A technology for lifting machine code to high-performance LLVM IR

2025/06/10

NTT Software Innovation Center

Masashi Yoshimura

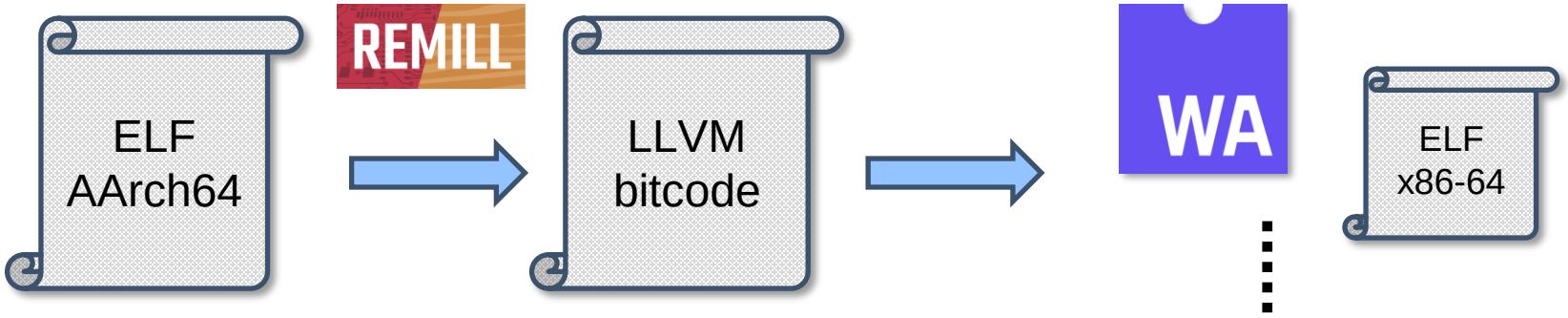
Porting applications to different environments

- Linux applications have been developed for a long time, so there are huge Linux binary assets.
- Reusing these assets can bring several benefits.
  - Reducing development costs
  - Improved application stability due to years of enhancements
  - etc...
- Today, the container is used in the cloud, local environment, and so on. However, "*WebAssembly*" is expected to be used as a more secure and portable application.

Porting Linux applications
to different environments (WebAssembly, ...)!

elfconv: Linux apps → WebAssembly, …

- elfconv: AOT binary translator for Linux/ELF binary
  - Repo: https://github.com/yomaytk/elfconv
  - Current status: Linux/ELF/AArch64 → WebAssembly, Linux/ELF/x86-64
    - We will work on translating to Mach-O binary in the future
  - **Remill**: The library for lifting machine code to LLVM IR



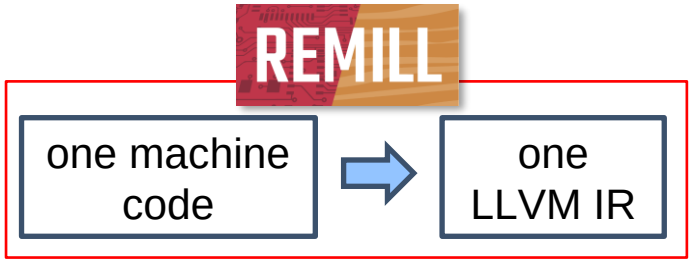
elfconv overview

NTT

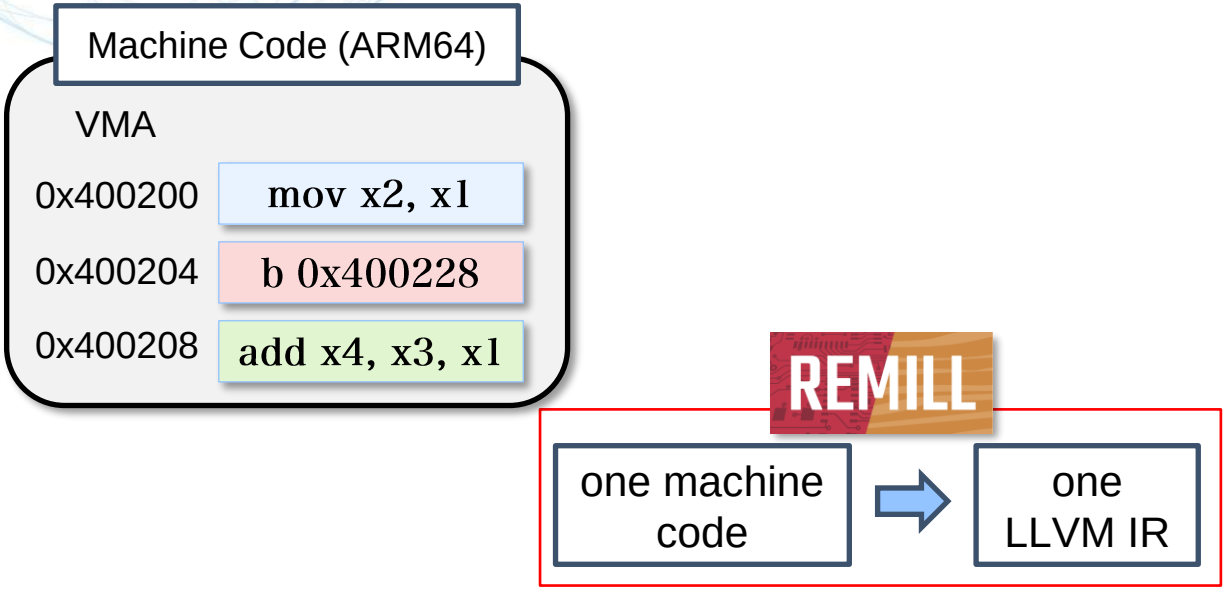
Demo can be accessed through the github repo

https://github.com/yomaytk/elfconv

What kind of LLVM IR does Remill generate?



What kind of LLVM IR does Remill generate?



What kind of LLVM IR does Remill generate?

NTT

CPU State (by Remill)

```
struct State {
    SIMD simd; // 512 bytes
    GPR gpr; // 518 bytes
    uint64_t NZCV;
    ...
};
```

Machine Code (ARM64)

VMA	
0x400200	mov x2, x1
0x400204	b 0x400228
0x400208	add x4, x3, x1

Remill Translation

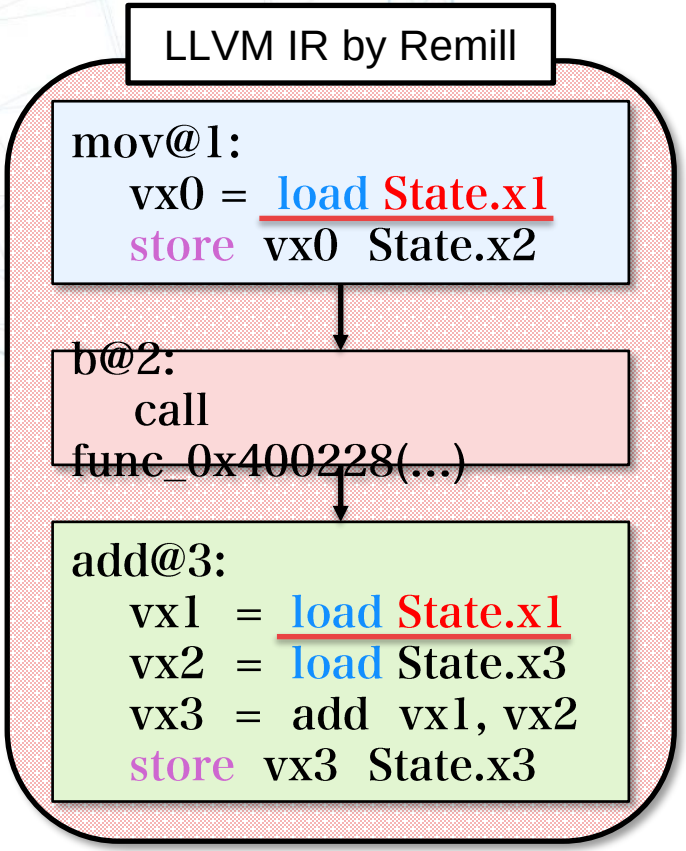
REMILL

one machine code → one LLVM IR

LLVM IR by Remill

```
mov@1:
    vx0 = load State.x1
    store vx0 State.x2
b@2:
  call func_0x400228(...)
add@3:
    vx1 = load State.x1
    vx2 = load State.x3
    vx3 = add vx1, vx2
    store vx3 State.x3
```

What's the performance bottleneck?



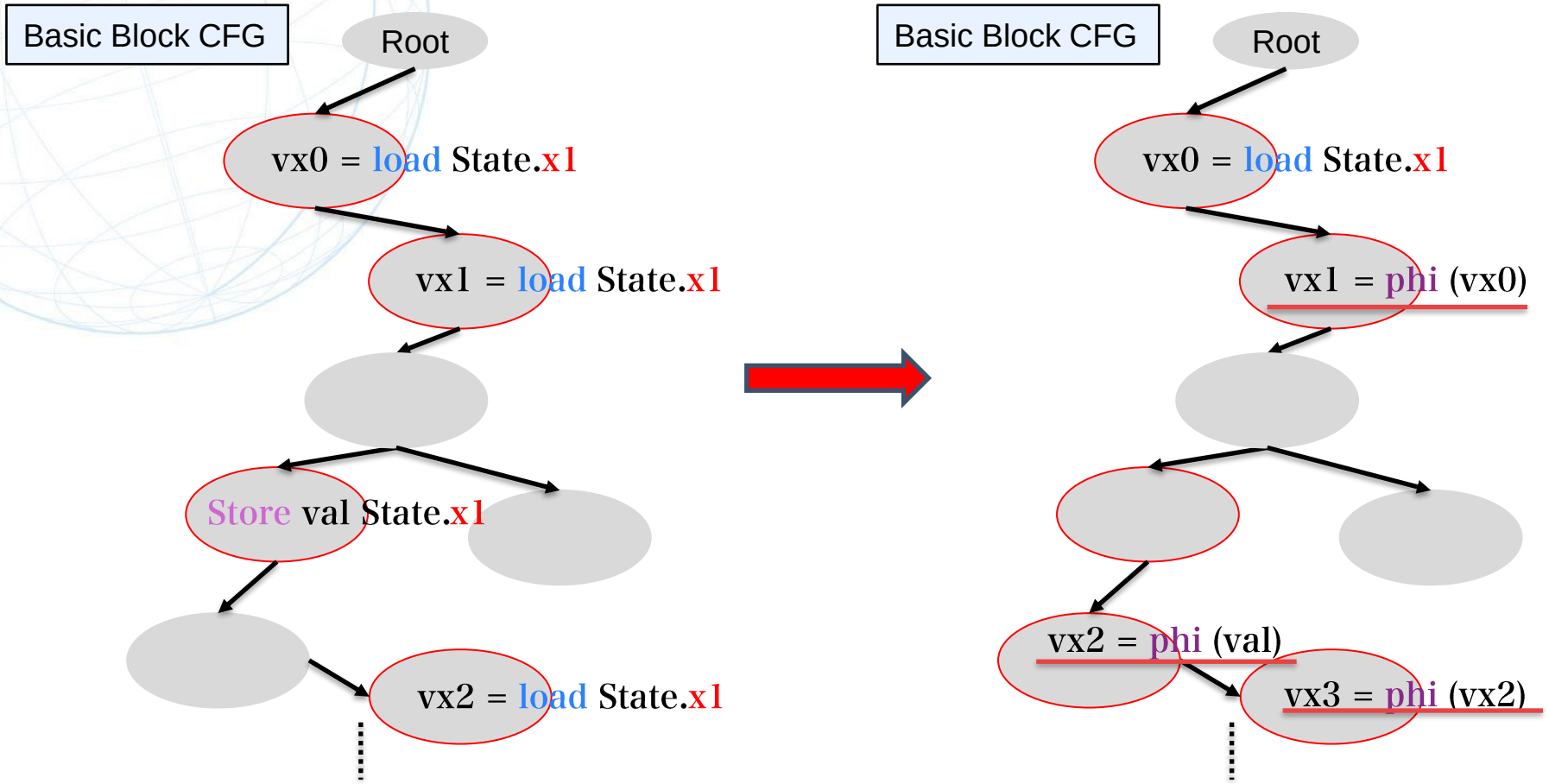
Remill generates the Basic Block "*independently*"

Virtual registers are not propagated between basic blocks

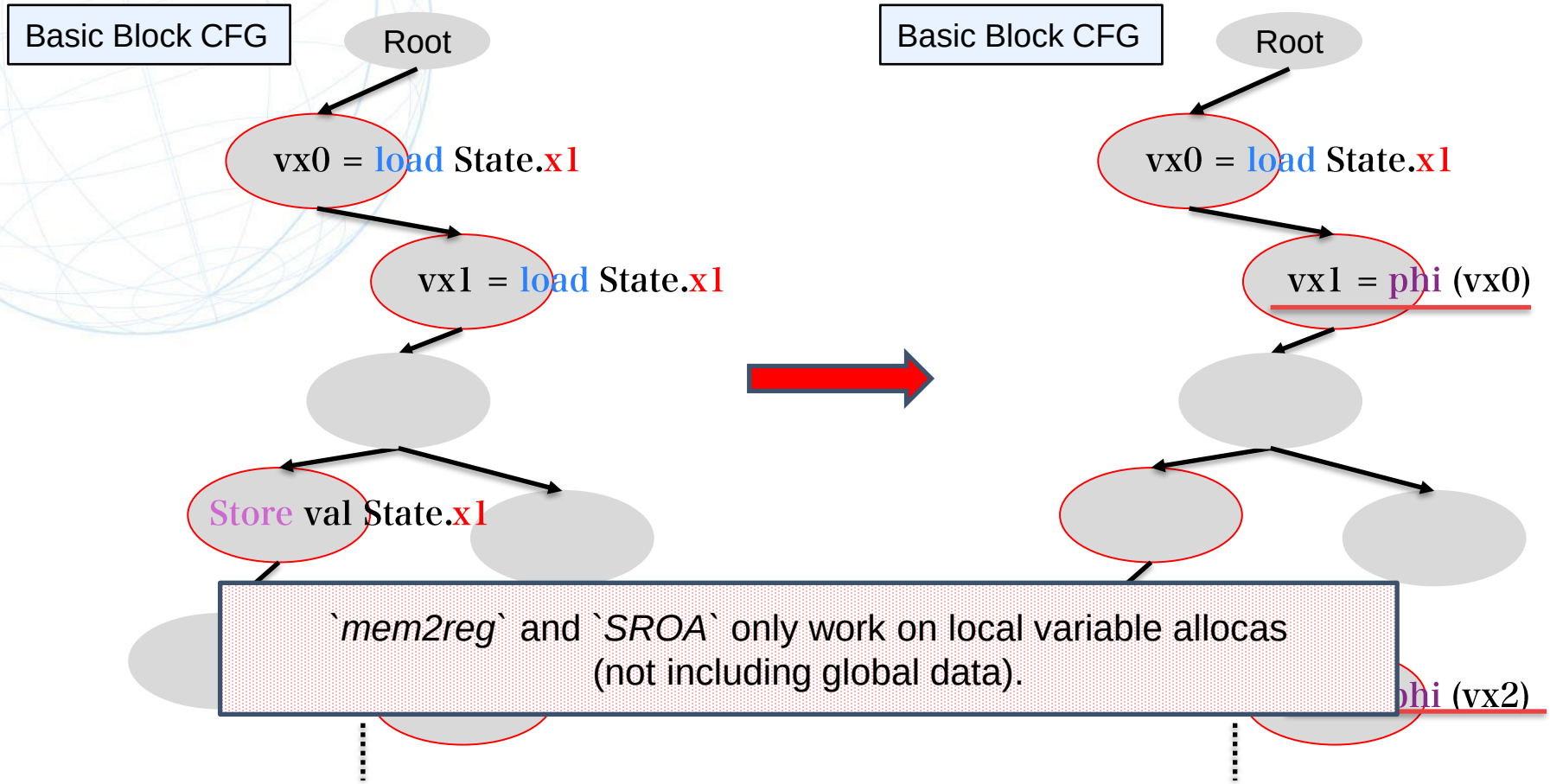
For the same register, access the CPU state (i.e., the data in global memory data) multiple times by many load and store.

This optimization PR: https://github.com/yomaytk/elfconv/pull/53

Overview of performance improvement



Overview of performance improvement



Benchmark Test

- elfconv enables the creation of binaries that perform well in practice.

	Prime Numbers Calculation	LINPACK benchmark
elfconv (normal)	1.98 (s)	726 (MFLOPS)
elfconv (Optimization)	1.39 (s)	1,256 (MFLOPS)
	1.43x faster (lower is better)	1.73x faster (higher is better)

Fig 1. Performance Improvement with Optimization

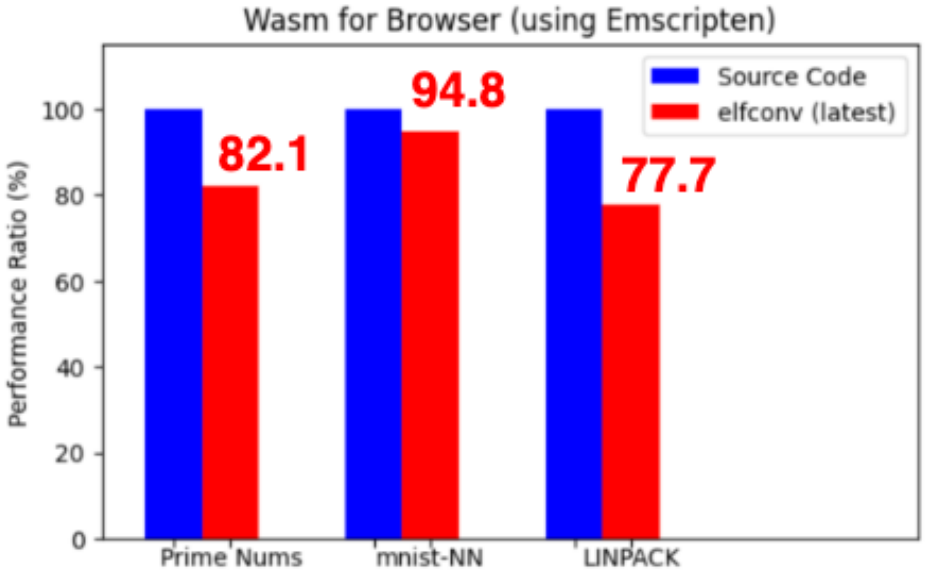


Fig 2. Wasm from source code by Emscripten
vs
Wasm from Linux/ELF by elfconv

Future Work

- Reducing Compile Time
  - It sometimes takes several tens of minutes, especially in targeting Wasm.
  - The generated LLVM IR may be too large.
- Implement more Linux system calls
  - e.g., difficult to implement fork, exec for Wasm
- Enhance aarch64 and x86-64 machine code conversion

Any issues or PRs are welcome!

Repo: https://github.com/yomaytk/elfconv