Typed Continuations, the Wasmtime Perspective

Daniel Hillerström

Computing Systems Laboratory
Zurich Research Center
Huawei Technologies, Switzerland

July 17, 2023

I am but one of many

Sam Lindley

Andreas Rossberg

Daan Leijen

KC Sivaramakrishnan

Matija Pretnar

Frank Emrich

Luna Phipps-Costin

Arjun Guha

`https://wasmfx.dev`

I am but one of many

Sam Lindley

Andreas Rossberg

Daan Leijen

KC Sivaramakrishnan

Matija Pretnar

Frank Emrich

Luna Phipps-Costin

Arjun Guha

`https://wasmfx.dev`

Paper accepted at OOPSLA'23

Continuing WebAssembly with Effect Handlers

LUNA PHIPPS-COSTIN, Northeastern University, USA
ANDREAS ROSSBERG, Unaffiliated, Germany
ARJUN GUHA, Northeastern University, USA
DAAN LEIJEN, Microsoft Research, USA
DANIEL HILLERSTRÖM, Huawei Zurich Research Center, Switzerland
KC SIVARAMAKRISHNAN, Tarides and IIT Madras, India
MATIJA PRETNAR, Ljubljana University, Slovenia
SAM LINDLEY, The University of Edinburgh, United Kingdom

WebAssembly (Wasm) is a low-level portable code format offering near native performance. It is intended as a compilation target for a wide variety of source languages. However, Wasm provides no direct support for non-local control flow features such as async/await, generators/iterators, lightweight threads, first-class continuations, etc. This means that compilers for source languages with such features must ceremoniously transform whole source programs in order to target Wasm.

We present *WasmFX*, an extension to Wasm which provides a universal target for non-local control features via *effect handlers*, enabling compilers to translate such features directly into Wasm. Our extension is minimal and only adds three main instructions for creating, suspending, and resuming continuations. Moreover, our primitive instructions are type-safe providing typed continuations which are well-aligned with the design principles of Wasm whose stacks are typed.

OOPSLA'23 artifact

README.md

Artifact Evaluation Instructions

This document describes how to reproduce the experiments in Section 5.1 of the paper:

Luna Phipps-Costin, Andreas Rossberg, Arjun Guha, Daan Leijen, Daniel Hillerström, KC Sivaramakrishnan, Matija Pretnar, Sam Lindley, "Continuing WebAssembly with Effect Handlers", Proc. ACM Program. Lang. 7(OOPSLA2), 2023.

Note: The empirical data presented in the paper were measured on a particular reference machine available at the time of writing. In order to reproduce this data, it would be necessary to have access to the particular reference machine (or a virtually identical one). The purpose of this document is to describe how to obtain measurements like those reported in the paper.

This document is best viewed on GitHub (https://github.com/wasmfx/oopsla23-artifx).

Overview of the Artifact

The artifact is structured as follows

1. The section Getting Started Guide enumerates the software and hardware requirements to build and run the artifact software.
2. The section Step by Step Instructions is a detailed guide on how to run the experiments inside a Docker container running the provided Docker image.
3. The section Inspecting the Source Files highlights some relevant source files with our WasmFX additions.
4. The section The WasmFX Toolchains describes how our "toolchains" work.
5. The section Reference Machine Specification contains some detailed information about the reference machine used to conduct the experiments.

`https://github.com/wasmfx/oopsla23-artifx`

WasmFX: Current status

What we got so far

- Formal specification (`https://github.com/WebAssembly/stack-switching/blob/main/proposals/continuations/Overview.md`)
- Informal explainer document (`https://github.com/WebAssembly/stack-switching/blob/main/proposals/continuations/Explainer.md`)
- Reference implementation (`https://github.com/effect-handlers/wasm-spec`)
- Research prototype implementation in Wasmtime (`https://github.com/effect-handlers/wasmtime`)

This project is known as WasmFX.

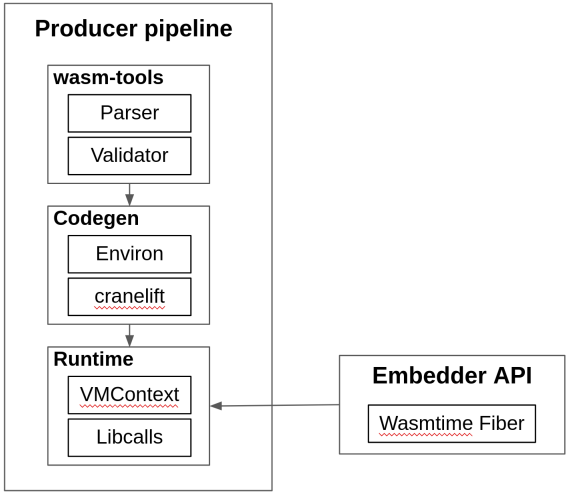
WasmFX: Current status

What we got so far

- Formal specification (`https://github.com/WebAssembly/stack-switching/blob/main/proposals/continuations/Overview.md`)
- Informal explainer document (`https://github.com/WebAssembly/stack-switching/blob/main/proposals/continuations/Explainer.md`)
- Reference implementation (`https://github.com/effect-handlers/wasm-spec`)
- **Research prototype implementation in Wasmtime** (`https://github.com/effect-handlers/wasmtime`)

This project is known as WasmFX.

Wasmtime overview



Wasmtime fiber interface

The essence of the Wasmtime fiber interface in Rust

```
trait FiberStack {
  fn new(size: usize) -> io::Result<Self>
}

trait<Resume, Yield, Return> Fiber<Resume, Yield, Return> {
  fn new(stack: FiberStack,
         func: FnOnce(Resume, &Suspend<Resume, Yield, Return>) -> Return
  fn resume(&self, val: Resume) -> Result<Return, Yield>
}

trait Suspend<Resume, Yield, Return> {
  fn suspend(&self, Yield) -> Resume
}
```

Wasmtime fiber interface

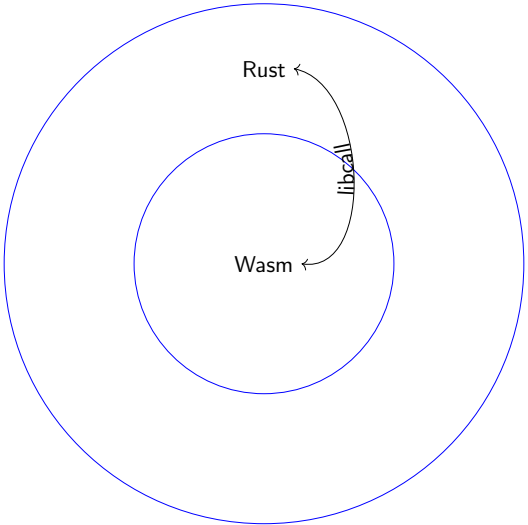
The essence of the Wasmtime fiber interface in Rust

```
trait FiberStack {
  fn new(size: usize) -> io::Result<Self>
  fn malloc(size: usize) -> io::Result<Self>
}

trait<Resume, Yield, Return> Fiber<Resume, Yield, Return> {
  fn new(stack: FiberStack,
         func: FnOnce(Resume, &Suspend<Resume, Yield, Return>) -> Return
  fn resume(&self, val: Resume) -> Result<Return, Yield>
}

trait Suspend<Resume, Yield, Return> {
  fn suspend(&self, Yield) -> Resume
}
```

Jumping between worlds



Implications of jumping between Wasm and Rust

Libcalling

- Function preamble sets up a trampoline
- Boxing required for $|\mathrm{payloads}| > 1$
- Everything is coded against an universal type

The instruction set extension

Types

- **cont** $\$ft$

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*)

Instructions

- **cont.new**
- **resume**
- **cont.bind**
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*)

Instructions

- **cont.new**
- **resume**
- **cont.bind**
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new**
- **resume**
- **cont.bind**
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume**
- **cont.bind**
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume** ✓
- **cont.bind**
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume** ✓
- **cont.bind** ✓
- **suspend**
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume** ✓
- **cont.bind** ✓
- **suspend** ✓
- **resume_throw**
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume** ✓
- **cont.bind** ✓
- **suspend** ✓
- **resume_throw** ✗
- **barrier**

The instruction set extension

Types

- **cont** $\$ft$ ✓

Tags

- **tag** $\$tag$ (**param** σ^*) (**result** τ^*) ✓

Instructions

- **cont.new** ✓
- **resume** ✓
- **cont.bind** ✓
- **suspend** ✓
- **resume_throw** ✗
- **barrier** ✗

Type section extension

Continuation type

$$(\textbf{cont}\ \$ft)$$

cont is a new reference type constructor parameterised by a function type, $\$ft : [\sigma^*] \to [\tau^*]$

Tag section extension

Control tag declaration

$$(\textbf{tag}\ \$\mathit{tag}\ (\textbf{param}\ \sigma^*)\ (\textbf{result}\ \tau^*))$$

it's a mild extension of the exception handling proposal's tag

Instruction extension (1)

Continuation allocation

$$\texttt{cont.new} : [(\texttt{ref null}\ \$ft)] \to [(\texttt{ref}\ \$ct)]$$

where $\$ft : [\sigma^*] \to [\tau^*]$
and $\$ct : \texttt{cont}\ \$ft$

Instruction extension (1)

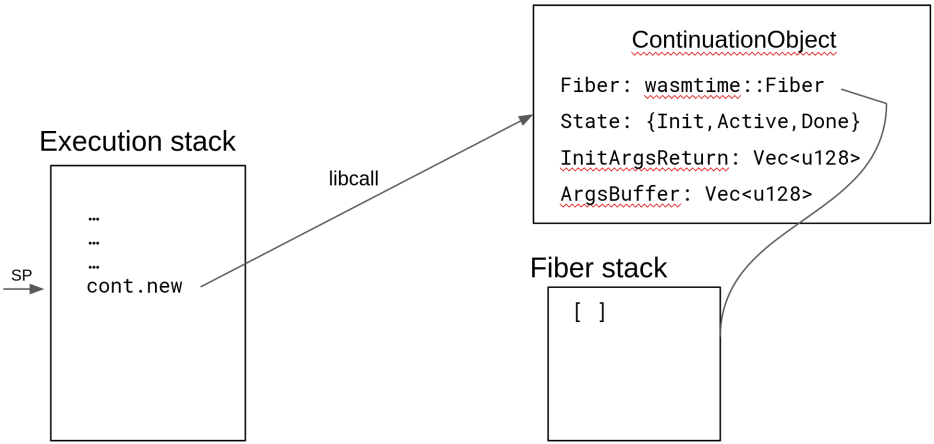
Continuation allocation

$$\textbf{cont.new}\ \$ct : [(\textbf{ref null}\ \$ft)] \to [(\textbf{ref}\ \$ct)]$$

where $\$ft : [\sigma^*] \to [\tau^*]$
and $\$ct : \textbf{cont}\ \$ft$

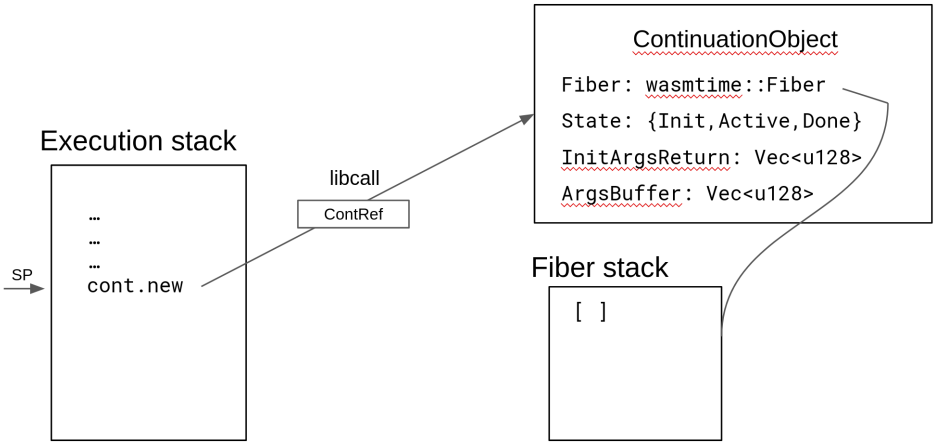
Spec change: Continuation type annotation on **cont.new**.

Implementation of **cont.new**



Number of libcalls: 1

Implementation of **cont.new**



Number of libcalls: 1

Instruction extension (2)

Continuation resumption

$$\textbf{resume}\ (\textbf{tag}\ \$tag\ \$h)^*\ :\ [\sigma^*\ (\textbf{ref null}\ \$ct)] \to [\tau^*]$$

where $\{\$tag_i : [\sigma_i^*] \to [\tau_i^*]$ and $\$h_i : [\sigma_i^*\ (\textbf{ref null}\ \$ct_i)]$ and
$\$ct_i : \textbf{cont}\ \$ft_i$ and $\$ft_i : [\tau_i^*] \to [\tau^*]\}_i$
and $\$ct : \textbf{cont}\ \$ft$
and $\$ft : [\sigma^*] \to [\tau^*]$

The instruction fully consumes the continuation argument.

Instruction extension (2)

Continuation resumption

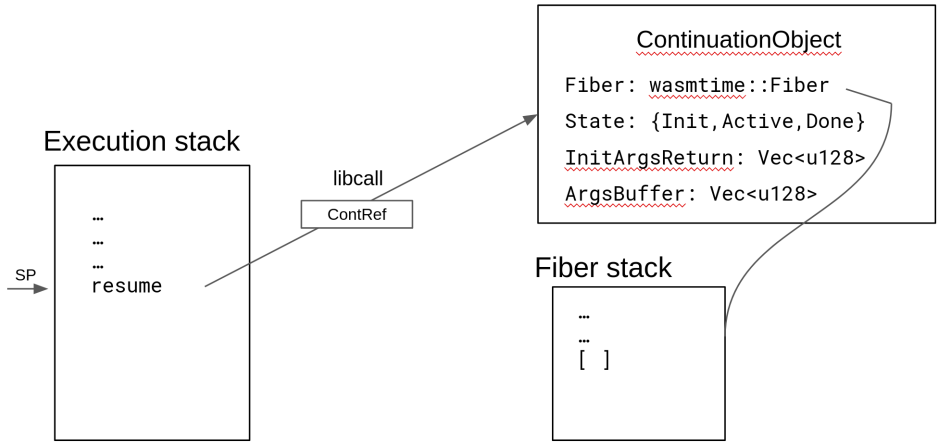
$$\textbf{resume}\ \$ct\ (\textbf{tag}\ \$tag\ \$h)^* : [\sigma^*\ (\textbf{ref null}\ \$ct)] \to [\tau^*]$$

where $\{\$tag_i : [\sigma_i^*] \to [\tau_i^*]$ and $\$h_i : [\sigma_i^*\ (\textbf{ref null}\ \$ct_i)]$ and
$\$ct_i : \textbf{cont}\ \$ft_i$ and $\$ft_i : [\tau_i^*] \to [\tau^*]\}_i$
and $\$ct : \textbf{cont}\ \$ft$
and $\$ft : [\sigma^*] \to [\tau^*]$

The instruction fully consumes the continuation argument.

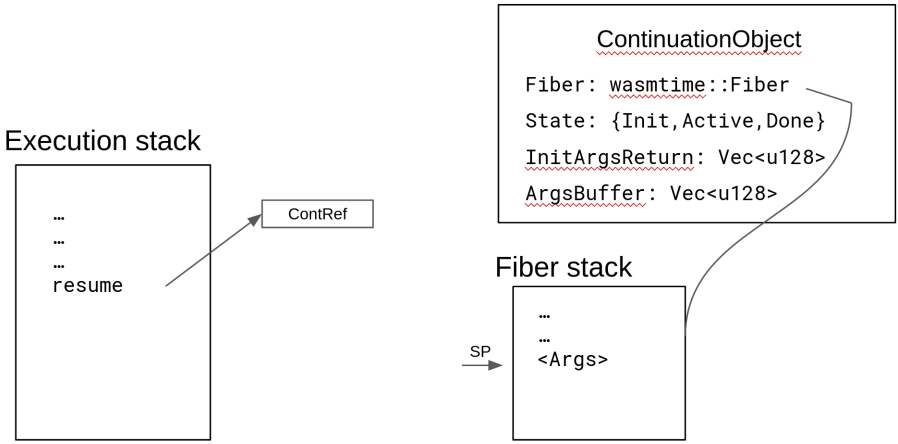
Spec change: Type annotation on **resume**

Implementation of **resume**



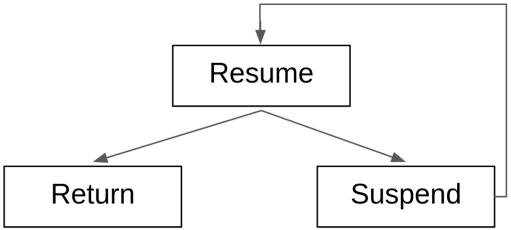
Number of libcalls: 4

Implementation of **resume**



Number of libcalls: 4

Basic block encoding of **`resume`**



Instruction extension (3)

Partial continuation application

$$\texttt{cont.bind}\ (\texttt{type}\ \$ct) : [\sigma_0^*\ (\texttt{ref null}\ \$ct)] \to [(\texttt{ref}\ \$ct')]$$

where $\$ct$: **cont** $\$ft$ and $\$ft$: $[\sigma_0^*\ \sigma_1^*] \to [\tau^*]$
and $\$ct'$: **cont** $\$ft'$ and $\$ft'$: $[\sigma_1^*] \to [\tau^*]$

Instruction extension (3)

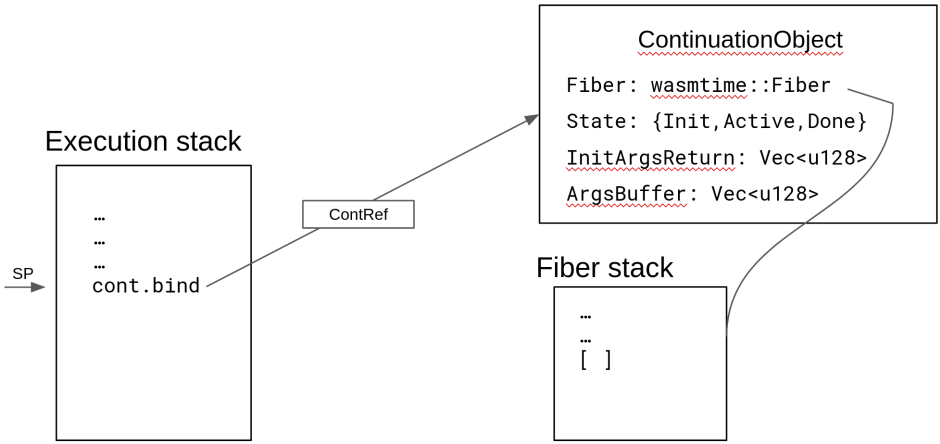
Partial continuation application

$$\texttt{cont.bind}\ \$sct\ \$dct : [\sigma_0^*\ (\texttt{ref null}\ \$sct)] \to [(\texttt{ref}\ \$dct)]$$

where $\$sct : \texttt{cont}\ \$ft$ and $\$ft : [\sigma_0^*\ \sigma_1^*] \to [\tau^*]$
and $\$dst : \texttt{cont}\ \$ft'$ and $\$ft' : [\sigma_1^*] \to [\tau^*]$

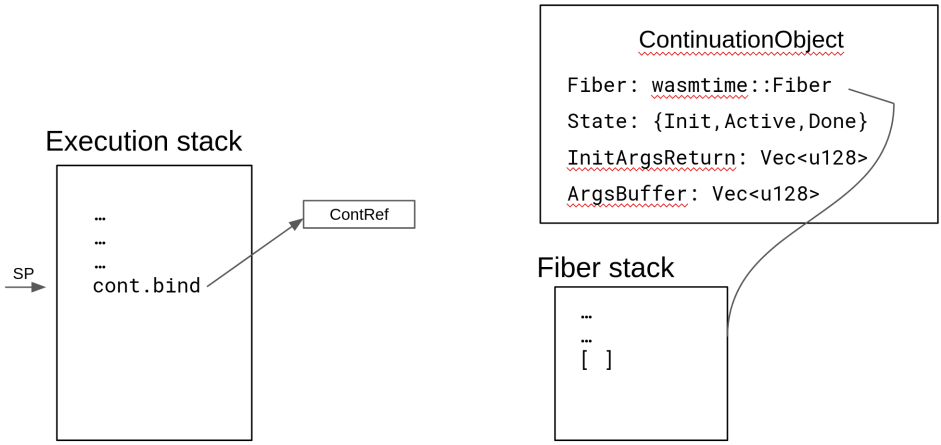
Spec change: **`cont.bind`** is annotated with both input and output continuation type.

Implementation of **cont.bind**



Number of libcalls: 3

Implementation of **cont.bind**



Number of libcalls: 3

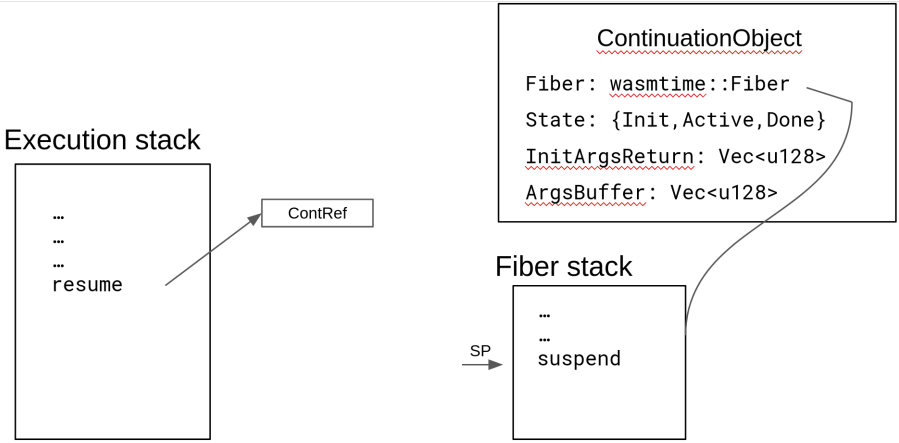
Instruction extension (4)

Continuation suspension

suspend $\$tag : [\sigma^*] \to [\tau^*]$

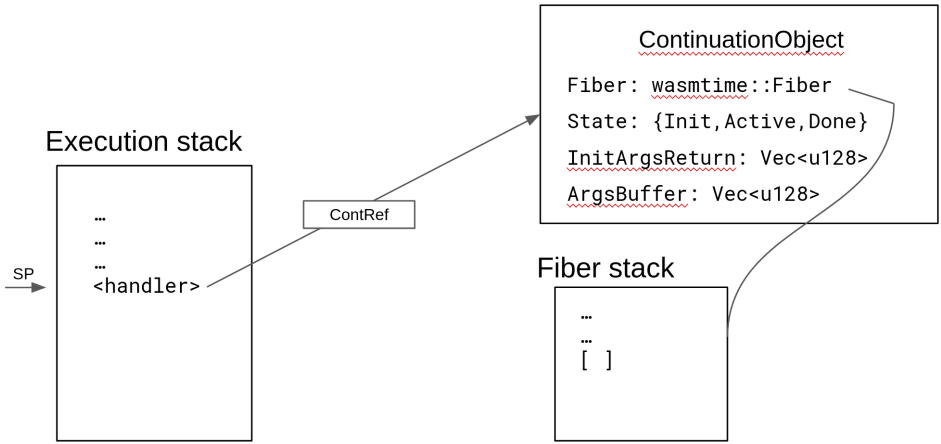
where $\$tag : [\sigma^*] \to [\tau^*]$

Implementation of **suspend**



Number of libcalls: 3

Implementation of **suspend**



Number of libcalls: 3

Instruction extension (5)

Continuation cancellation

$$\textbf{resume\_throw}\ (\textbf{tag}\ \$exn)\ (\textbf{tag}\ \$tag\ \$h)^* : [\sigma_0^*\ (\textbf{ref null}\ \$ct)] \to [\tau^*]$$

where $\$exn : [\sigma_0^*] \to []$, $\{\$tag_i : [\sigma_i^*] \to [\tau_i^*]$ and $\$h_i : [\sigma_i^*\ (\textbf{ref null}\ \$ct_i)]$ and $\$ct_i : \textbf{cont}\ \$ft_i$ and $\$ft_i : [\tau_i^*] \to [\tau^*]\}_i$

and $\$ct : \textbf{cont}\ ([\sigma^*] \to [\tau^*])$

Instruction extension (5)

Continuation cancellation

$$\textbf{resume\_throw}\ \$ct\ (\textbf{tag}\ \$exn)\ (\textbf{tag}\ \$tag\ \$h)^* : [\sigma_0^*\ (\textbf{ref null}\ \$ct)] \to [\tau^*]$$

where $\$exn : [\sigma_0^*] \to []$, $\{\$tag_i : [\sigma_i^*] \to [\tau_i^*]$ and $\$h_i : [\sigma_i^*\ (\textbf{ref null}\ \$ct_i)]$ and $\$ct_i : \textbf{cont}\ \$ft_i$ and $\$ft_i : [\tau_i^*] \to [\tau^*]\}_i$

and $\$ct : \textbf{cont}\ ([\sigma^*] \to [\tau^*])$

Spec change: **resume_throw** is annotated with the type of the continuation.

Instruction extension (6)

Control barriers

$$\textbf{barrier}\ \$lbl\ (\textbf{type}\ \$bt)\ instr^* : [\sigma^*] \to [\tau^*]$$

where $\$bt = [\sigma^*] \to [\tau^*]$ and $instr^* : [\sigma^*] \to [\tau^*]$

Instruction extension (6)

Control barriers

$$\textbf{barrier}\ \$lbl\ \$bt\ instr^* : [\sigma^*] \to [\tau^*]$$

where $\$bt = [\sigma^*] \to [\tau^*]$ and $instr^* : [\sigma^*] \to [\tau^*]$

Spec change: simplify syntax.

Preliminary experiments (1)

Microbenchmark

- Create 10000000 coroutines
- Each coroutine yields once
- 10000 coroutines are ready to run, the rest are blocked

Preliminary experiments (1)

Microbenchmark

- Create 10000000 coroutines
- Each coroutine yields once
- 10000 coroutines are ready to run, the rest are blocked

	Run-time ratio	Memory footprint ratio
Asyncify	1.0	1.0 (63mb)
WasmFX	0.2	1.0 (63mb)

Preliminary experiments (1)

Microbenchmark

- Create 10000000 coroutines
- Each coroutine yields once
- 10000 coroutines are ready to run, the rest are blocked

	Run-time ratio	Memory footprint ratio
Asyncify	1.0	1.0 (66mb)
WasmFX	0.28	1.05 (63mb)

earlier version

Preliminary experiments (1)

Microbenchmark

- Create 10000000 coroutines
- Each coroutine yields once
- 10000 coroutines are ready to run, the rest are blocked

	Run-time ratio	Memory footprint ratio
Asyncify	1.0	1.0 (66mb)
WasmFX	0.28	1.05 (63mb)

earlier version

Caution: the implementations are backed by different data structures.

Preliminary experiments (2)

Wasm binary size microbenchmarks

- Patched TinyGo to emit WasmFX
- Compile programs from the TinyGo repository

Preliminary experiments (2)

Wasm binary size microbenchmarks

- Patched TinyGo to emit WasmFX
- Compile programs from the TinyGo repository

	main-kjp.go	coroutines.go
Asyncify	597 KB	40 KB
WasmFX	156 KB	7.2 KB

Preliminary experiments (2)

Wasm binary size microbenchmarks

- Patched TinyGo to emit WasmFX
- Compile programs from the TinyGo repository

	main-kjp.go	coroutines.go
Asyncify	597 KB	40 KB
WasmFX	156 KB (26.13%)	7.2 KB (18%)

Next steps

Benchmarks

- Microbenchmarking (e.g. Sieve of Eratosthenes)
- Macrobenchmarking (e.g. HTTP/2-compliant webserver)

Future experiments (1)

Backends

- libmprompt
- Internalise Wasmtime Fiber in codegen
- Cranelift native stack switching

Memory

- Deferred stack allocation
- Stack pools
- Novel allocation schemes

Extensions

- Named resume blocks
- First-class & generative control tags

Future experiments (2)

Toolchain support

- Compiling control abstractions
- Retrofitting existing toolchains
- Develop new (researchy) toolchains

Future experiments (2)

Toolchain support

- Compiling control abstractions
- Retrofitting existing toolchains
- Develop new (researchy) toolchains

An open invitation

We'd like to work with parties interested in exploring compilation to the WasmFX instruction set.

Summary

Summary

- A working prototype implementation of WasmFX in wasmtime
- Implementation feedback led to minor spec changes
- Next: focus on building benchmarks
- Next next: focus on performance

`https://wasmfx.dev`

References

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