RISC-V Summit China

LORELEI：适用于通用模拟器的本地库加速方案

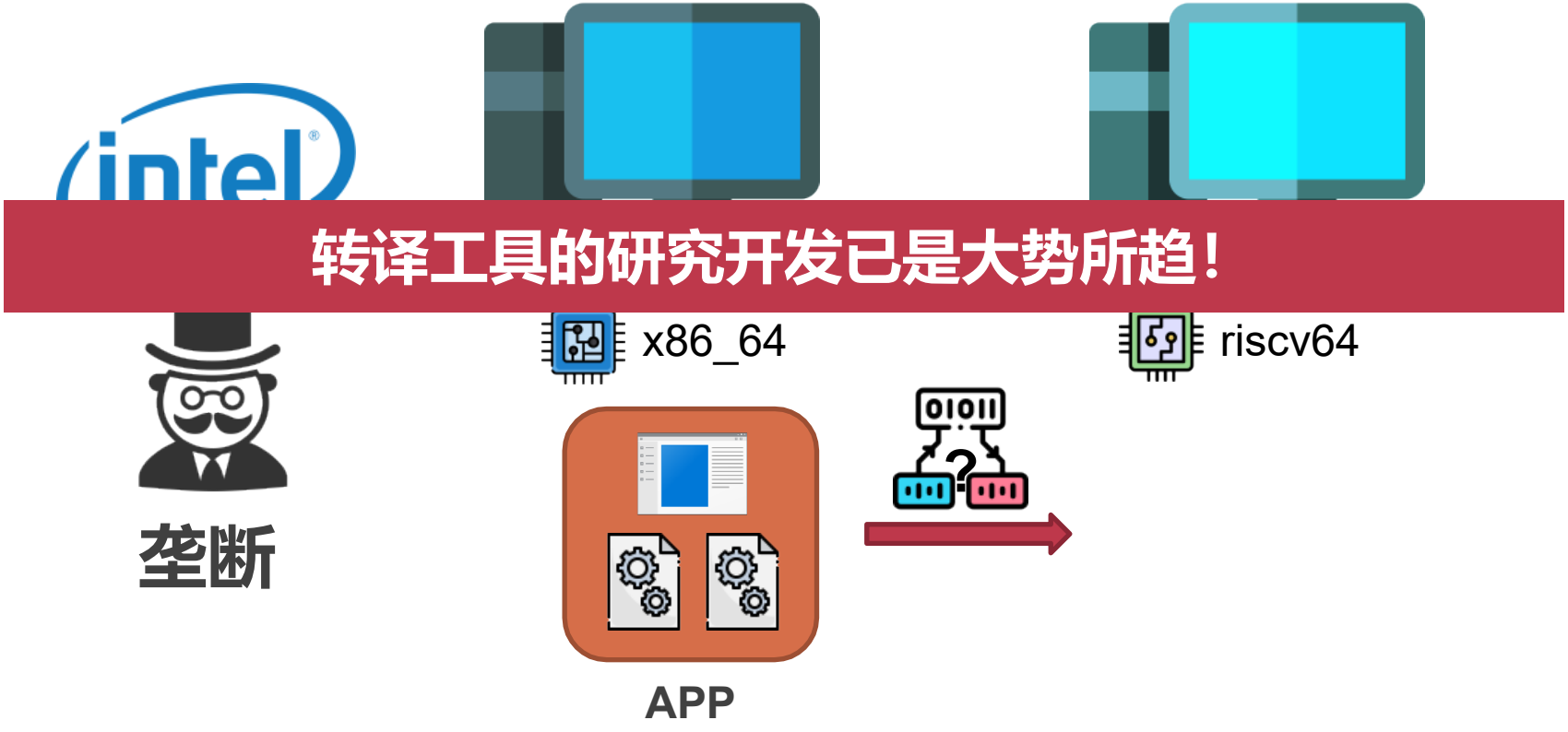
章子杨　汪　沄

上海交通大学

转译工具的需求与现状

研究背景

转译工具的需求



现有的转译工具

Apple Rosetta 2

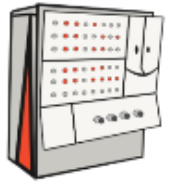
Windows On Arm

Huawei ExaGear

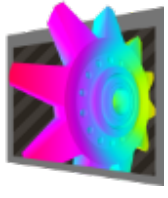
统统闭源

QEMU

Box64



Blinkenlights

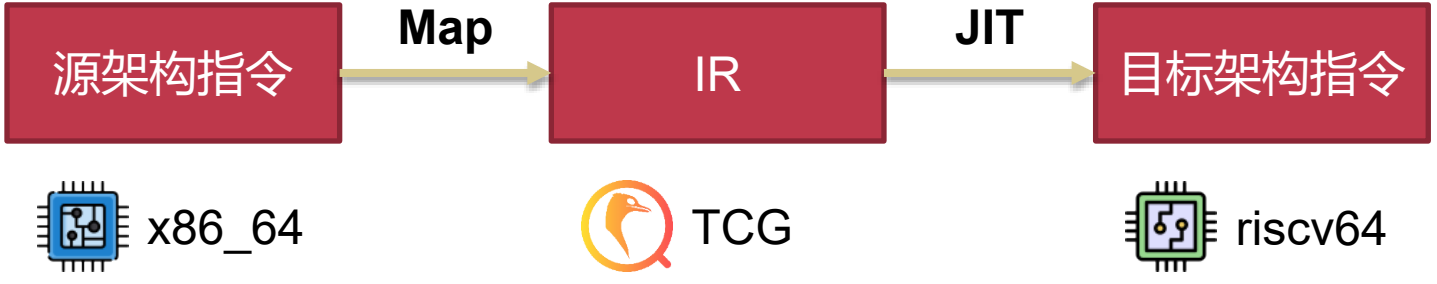


Fex-Emu

各有所长

研究现状

- 动态翻译具有更高的灵活性与用户体验，是主流选择。
- 动态二进制翻译器，备受关注的指标主要有执行效率、兼容性。
- 在现有的二进制翻译相关的学术研究与工程实现中，大部分优化措施，都集中在优化翻译规则上。



探索新的转译流程

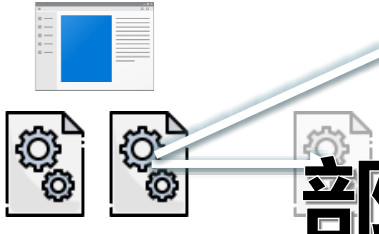
以 x86_64 翻译到 riscv64 为例

现状是，有些库是无法模拟的，如 CUDA 库以及硬件驱动库。

x86_64 源程序

完全模拟

痴于在“翻译规则”之
窠臼中上下求索



x86_64 源程序

部分模拟

将“物尽其用”思想
贯彻到底

使用本机动态库加速的基本方案（以 Box64 为例）

基本方案

本机优势

例：做一个矩阵乘法，如果“乘法”代码来自 Native 动态库，显然速度更快，因为充分发挥了本机的向量化优势，也没有转译开销。

效率优势：执行相同的逻辑，本地机器码直接执行比转译快得多

兼容性优势：支持 CUDA 指令、驱动指令等

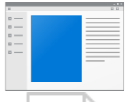
```
    int B[100 * 100];
    int C[100 * 100];

    // 初始化 A、B...

    matrix_mul(A, B, Result: C, Width: 100, Height: 100);

    printf(format: "%d\n", C[0]);

    return 0;
}
```

libmatrix.so

利用本地动态库的可行性

```c
main.c > ...
#include <stdio.h>

void matrix_mul(int A[], int B[], int Result[], int Width, int Height);

int main(int argc, char *argv[]) {
    int A[100 * 100];
    int B[100 * 100];
    int C[100 * 100];

    // 初始化 A、B...

    matrix_mul(A, B, Result: C, Width: 100, Height: 100);

    printf(format: "%d\n", C[0]);

    return 0;
}
```

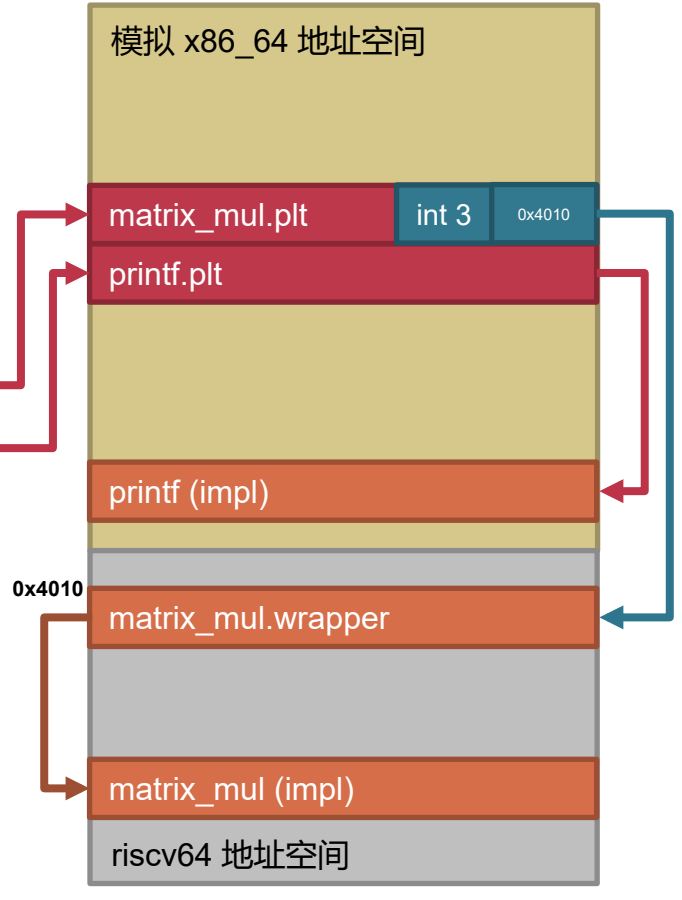
以 Box64 为例



main

libmatrix.so
libc.so

libmatrix.so

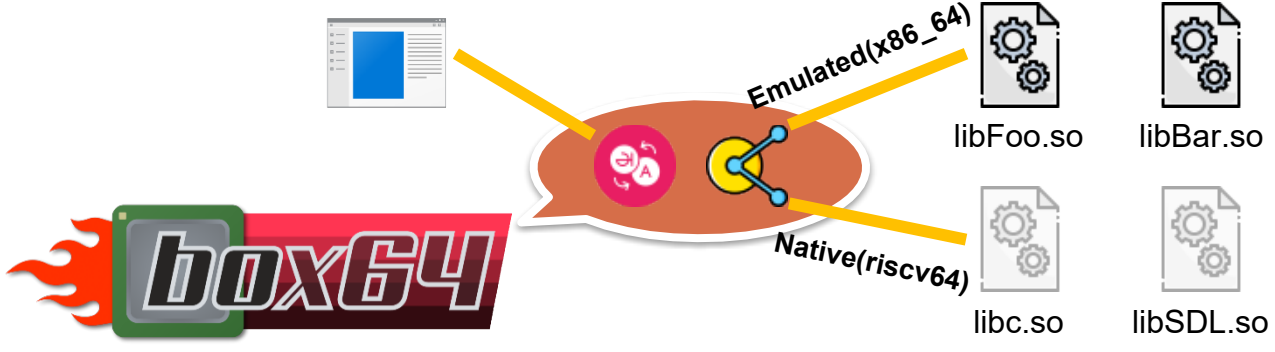


Box64 本机库加速方案的问题

现有方案的问题

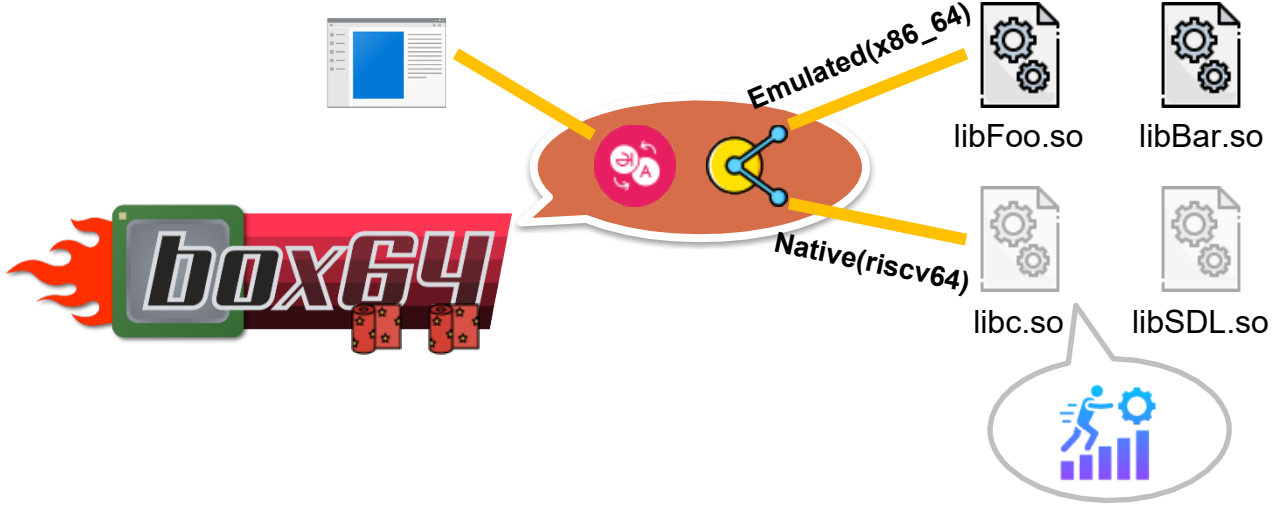
通用性差

- **1. 定制的翻译框架与动态链接器**
  - 传统翻译器（如 QEMU），只会完全模拟，工作流程完全与本机库无关
  - 传统动态链接器（通常指 ld-linux），工作流程完全与二进制翻译无关
  - Box64 的开发者，于是选择了从零开始开发**全新**的**模拟器**与**动态链接器**



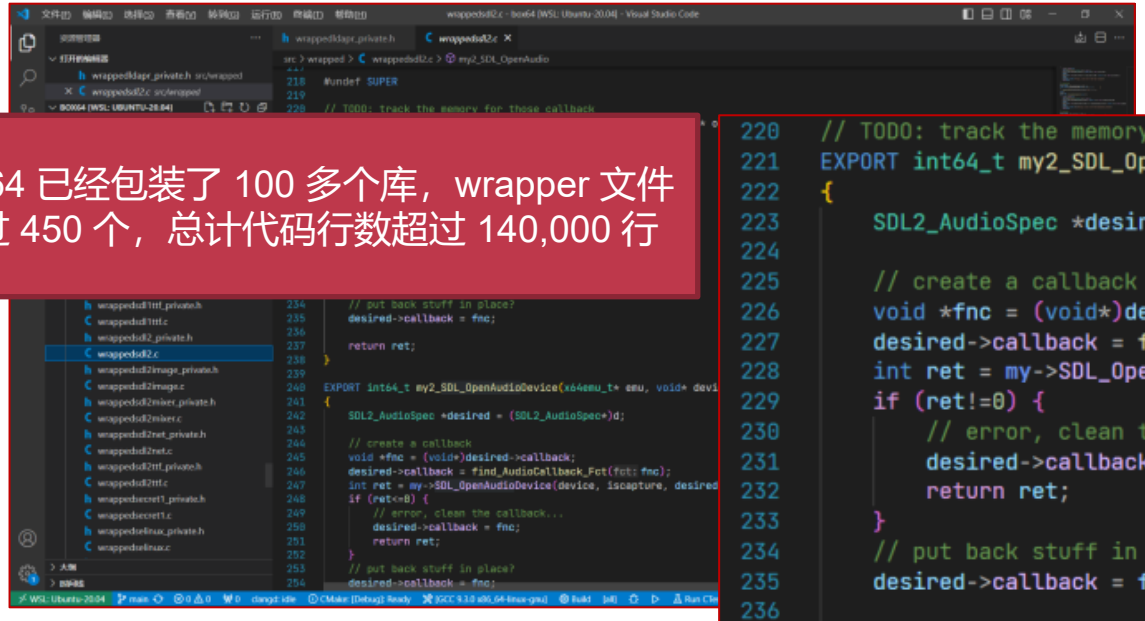
工作量大

- **2. 需要手工编写 Wrapper**
  - 由于 x86_64 与 riscv64 不兼容，因此需要转换层（称为 Wrapper）
  - Box64 需要手工编写 **ABI** 的 Wrapper（容易出错）
  - Box64 需要手工编写**每个库每个函数**的 Wrapper（代码量巨大）



Box64 的复杂性

- Box64 为大量第三方库的各函数都实现了包装器，开发工程规模极大



Box64 已经包装了 100 多个库，wrapper 文件超过 450 个，总计代码行数超过 140,000 行

```
220  // TODO: track the memory for those callback
221  EXPORT int64_t my2_SDL_OpenAudio(x64emu_t* emu, void* d, void* o)
222  {
223      SDL2_AudioSpec *desired = (SDL2_AudioSpec*)d;
224
225      // create a callback
226      void *fnc = (void*)desired->callback;
227      desired->callback = find_AudioCallback_Fct(fct: fnc);
228      int ret = my->SDL_OpenAudio(desired, (SDL2_AudioSpec*)o);
229      if (ret!=0) {
230          // error, clean the callback...
231          desired->callback = fnc;
232          return ret;
233      }
234      // put back stuff in place?
235      desired->callback = fnc;
236
237      return ret;
238  }
```

Wrapper 的内容：

1. 调用约定转换
2. 函数指针临时替换

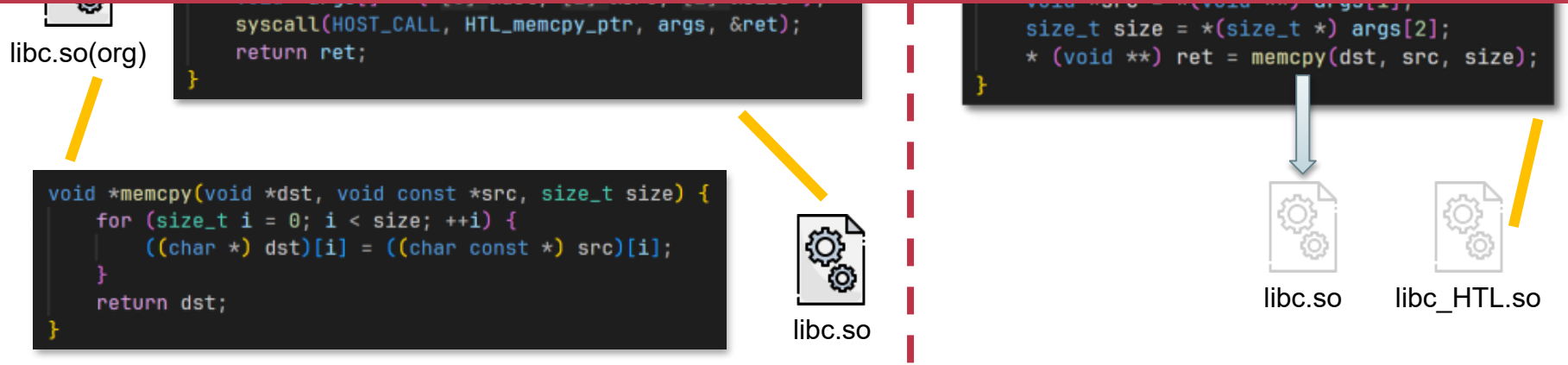
使已有的模拟器具备调用本机动态库加速的方法

LORELEI：通用本地库直通方案

Thunk Library

- **1. 转换库（Thunk Library，简称 TL）**
  - **对于一个要支持本机直通的库，需要准备两个转换库，一个是**

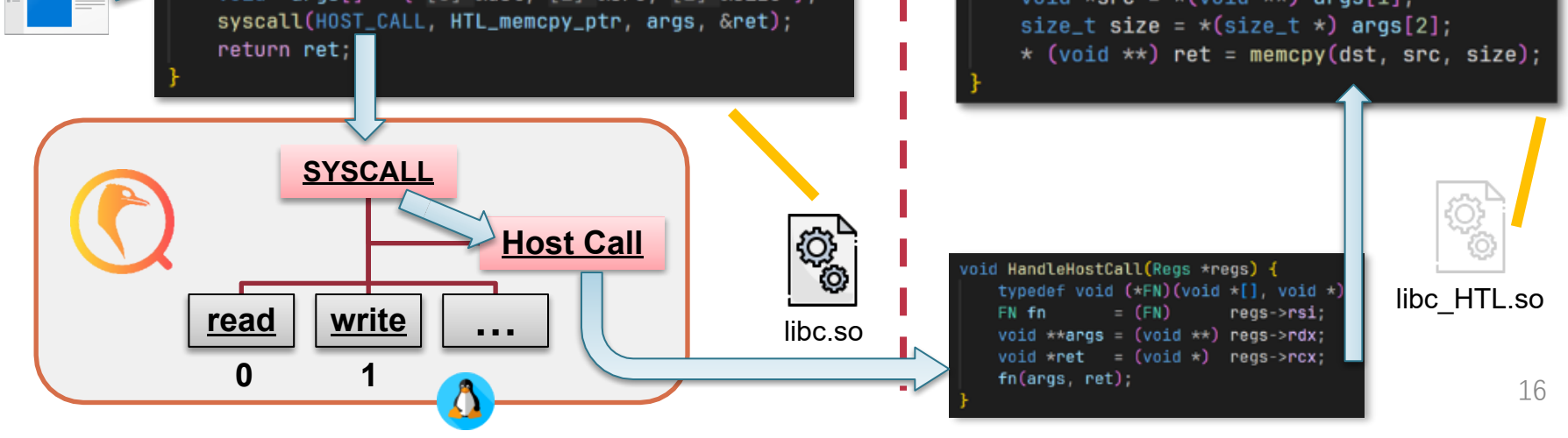
Thunk Library 的代码可自动生成，再由编译器生成二进制库。



对模拟器的改装

- **2. 模拟器增加系统调用处理分支**
  - 新增一个系统调用（ID 很大）

对模拟器的改动非常小！



其他技术难点

- 1. 回调函数 / 函数指针
- 2. 多线程
- 3. 可变长参数函数调用（Variadic Function）

本方案的优势

- **1. 通用性高**
  - 可扩展在通用的不支持本地直通的模拟器中，拯救老模拟器
  - 与具体架构无关，支持任意架构到任意架构（前提是内存对齐一致）
  - Thunk Library 与模拟器完全解耦，可复用
- **2. 可自动化**
  - Thunk Library 可自动生成
- **3. 复用现有基础设施**
  - 复用了已有的模拟器与动态链接器，维护成本低，不易出错

使用 Lorelei 扩展已有编译器后达成的加速效果

实验数据

实验信息

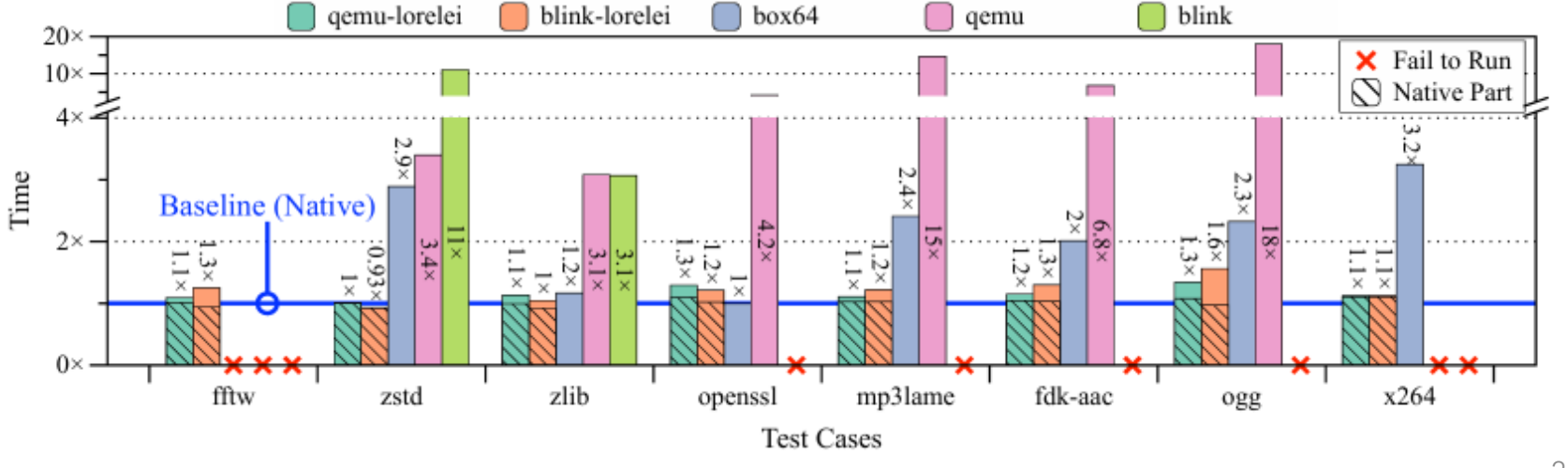
- **实验平台：**
  - Arm64: NVIDIA Jetson Xavier AGX
  - riscv64: UltraRISC DP1000
- **实验对象：**
  - QEMU
  - Blink: 由程序员 jart 开发的一个小型 x86_64 模拟器
  - Box64
  - QEMU-Lorelei: QEMU 应用 Lorelei 扩展后，修改量约 200 LOC
  - Blink-Lorelei: Blink 应用 Lorelei 扩展后，修改量约 400 LOC
- **已适配的库：**
  - OpenGL、SDL2、zlib、zstd 等
  - 自动生成代码行数约 100 万行，几乎没有手工代码

命令行程序执行时间

NVIDIA Jetson Xavier AGX

- Arch: ARM64
- GPU: NVIDIA GeForce GTX 1060
- Mem: 32GB

ID	Lib + CLI	Workload Specification
1	fftw3 + bench cli	Bench problem=6000000
2	zstd + zstd cli	Compress at Lv.19 (max)
3	zlib + minizip	Compress at Lv.9 (max)
4	crypto + openssl	AES speed test
5	mp3lame + FFmpeg	Encode WAV → MP3
6	fdk-aac + FFmpeg	Encode WAV → AAC
7	ogg + FFmpeg	Encode WAV → OGG
8	x264 + FFmpeg	Encode AVI → H.264 MP4



GUI 程序執行結果

NVIDIA Jetson Xavier AGX (Jetson)

- Arch: Arm64
- GPU: NVIDIA GeForce GTX 1060
- Mem: 32GB

UltraRISC DP1000 (DP1000)

- Arch: RISCV64
- GPU: AMD Radeon RX 560 GPU
- Mem: 16GB

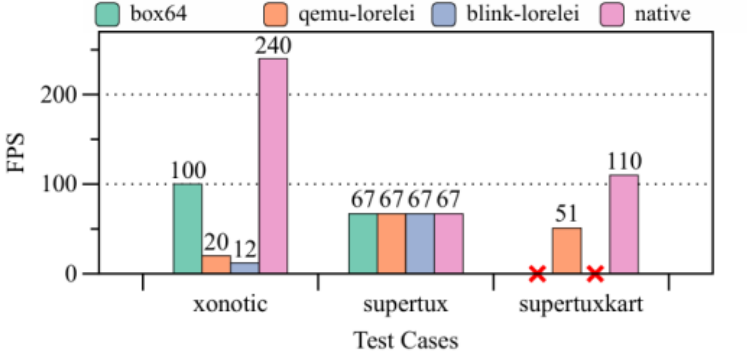


Fig. 7. Average FPS of GUI applications measured on Jetson (Arm64).

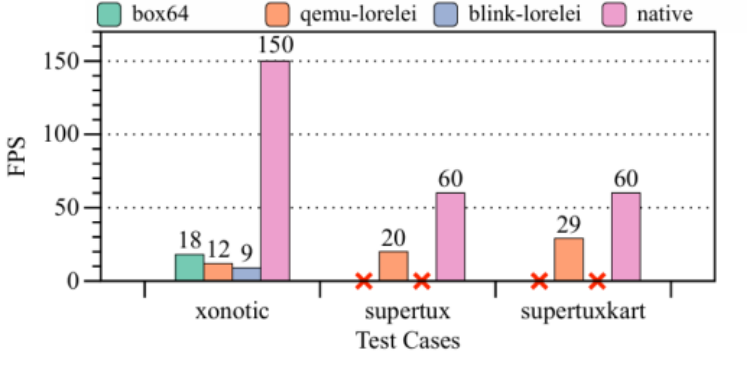
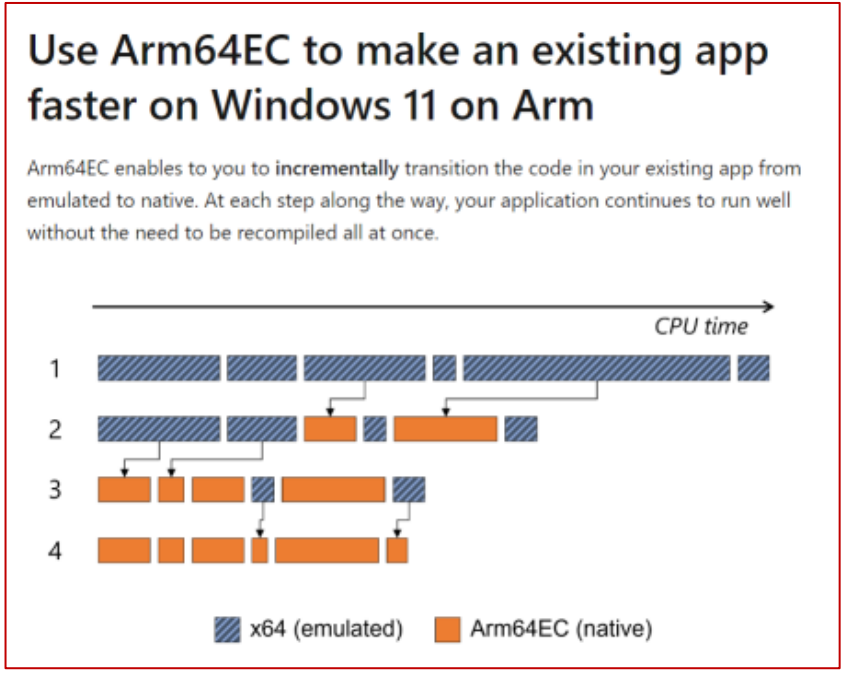


Fig. 8. Average FPS of GUI applications measured on DP1000 (RISCV64).

参考资料

https://learn.microsoft.com/en-us/windows/arm/arm64ec



Windows 11 Arm64EC

https://www.qemu.org/docs/master/user

QEMU

https://github.com/ptitSeb/box64

Box64

感谢聆听！

章子杨　汪　沄

上海交通大学