RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳

用 Rust 设计高性能 JIT 执行引擎

周鹤洋 @ Rust China Conf 2020

关于我

- GitHub @losfair
- 南京航空航天大学 2018 级本科生
- 编译 / OS / VM / 微架构
- 从 2017 年开始使用 Rust

Projects that I worked on

- Wasmer / 跨平台 WebAssembly Runtime
- kernel-wasm / Linux 内核态 WebAssembly SFI 运行环境
- Violet / 函数式设计的超标量 RISC-V CPU
- VMesh / 去中心化 Mesh 路由协议
- Wavelet / DAG 分布式账本
- FlatMk / Capability-based microkernel in Rust
- … and a lot of toy projects

Let's begin

Just-In-Time compilation

- “ 即时编译”
- 运行时生成代码并执行

Where are JITs used?

- 虚拟机
- 动态链接器
- Linux 内核

Virtual machines

- JVM / CLR / V8 / Wasmer / Wasmtime / CPython etc.

Why?

目标语言规范与硬件规范不一致时，需要使用虚拟机（VM）运行

例如：

- 动态特性
- 硬件层面难以实现的沙盒特性（ SFI ）
- 不同的指令集（二进制翻译)

VM 技术

- 简单解释器
  - wasmi
- 优化的解释器
  - CPython, BEAM, wasm3
  - Threaded interpreter, fused operations, inline caching
- 即时编译 / Just-In-Time (JIT) Compilation
  - JVM, CLR, V8, LuaJIT, Wasmer, Wasmtime

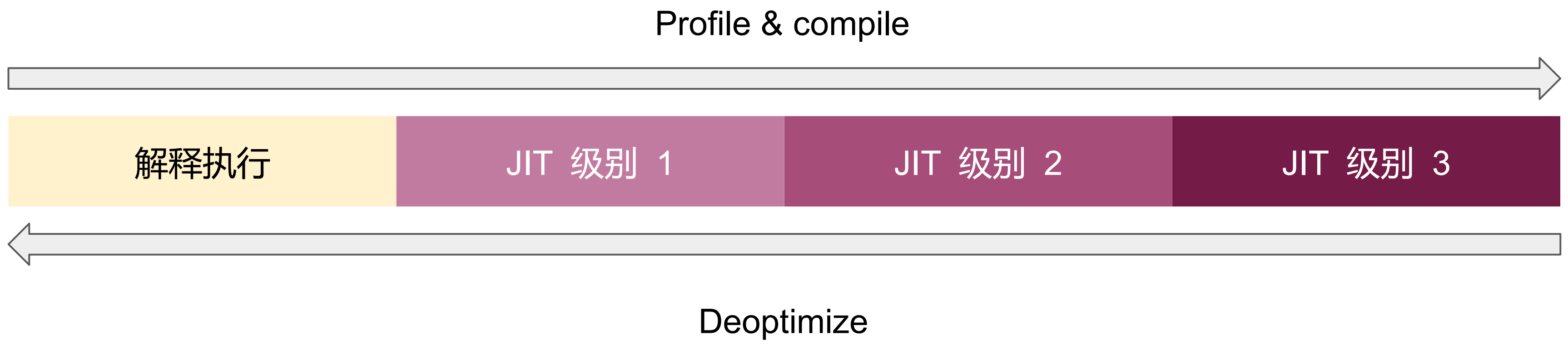
Why do we need JITs?

- 动态优化代码
- 高效支持语言的动态特性 / 安全性要求
- 初始化（动态链接器 & Static call 机制）

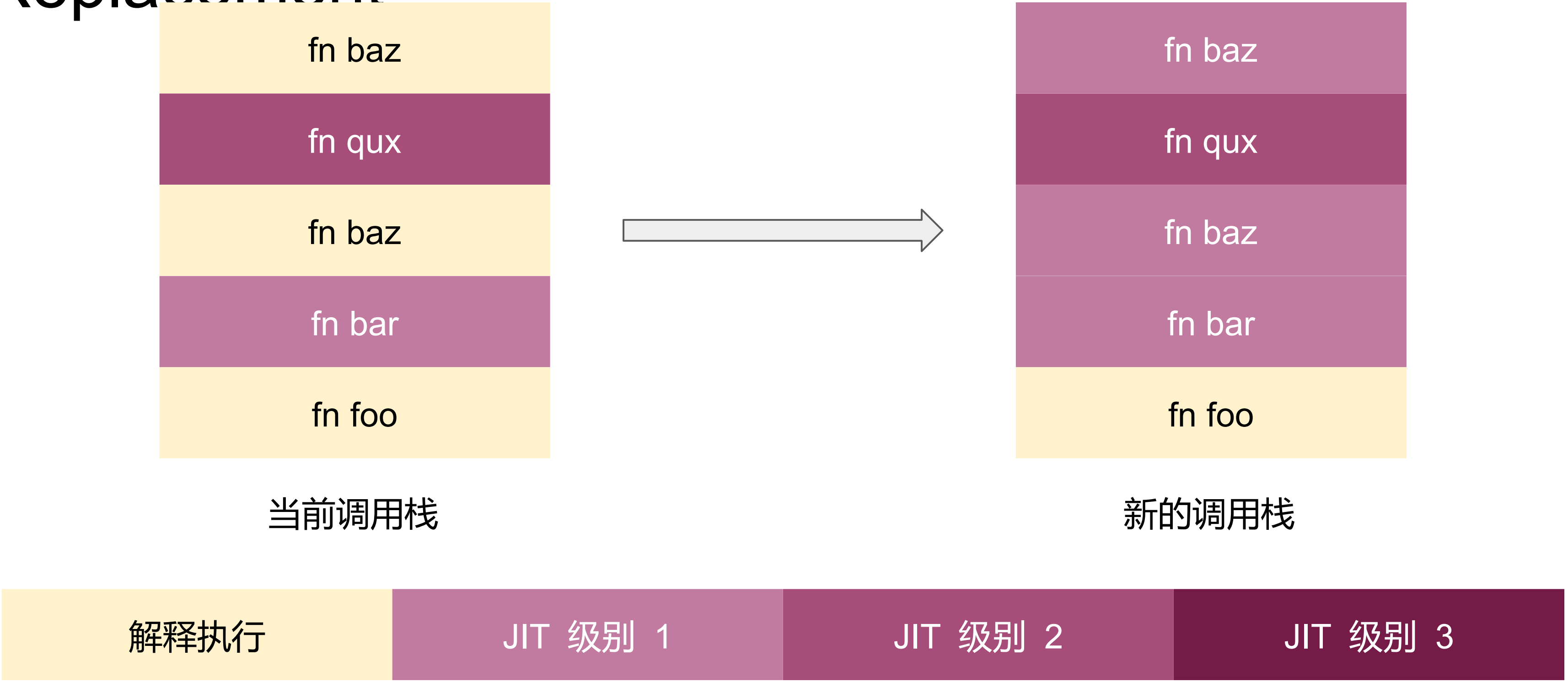
Dynamic optimization

JIT 优化级别

JavaScriptCore	LLInt	Baseline JIT	DFG JIT	FTL JIT
V8	Ignition	-	-	TurboFan
Wasmer	-	Singlepass	Cranelift	LLVM
	解释执行	JIT 级别 1	JIT 级别 2	JIT 级别 3


Profile & compile
解释执行
JIT 级别 1
JIT 级别 2
JIT 级别 3
Deoptimize

切换优化级别：栈上替换 / On-Stack Replacement



OSR in Wasmer

Su Engine @ https://github.com/wasmerio/wasmer/pull/489

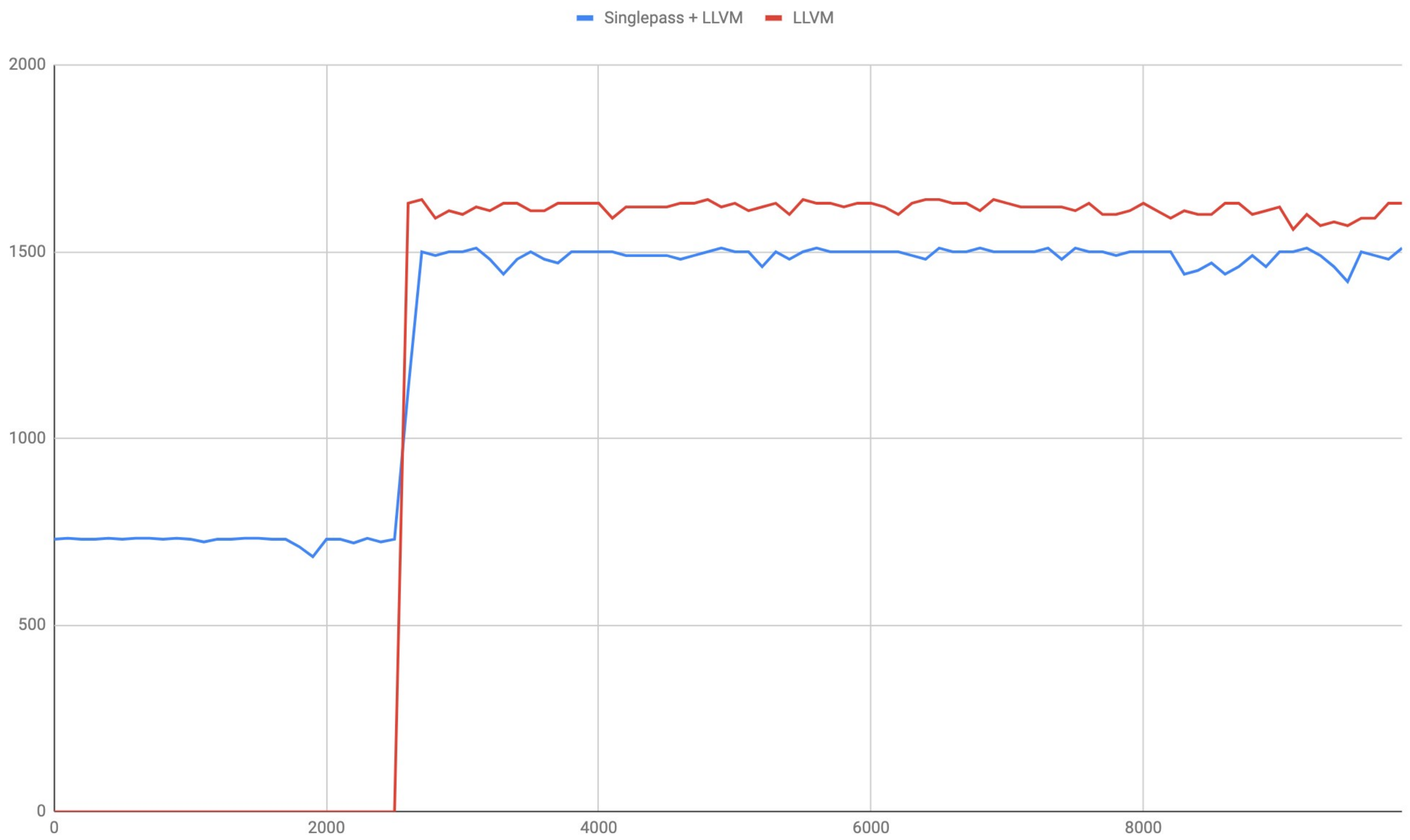
My work!

```
pub unsafe fn read_stack(
    msm: &ModuleStateMap,
    code_base: usize,
    mut stack: *const u64,
    initially_known_registers: [Option<u64>; 24],
    mut initial_address: Option<u64>,
) -> ExecutionStateImage {
```

OSR exit

```
pub unsafe fn invoke_call_return_on_stack(
    msm: &ModuleStateMap,
    code_base: usize,
    image: &InstanceImage,
    vmctx: &mut Ctx,
) -> u64 {
```

OSR entry



Performance / Time

内联缓存

JavaScript

```
for(let x of list) {
    document.write(x); // Method lookup
}
```

RISC-V 二进制翻译

```
   11040:   f9843603          ld    a2,-104(s0) # MMU lookup
   11044:   00078593          mv    a1,a5
   11048:   00070513          mv    a0,a4
   1104c:   000680e7          jalr  a3 # MMU lookup & translation
lookup
   11050:   00050793          mv    a5,a0
   11054:   02f04463          bgtz  a5,1107c
<core_list_mergesort+0x168>
   11058:   fe843783          ld    a5,-24(s0) # MMU lookup
```

rvjit-aa64：从内存加载数据（内联缓存快速路径）

```
// Compute effective address
ld_simm16(&mut self.a, trash_reg_w() as _, imm);
dynasm!(self.a
    ; .arch aarch64
    ; add X(trash_reg_w() as u32), X(trash_reg_w() as u32), X(rs as u32)
);

let lower_bound_slot = self.alloc_rtslot(std::u64::MAX);
let upper_bound_slot = self.alloc_rtslot(0);
let reloff_slot = self.alloc_rtslot(std::u64::MAX);

self.load_rtslot_pair(lower_bound_slot, 30, temp1_reg());

// Check bounds
dynasm!(self.a
    ; .arch aarch64

    // Lower bound
    ; cmp X(trash_reg_w() as u32), x30
    ; b.lo >fallback

    // Upper bound
    ; cmp X(trash_reg_w() as u32), X(temp1_reg())
    ; b.hs >fallback
);
```

rvjit-aa64：从内存加载数据（慢速路径）

```
let addr = self.read_register(pp.rs as _) + (pp.rs_offset as i64 as u64);
debug!("load/store miss. target = 0x{:016x}", addr);

let (target_base_v, target_section) = match registry.lookup_section(addr) {
    Ok(x) => x,
    Err(_) => return Err(ExecError::BadLoadStoreAddress),
};

if pp.is_store && !target_section.get_flags().contains(SectionFlags::W) {
    return Err(ExecError::BadLoadStoreFlags);
}

if !pp.is_store && !target_section.get_flags().contains(SectionFlags::R) {
    return Err(ExecError::BadLoadStoreFlags);
}

let target_base_real = target_section.get().as_ptr() as u64;
let target_len = target_section.get().len() as u64;
```

Memory safety

JIT 内存机制

- 空指针检查
- 访问越界检查

简单的实现方法： compare-and-branch

如何减少开销?

```
cmp $0, %rdi
je null_pointer_exception
mov %rdi, 16(%rsp)
...
```

```
null_pointer_exception:
    call host_npe_handler
    ...
```

```
mov %rdi, 16(%rsp)
...
```

Trap!

```
sigsegv_handler:
    ...
    call host_npe_handler
    ...
```

采用 Unix 信号处理同步异常

```rust
// Allow handling OOB with signals on all architectures
register(&mut PREV_SIGSEGV, libc::SIGSEGV);

// Handle `unreachable` instructions which execute `ud2` right now
register(&mut PREV_SIGILL, libc::SIGILL);

// x86 uses SIGFPE to report division by zero
if cfg!(target_arch = "x86") || cfg!(target_arch = "x86_64") {
    register(&mut PREV_SIGFPE, libc::SIGFPE);
}

// On ARM, handle Unaligned Accesses.
// On Darwin, guard page accesses are raised as SIGBUS.
if cfg!(target_arch = "arm") || cfg!(target_os = "macos") {
    register(&mut PREV_SIGBUS, libc::SIGBUS);
}
```

JIT in Linux kernel

JIT in Linux kernel: eBPF

- 允许用户代码安全介入内核的机制
- Interpreter + JIT

JIT in Linux kernel: Static call (Linux 5.10)

- 动态生成直接跳转指令序列替代间接跳转
- 比 Retpoline 更高效的 Spectre v2 缓解机制

```c
static void __ref __static_call_transform(void *insn, enum insn_type type, void *func)
{
	int size = CALL_INSN_SIZE;
	const void *code;

	switch (type) {
	case CALL:
		code = text_gen_insn(CALL_INSN_OPCODE, insn, func);
		break;

	case NOP:
		code = ideal_nops[NOP_ATOMIC5];
		break;

	case JMP:
		code = text_gen_insn(JMP32_INSN_OPCODE, insn, func);
		break;

	case RET:
		code = text_gen_insn(RET_INSN_OPCODE, insn, func);
		size = RET_INSN_SIZE;
		break;
	}

	if (memcmp(insn, code, size) == 0)
		return;

	if (unlikely(system_state == SYSTEM_BOOTING))
		return text_poke_early(insn, code, size);

	text_poke_bp(insn, code, size, NULL);
}
```

JIT in Linux kernel: Static call (Linux 5.10)

```
some_func:
    jmp *%rax
    ...
```

Original

```
some_func:
    call L2
L1:
    lfence
    jmp L1
L2:
    mov %rax, (%rsp)
    ret
    ...
```

Retpoline

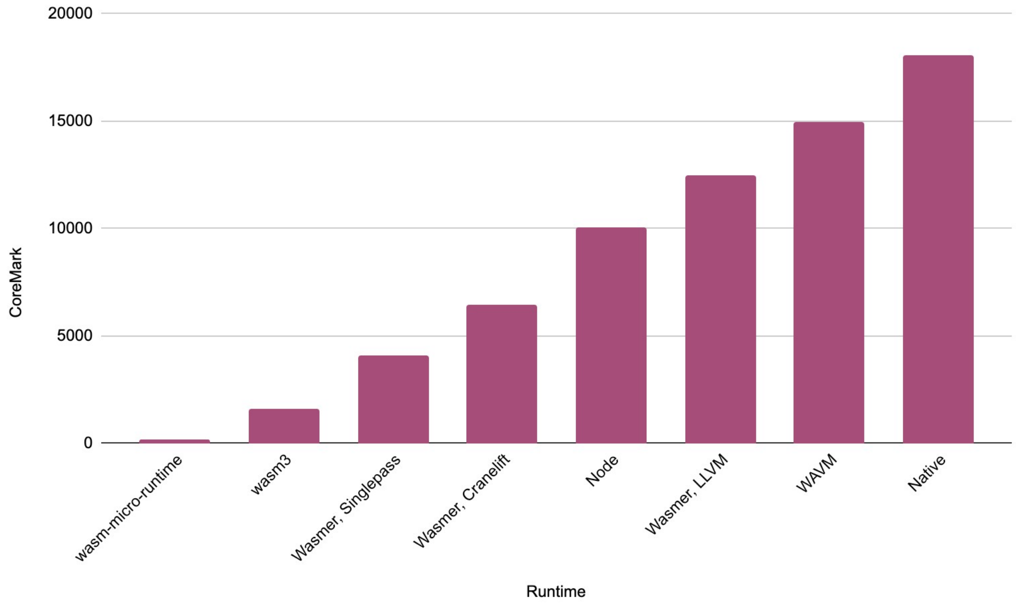
```
some_func:
    call _trampoline
    ...

_trampoline: # Rewritten at runtime
    .byte 0xe9 # Relative JMP
    .dword (target - _trampoline)
```

Static call

Should I use a JIT?

执行效率 & 工程复杂性

- 执行效率： JIT >>> 优化的解释器 >> 简单解释器
- 工程复杂性： JIT >>> 优化的解释器 > 简单解释器



Complexity -> Bugs

LPE bugs discovered in Linux eBPF JIT:

- CVE-2020-8835
- CVE-2020-27194

Implementing JIT in Rust

Thanks!

RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳