FEX-Emu 模块解析

徐泽逸 xuzeyi@stu.ecnu.edu.cn

TL;DR

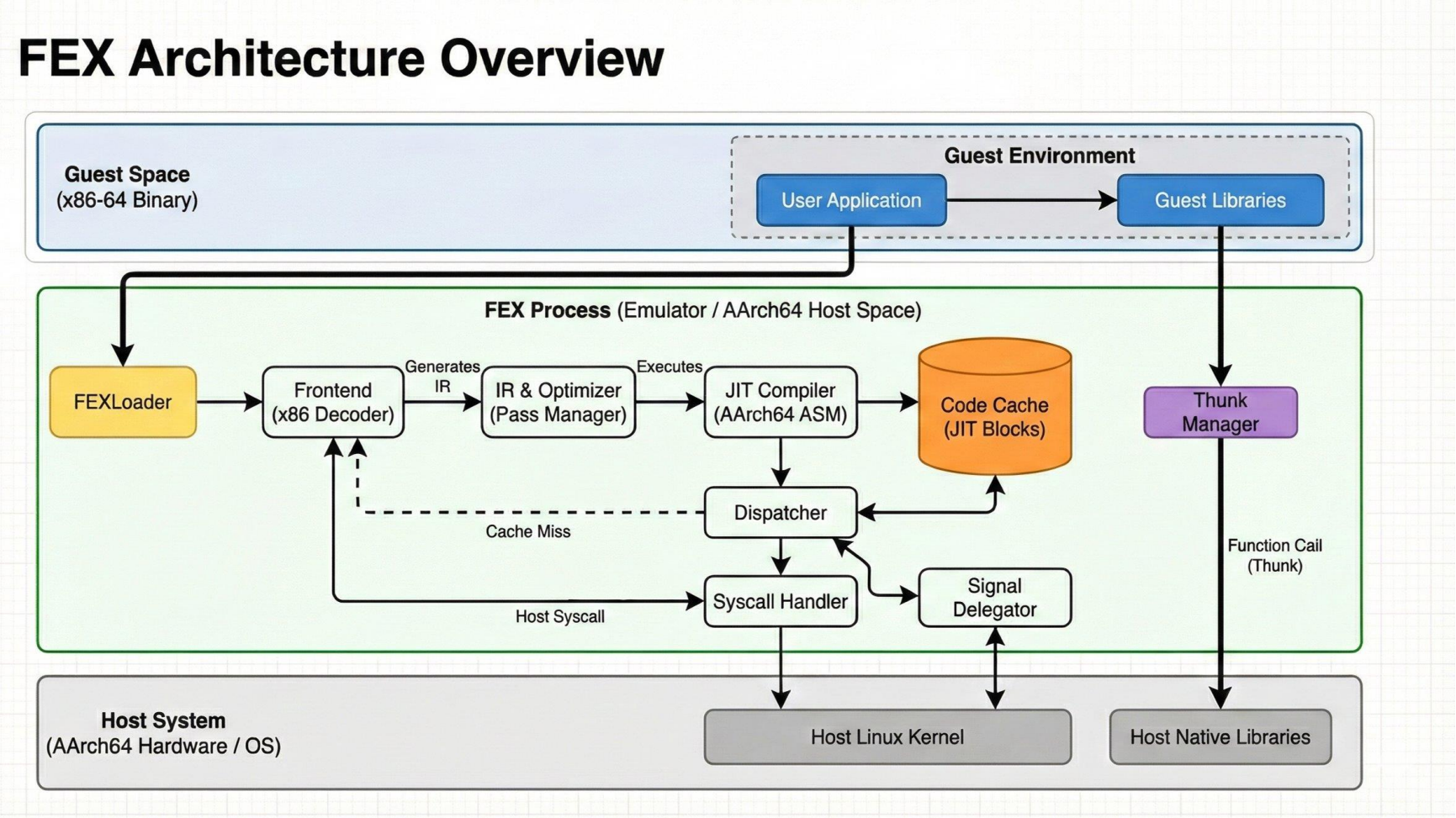
- 两个部分
  - FEX 源码的一些分享
  - RISC-V 的目前方案与可能会遇见的问题

Table Of Contents

- Basic Info
- 前端:
- 中端: IR 与 SSA 形式下的优化
- 后端: Syscall处理/代码生成
- 测试
- RISC-V 移植

Basic Info

- FEX-Emu 是一个从 x86(_64) 到 ARM 的动态二进制翻译工具



Why use FEX

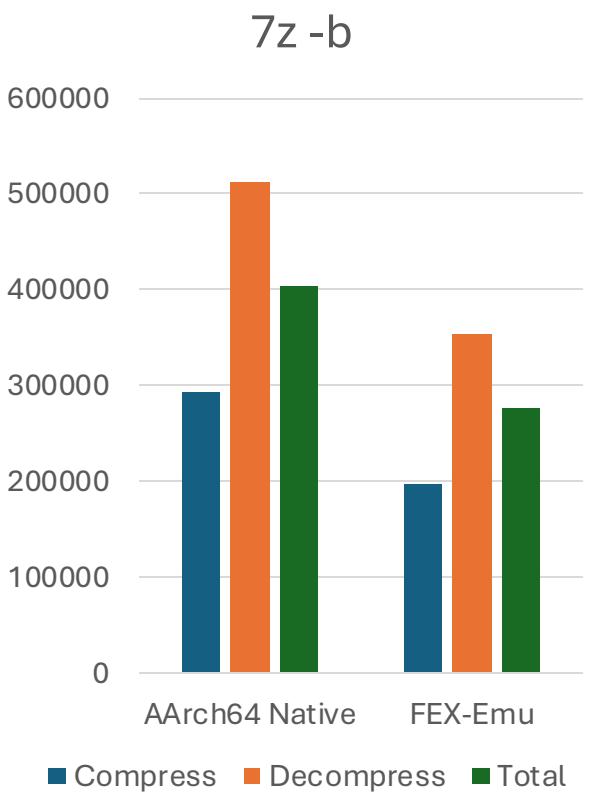
- 与传统编译器更接近的架构
  - 前端 -> IR -> Codegen
  - 比起其它 DBT 更清晰的设计

- 可接受的性能表现 (next slide)

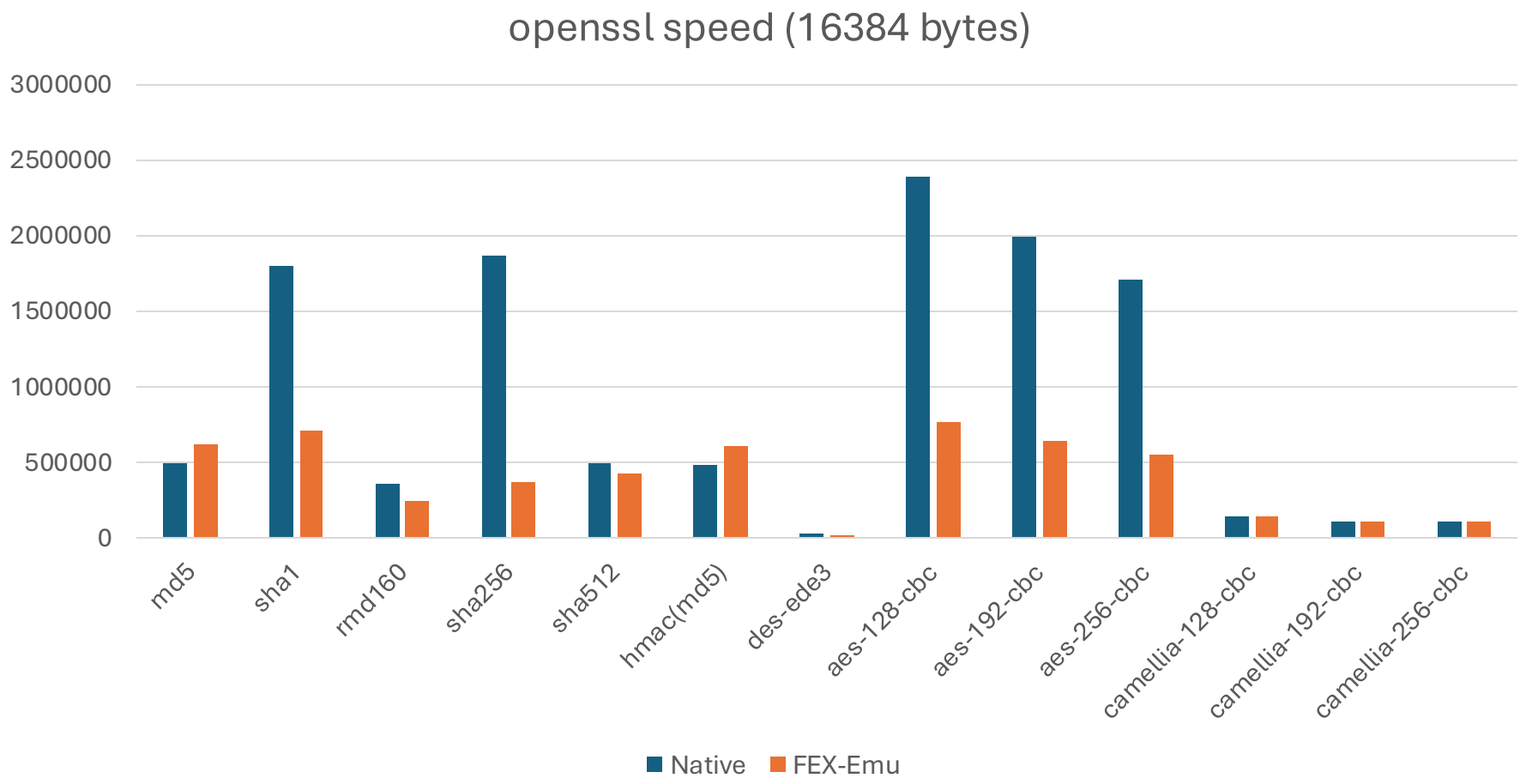
一个简单的 Benchmark

- 测试对象: FEX-Emu/Native
- 测试硬件: ampere@solelab
- 测试集: openssl/7z

Bench results:



Around 75% of native



前端: 字节读取

- FEX 在 `FEXCore::Frontend::Decoder` 中读取并解析程序原始的 x86 字节流 (`FEXCore/Source/Interface/Core/Frontend.cpp`)

```
6  uint8_t Decoder::ReadByte() {
7    LOGMAN_THROW_A_FMT(InstructionSize < MAX_INST_SIZE, "Max instruction size exceeded!");
8    std::optional<uint8_t> Byte = PeekByte(0);
9    if (!Byte) {
10     HitNonExecutableRange = true;
11     // Pretend we read 0, the main decode loop will see HitNonExecutableRange and rollback the instruction.
12     return 0;
13   }
14
15   Instruction[InstructionSize] = *Byte;
16   InstructionSize++;
17   return *Byte;
18 }
```

越界检查

PeekByte 内部检查内存是否可执行

存入指令缓存并返回

前端: OpCode 映射

FEX 定义了一个巨大的静态数组 `OpDispatch_BaseOpTable` (`FEXCore/Source/Interface/Core/OpcodeDispatcher/BaseTables.h`)

Decoder 读到字节后，会去查表拿到对应的函数指针

```
 4 namespace FEXCore::IR {
 5 constexpr inline DispatchTableEntry OpDispatch_BaseOpTable[] = {
 6   // Instructions
 7   {0x00, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::ALUOp, FEXCore::IR::IROps::OP_ADD, FEXCore::IR::IROps::OP_ATOMICFETCHADD, 0>},
 8
 9   {0x08, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::ALUOp, FEXCore::IR::IROps::OP_OR, FEXCore::IR::IROps::OP_ATOMICFETCHOR, 0>},
10
11   {0x10, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::ADCOp, 0>},
12
13   {0x18, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::SBBOp, 0>},
14
15   {0x20, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::ALUOp, FEXCore::IR::IROps::OP_ANDWITHFLAGS, FEXCore::IR::IROps::OP_ATOMICFETCHAND, 0>},
16
17   {0x28, 6, &OpDispatchBuilder::Bind<&OpDispatchBuilder::ALUOp, FEXCore::IR::IROps::OP_SUB, FEXCore::IR::IROps::OP_ATOMICFETCHSUB, 0>},
18
```

前端: IR 构造

- 相关文件:
  - /FEXCore/Source/Interface/IR/IREmitter.[h|cpp]
  - /FEXCore/Source/Interface/Core/OpcodeDispatcher.cpp
- 流程:
  - 加载源操作数
  - 加载目的操作数
  - 生成 IR
  - 计算标志位

中端: Overview

- IR Spec: IR.json
- IR Dumper: IRDumper.cpp
- IR Debug: IRDumperPass.cpp
- IR Emitter: IREmitter.cpp
  - 提供 IR 的构造函数
- IR to IR Optimizations:
  - 见 PassManager 与 Passes 文件夹下的文件

中端: IR

- FEX 的 IR 定义在 `/FEXCore/Source/Interface/IR/IR.json` 中
- 在构建时 CMake 会使用该 `.json` 定义生成三个文件:
  - `IR.md`: FEX-Emu 的 IR 文档
  - `IRDefines.inc`: IR 使用的所有结构体定义
  - `IRDefines_Dispatch.inc`: JIT 后端所使用的函数声明与 Alias

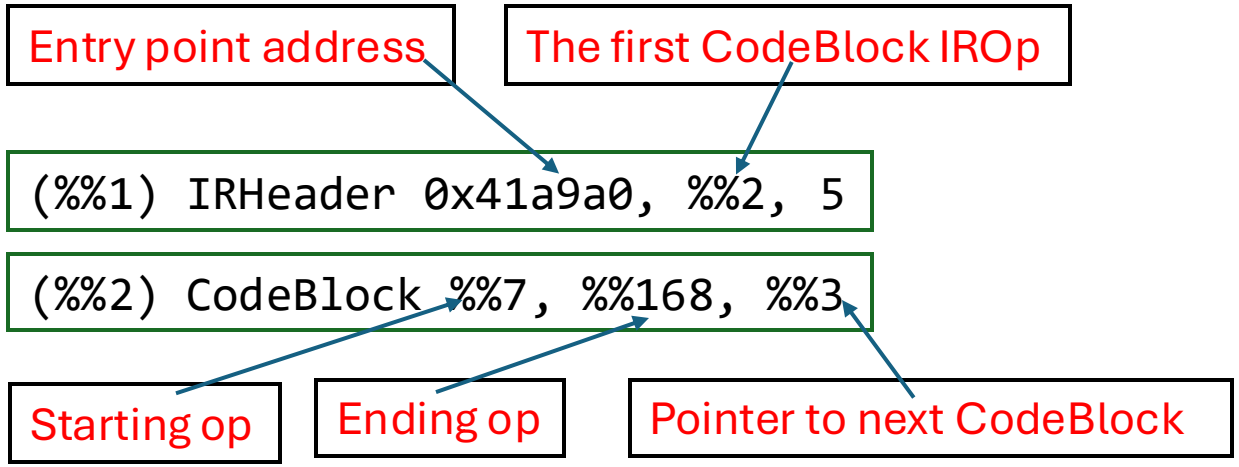
```
"GPR:$EAX, GPR:$EDX = XGetBV GPR:$Function": {
  "Desc": ["Calls in to the XCR handler function to return emulated XCR"],
  "DestSize": "OpSize::i32Bit",
  "HasSideEffects": true
}
```

```
struct __attribute__((packed)) IROp_XGetBV {
    IROp_Header Header;
    // SSA arguments
    OrderedNodeWrapper Function;
    OrderedNodeWrapper OutEAX;
    OrderedNodeWrapper OutEDX;
    static constexpr IROps OPCODE = OP_XGETBV;
    // Get index of argument by name
    static constexpr size_t Function_Index = 0;
    static constexpr size_t OutEAX_Index = 1;
    static constexpr size_t OutEDX_Index = 2;
};
static_assert(std::is_trivially_copyable_v<IROp_XGetBV>);
static_assert(std::is_standard_layout_v<IROp_XGetBV>);
```

OpCode Class	操作
ALU	算术逻辑运算，加/减/乘/除/位运算等
Atomic	原子操作，处理多线程/并发环境中的同步机制
Backend	只有一个特殊的 OpCode: Last，用来获取列表中的最后一个元素
Branch	SSA 基本块跳转，Syscall 转发/调用，Library forwarding，CPUID，断点...
Conv	(整数/浮点/向量)类型转换
Crypto	加密和哈希函数 (AES/SHA-256/SHA-1)
F64	double 相关的运算 (FEX 中主要是三角/对数/指数函数)
F80	模拟 x87 FPU 的 long double
Memory	与内存的交互 (加载/存储/地址计算等)
Misc	SSA 的元数据 (IRHeader/各种 Block)，舍入模式，随机数等
Moves	Copy && Swap
StaticRA	绕过 SSA 虚拟寄存器，直接读写物理寄存器
Vector	纯向量操作类指令
VectorScalar	标量-向量混合浮点运算

中端: IR

- FEX-Emu 的 IR 使用了 SSA 形式，并有以下特殊约定:
  - 第一个 SSA Node 被认为是 Invalid 的 (即 %0 永远为 null)
  - 同时第二个 SSA Node 一定是 IRHeader Node (即 %1 永远为 IRHeader)




```
(%%3) CodeBlock %%169, %%173, %%4
(%%169) BeginBlock %3
(%%170) i64 = Constant 0x41a9e1
(%%171) StoreContext %170 i64, 0x8, 0x0
(%%172) ExitFunction
(%%173) EndBlock %3
```

中端: IR (In-memory)

- **内存分配: 在** `IntrusiveIRList.h` **提供函数，将一段**连续内存一分为二
  - Data **区: 存** IROp (`IROp_Header + IROp_*`)
  - List **区: 存** OrderedNode (实际**的双向链表节点**，见 `IR.h`)

```
20 /**
19  * @brief This is a node in our IR representation
18  * Is a doubly linked list node that lives in a representation of a linearly allocated node list
17  * The links in the nodes can live in a list independent of the data IR data
16  *
15  * ex.
14  *  Region1 : ... <-> <OrderedNode> <-> <OrderedNode> <-> ...
13  *                    | *<Value>        |
12  *                    v                 v
11  *  Region2 : <IROp>..<IROp>..<IROp>..<IROp>
10  *
 9  *  In this example the OrderedNodes are allocated in one linear memory region (Not necessarily contiguous with one another linking)
 8  *  The second region is contiguous but they don't have any relationship with one another directly
 7  */
```

```
(%%3) CodeBlock %%169, %%173, %%4
```

Region2

```
3  struct __attribute__((packed)) IROp_CodeBlock {
4      IROp_Header Header;
5      // SSA arguments
6      OrderedNodeWrapper Begin;
7      OrderedNodeWrapper Last;
8      // Non-SSA arguments
9      uint32_t ID;
10     bool EntryPoint;
11     uint32_t GuestEntryOffset;
12     static constexpr IROps OPCODE = OP_CODEBLOCK;
13     // Get index of argument by name
14     static constexpr size_t Begin_Index = 0;
15     static constexpr size_t Last_Index = 1;
16 };
```

Region1: 链表

```
using OpNodeWrapper = NodeWrapperBase<IROp_Header>;
using OrderedNodeWrapper = NodeWrapperBase<OrderedNode>;
```

中端: 基于 SSA 的 IR 优化

FEX-Emu 的 PassManager 目前注册了这些 Pass:

- IR 验证: /FEXCore/Source/Interface/IR/Passes/IRValidation.cpp
- 寄存器分配: 基于线性扫描的简易 Allocator
  /FEXCore/Source/Interface/IR/Passes/RegisterAllocationPass.cpp
- Dead Flag/DCE 消除:
  /FEXCore/Source/Interface/IR/Passes/RedundantFlagCalculationElimination.cpp
- X87 浮点运算的 StackOptimizationPass

后端: CPU 能力探测

- 相关文件:
  - 接口
    - /FEXCore/include/FEXCore/Core/HostFeatures.h
  - 具体实现
    - /Source/Common/HostFeatures.h
    - /Source/Common/HostFeatures.cpp
  - 实现方式
    - 读系统寄存器 (见 GetSysReg 的实现)
    - 内联汇编
    - 从 Linux 提供的接口读取信息

```
12 struct HostFeatures {
11   /**
10    * @brief Backend features that change how codegen is generated from IR
 9    *
 8    * Specifically things that affect the IR->Codegen process
 7    * Not the x86->IR process
 6    */
 5   uint32_t DCacheLineSize {};
 4   uint32_t ICacheLineSize {};
 3   bool SupportsCacheMaintenanceOps {};
 2   bool SupportsAES {};
 1   bool SupportsCRC {};
20   bool SupportsCLZERO {};
 1   bool SupportsAtomics {};
 2   bool SupportsRCPC {};
 3   bool SupportsTSOImm9 {};
 4   bool SupportsRAND {};
 5   bool SupportsAVX {};
 6   bool SupportsSVE128 {};
 7   bool SupportsSVE256 {};
 8   bool SupportsSHA {};
 9   bool SupportsPMULL_128Bit {};
10   bool SupportsCSSC {};
11   bool SupportsFCMA {};
12   bool SupportsFlagM {};
13   bool SupportsFlagM2 {};
14   bool SupportsRPRES {};
15   bool SupportsPreserveAllABI {};
16   bool SupportsAES256 {};
17   bool SupportsSVEBitPerm {};
18   bool SupportsCPUIndexInTPIDRRO {};
19   bool SupportsFRINTTS {};
20   bool SupportsECV {};
21   bool SupportsWFXT {};
22   bool Supports3DNow {};
23   bool SupportsSSE4a {};
24
25   // Float exception behaviour
26   bool SupportsAFP {};
27   bool SupportsFloatExceptions {};
28
29   // Flag if this is InstCountCI
30   bool IsInstCountCI {};
31
32   // MIDR information
33   // Also used for determining number of CPU cores for CPUID
34   fextl::vector<uint32_t> CPUMIDRs;
35 };
36 } // namespace FEXCore
```

后端: JIT

- FEXCore/Source/Interface/Core/JIT/JITClass.h 中定义了 FEXCore::CPU::CPUBackend, FEXCore::CPU::Arm64JITCore 和 FEXCore::CPU::Arm64Emitter
- FEXCore/Source/Interface/Core/JIT/JIT.cpp 中利用先前自动生成的 IRDefines_Dispatch.inc 维护巨大的 Switch Case 语句
- 调用 Arm64Emitter 写二进制码

On Testing

FEX-Emu's CI uses these tests to avoid regression and prove correctness:

- A lot of handwritten assembly unit tests
  - Unable to run under RISC-V, but may be portable
- 64-bit posixtest from http://posixtest.sourceforge.net/
- 64-bit gvisor tests from https://github.com/google/gvisor,
- gcc-target-tests-32 and gcc-target-tests-64

- Some tests are really old (e.g. 17 years ago)

Issues:


What features will not be in FEX?
• Secure mode
• Hardware reference platform
• 16-bit x86
• 100% accurate
◦ What to compare to even?

Which causes

- Some testcases are failing in unittests
  - *timer-sigev-thread* crashes FEX with *SIGSEGV*
  - *synchronous-signal-block* trigger various quirks in FEX's signal handling
  - A lot of signal related tests can't run properly due to race/hang/timeout

目前的 workaround 是手动关掉这些测试点

On RISC-V Porting

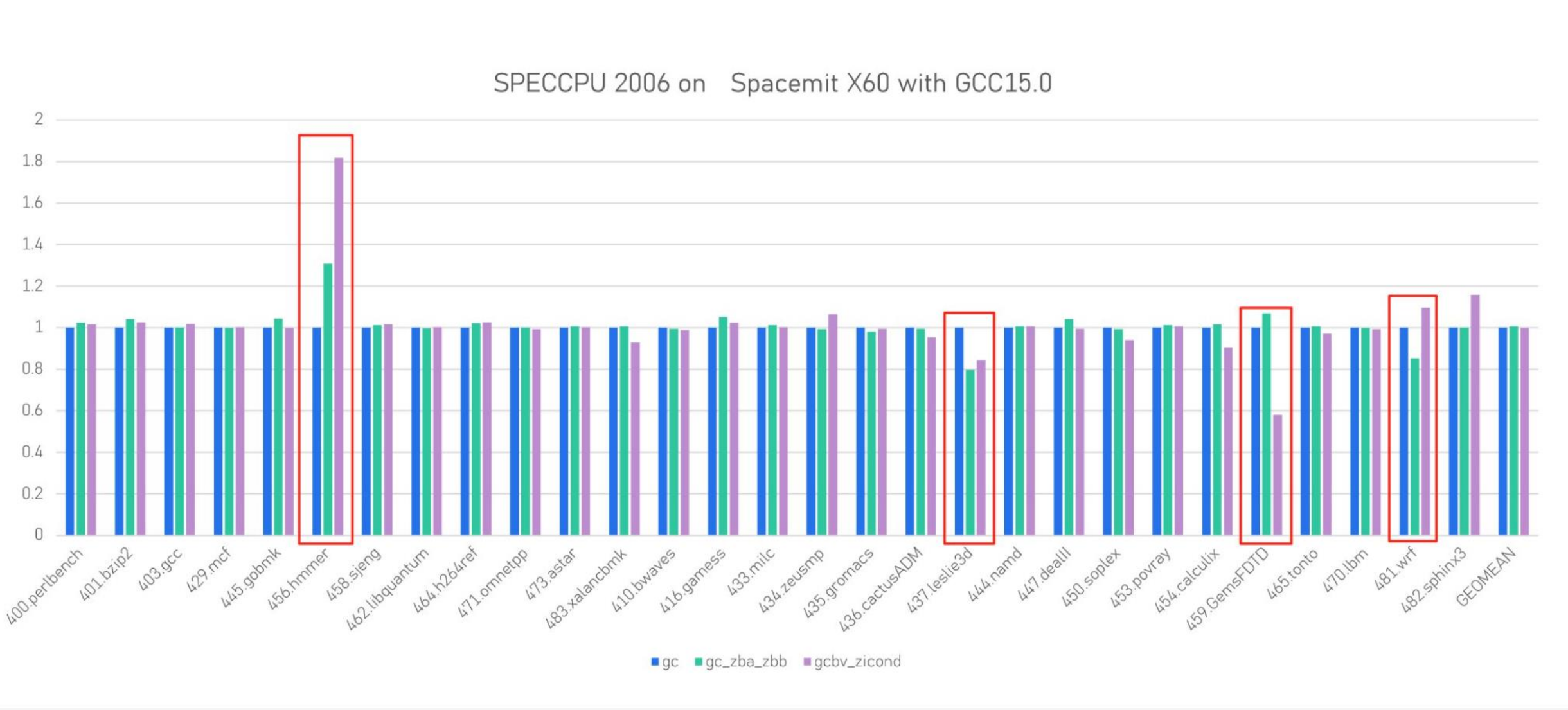
- ONGOING CPU 能力探测
  - `syscall: sys_riscv_hwprobe`
- DONE Syscall Handling
- TODO Library Thunking
- ONGOING JIT 后端的代码生成
- TODO RootFS

sys_riscv_hwprobe 支持情况

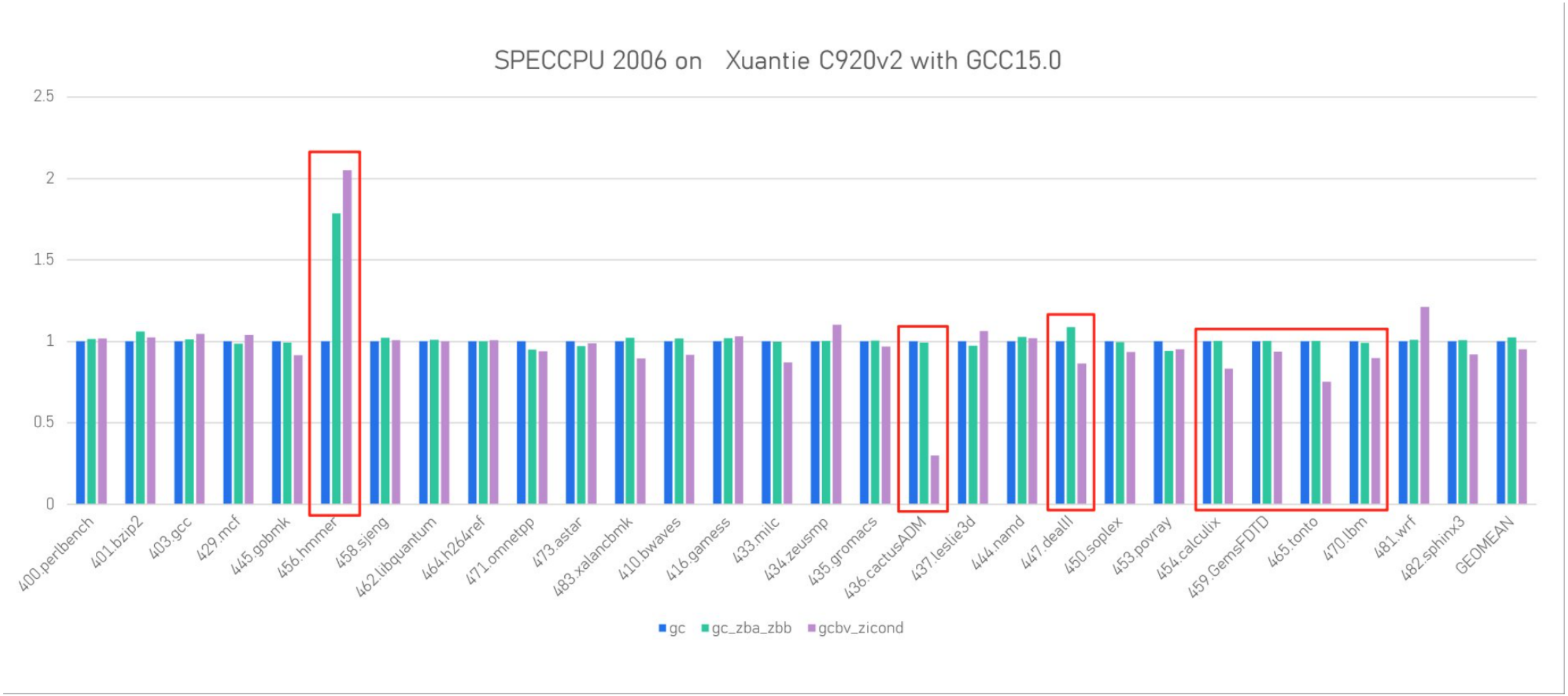
	Arch Linux	Debian/Ubuntu	Deepin	openEuler	Fedora
Lichee Pi 4A	✅	✅	❌	✅	✅
BPI-F3	暂无可用镜像	✅	✅	系统镜像无法启动	❌
CanMV K230	暂无可用镜像	❌	暂无可用镜像	暂无可用镜像	✅
VisionFive 2	✅	✅	✅	❌	❌
Megrez	暂无可用镜像	✅	✅	暂无可用镜像	系统镜像无法启动

目前已知的问题

- Full extension information != Better Codegen
- Example: (shown in the slides below)



https://rv64.zip


SPECCPU 2006 on Xuantie C920v2 with GCC15.0
2.5
2
1.5
1
0.5
0
400.perlbench
401.bzip2
403.gcc
429.mcf
445.gobmk
456.hmmer
458.sjeng
462.libquantum
464.h264ref
471.omnetpp
473.astar
483.xalancbmk
410.bwaves
416.gamess
433.milc
434.zeusmp
435.gromacs
436.cactusADM
437.leslie3d
444.namd
447.dealII
450.soplex
453.povray
454.calculix
459.GemsFDTD
465.tonto
470.lbm
481.wrf
482.sphinx3
GEOMEAN
gc
gc_zba_zbb
gcbv_zicond

THANKS