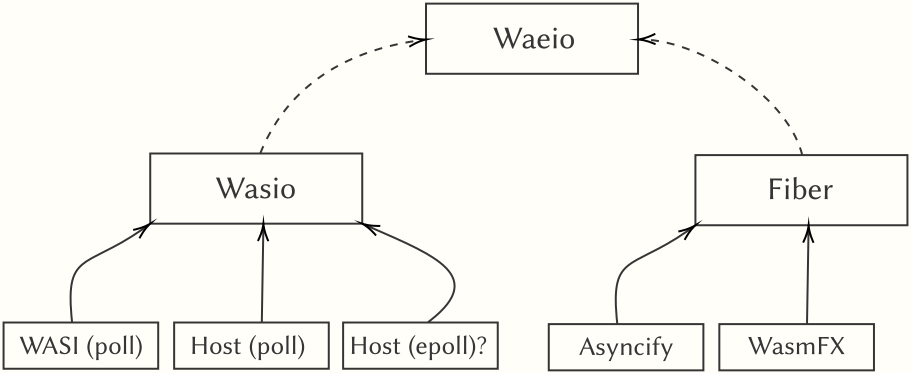
HTTP server

- HTTP/1.1 servers written in C using Waeio (bespoke library)
- Waeio: a prototype framework for interleaving I/O using stack switching
- We serve a static page on `/`, and kill the server on `/quit`
- We measure throughput and tail latency

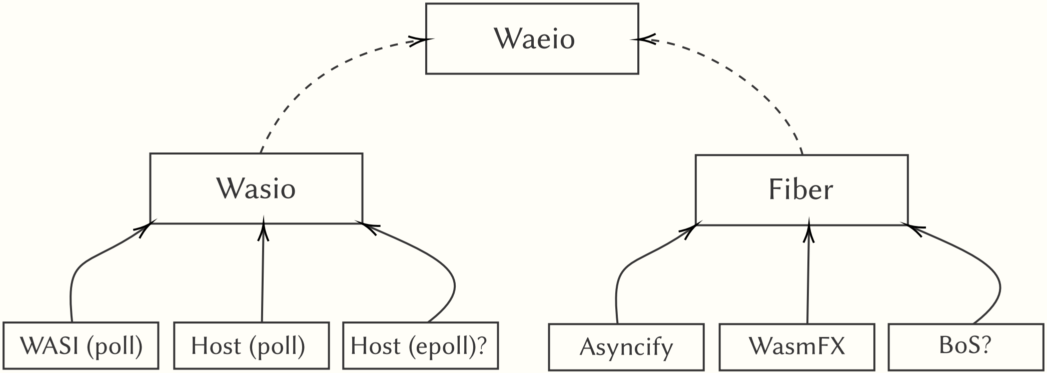
Waeio: An effect-based I/O library



Waeio: An effect-based I/O library



Waeio: An effect-based I/O library



Macrobenchmark setup

Setup

- Setup is the same as for microbenchmarks (+Waeio)

Http parser

- picohttpparser (main branch commit `f8d0513`)
- `https://github.com/h2o/picohttpparser`

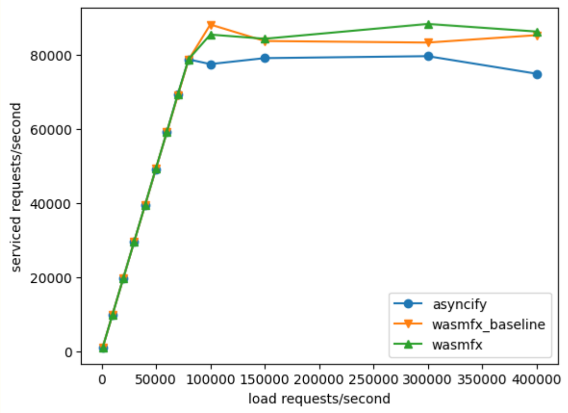
Workload generator

- wrk2 (main branch commit `44a94c1`)
- `https://github.com/giltene/wrk2`
- Options:
    - `-t4 -c1000 -R{80,60,40}000 -d60s`

Binary size

- Asyncify: 41kb (1.37×)
- WasmFX: 30kb
- Host driver: 30mb (statically linked)

Macrobenchmark: HTTP server throughput

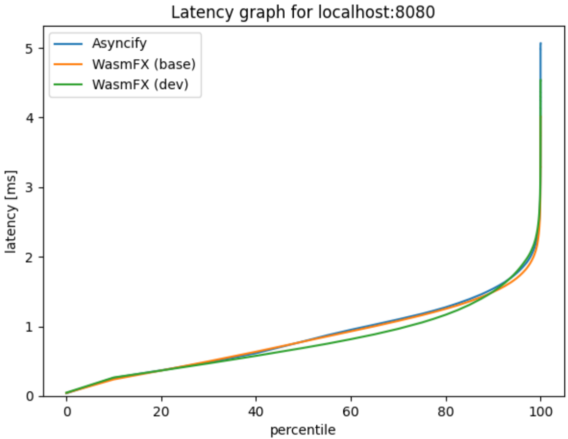


	Peak (req/s)
Asyncify	79587
WasmFX (base)	88116
WasmFX (dev)	88270

Higher is better

Macrobenchmark: HTTP server 40K req/s

40000 requests/sec

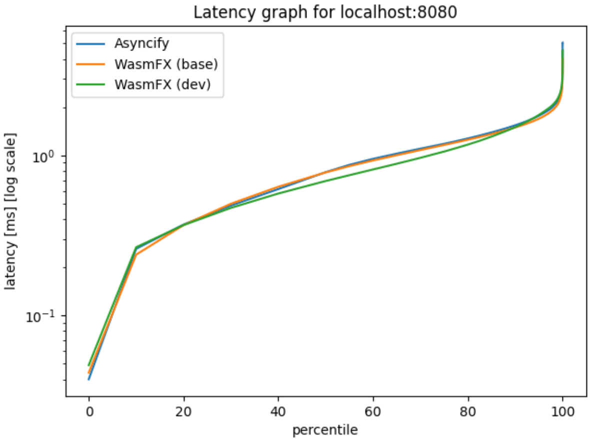


	Max (ms)
Asyncify	6.6
WasmFX (base)	6.0
WasmFX (dev)	6.3

Lower is better

Macrobenchmark: HTTP server 40K req/s

40000 requests/sec

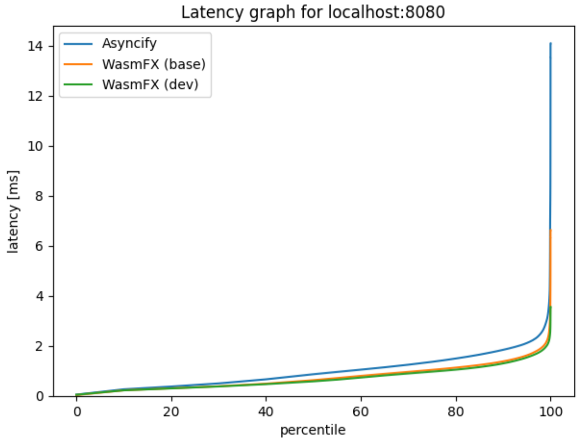


	Max (ms)
Asyncify	6.6
WasmFX (base)	6.0
WasmFX (dev)	6.3

Lower is better

Macrobenchmark: HTTP server 60K req/s

60000 requests/sec

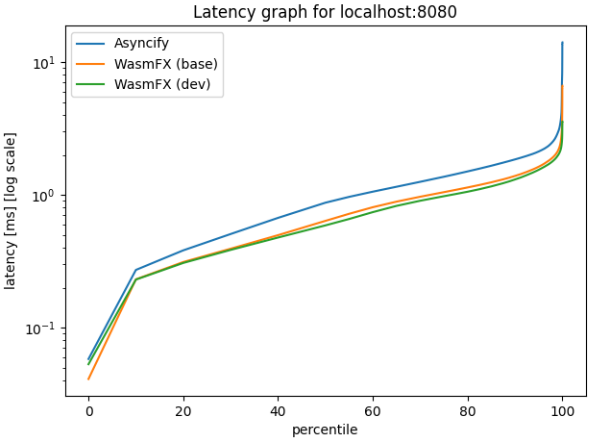


	Max (ms)
Asyncify	14.89
WasmFX (base)	7.6
WasmFX (dev)	6.3

Lower is better

Macrobenchmark: HTTP server 60K req/s

60000 requests/sec

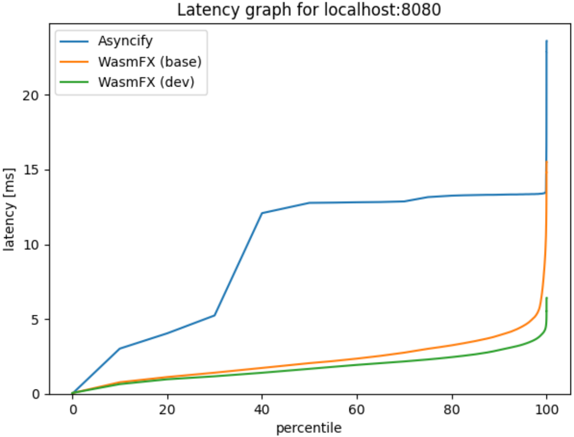


	Max (ms)
Asyncify	14.89
WasmFX (base)	7.6
WasmFX (dev)	6.3

Lower is better

Macrobenchmark: HTTP server 80K req/s

80000 requests/sec

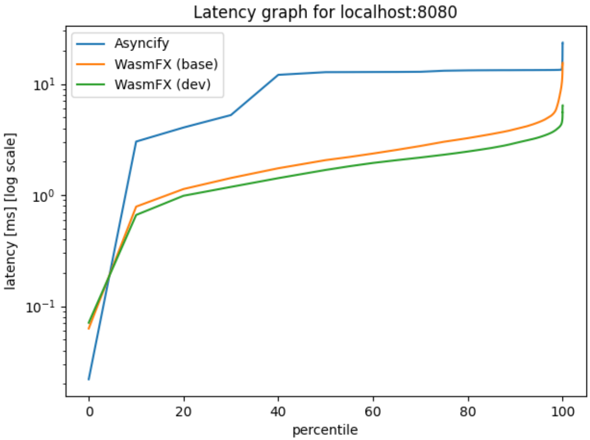


	Max (ms)
Asyncify	742
WasmFX (base)	16
WasmFX (dev)	8

Lower is better

Macrobenchmark: HTTP server 80K req/s

80000 requests/sec



	Max (ms)
Asyncify	742
WasmFX (base)	16
WasmFX (dev)	8

Lower is better

Discussion: Benchmarks

Which kind of programs should we benchmark?

- Microbenchmarks: what are the key interesting properties to measure?
- Macrobenchmarks: what are some inherently stack-switching-y representative workloads?

Discussion: Benchmarks

Which kind of programs should we benchmark?

- Microbenchmarks: what are the key interesting properties to measure?
- Macrobenchmarks: what are some inherently stack-switching-y representative workloads?

Task-oriented programs

- Http servers
- Generator programs?
- HPC?
- Canonical work stealing benchmark?

Discussion: Benchmarks

Which kind of programs should we benchmark?

- Microbenchmarks: what are the key interesting properties to measure?
- Macrobenchmarks: what are some inherently stack-switching-y representative workloads?

Task-oriented programs

- Http servers
- Generator programs?
- HPC?
- Canonical work stealing benchmark?

Multifaceted stack switching

- What are some representative workloads that combine stack switching features?

WasmFX resource list

Latest resources

- Waeio (`https://github.com/wasmfx/waeio`)
- Fiber library (`https://github.com/wasmfx/fiber-c`)
- Benchmark suite (`https://github.com/wasmfx/benchfx`)

Previous resources

- Formal specification (`https://github.com/WebAssembly/stack-switching/blob/wasmfx/proposals/continuations/Overview.md`)
- Informal explainer document (`https://github.com/WebAssembly/stack-switching/blob/wasmfx/proposals/continuations/Explainer.md`)
- Reference implementation (`https://github.com/WebAssembly/stack-switching/tree/wasmfx`)
- Wasmtime implementation (`https://github.com/wasmfx/wasmfxtime`)
- Toolchain support (`https://github.com/wasmfx/binaryenfx`)
- OOPSLA'23 research paper (`https://doi.org/10.48550/arXiv.2308.08347`)

`https://wasmfx.dev`

References I

Phipps-Costin, Luna et al. (2023). “Continuing WebAssembly with Effect Handlers”. In: *Proc. ACM Program. Lang.* 7.OOPSLA2, pp. 460–485.