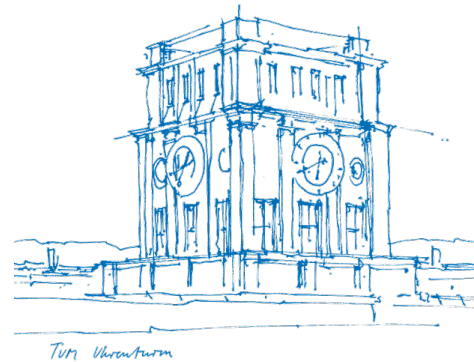
Instrew: Fast LLVM-based Dynamic Binary Instrumentation and Translation

Alexis Engelke Martin Schulz

Chair of Computer Architecture and Parallel Systems
Department of Informatics
Technical University of Munich

LLVM Performance Workshop
at CGO 2021, virtual



Dynamic Binary Translation

- Run program on other architecture, translate code for host CPU
- Use-cases: compatibility, architecture research
- Example: QEMU-user, Rosetta 2

x86-64

```
mov rax, rcx
add rax, 4
mov [rdx+rsi+16], rax
```

→

AArch64

```
add x0, x1, 4

add x16, x6, 16
str x0, [x2, x16]
```

Dynamic Binary Instrumentation

- Modify program behavior, add additional code/checks
- Use-cases: analysis, debugging, profiling, arch. research
- Examples: Valgrind, Pin

x86-64

```
mov rax, rcx
add rax, 4

mov [rdx+rsi+16], rax
```

→

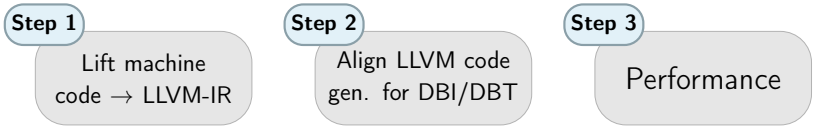
x86-64 +checks

```
mov rax, rcx
add rax, 4
call check_bounds
mov [rdx+rsi+16], rax
incq [num_stores]
```

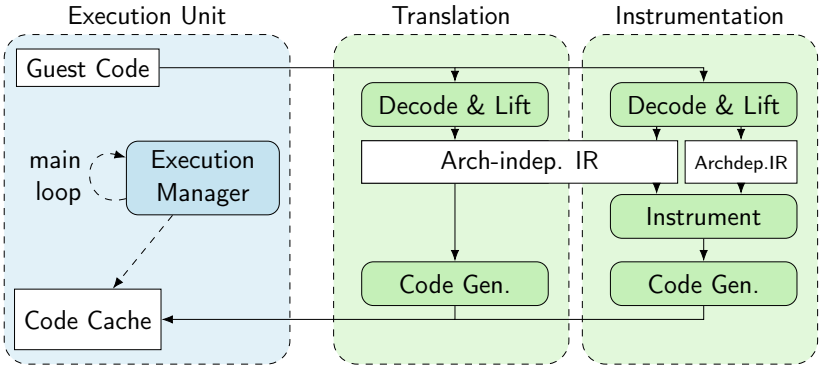
Motivation: LLVM for DBT/DBI

- Many DBT/DBI systems focus on **translation** performance
  - QEMU, Valgrind, Pin, ...
- Instead, use LLVM code generation for **run-time** performance

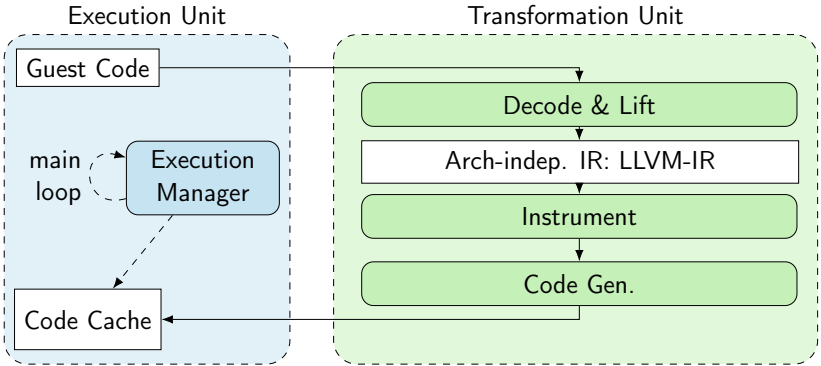
Instrew: a fast LLVM-based DBT/DBI framework



Overview: Process-level DBT and DBI



Overview: Process-level DBT and DBI



Lifting Machine Code to LLVM-IR

- Actually widely researched...
  - McSema: excellent coverage of x86-64 and others, but slow
  - HQEMU/DBILL (TCG→LLVM): limited by TCG (FP, basic blocks)
  - MCTOLL: very low instruction coverage
- ... but not sufficient for good coverage and overall performance!

Rellume: performance-oriented lifting library

Rellume: Lifting Approach

- Target-independent & idiomatic LLVM-IR
  - Use LLVM constructs where possible, e.g. vectors, comparisons
  - Helper functions for syscalls and `cpuid`
- Lifting performance: avoid heavy transformations (like mem2regs)
- Lifting stages:
  1. Decode instructions, recover control flow with basic blocks
  2. Lift individual instructions/basic blocks
  3. Fixup branches and PHI nodes
- Architectures: x86-64 (up to SSE2), RISC-V64 (imafdc)

Rellume: Example

```
define void @func_40061e(i8* %cpu) {
prologue:
  ; ...


bb_40061e:
  %rsp_2 = phi i64 [%rsp, %prologue]
  ; sub rsp, 176
  %rsp_3 = sub i64 %rsp_2, 176
  ; ... compute flags ...
  br label %epilogue

epilogue:
  ; ...
}
```

Lift instruction semantics

Rellume: Example

```
define void @func_40061e(i8* %cpu) {
prologue:
  ; ...



bb_40061e:
  ; ...



epilogue:
  ; ...
}
```

Single LLVM function, single parameter:
CPU state

- Instruction pointer
- Registers
- Status flags
- TLS pointer
- ...

Rellume: Example

```
define void @func_40061e(i8* %cpu) {
prologue:
  %rip_p_i8 = getelementptr i8, i8* %cpu, i64 0
  %rip_p = bitcast i8* %rip_p_i8 to i64*
  %rsp_p_i8 = getelementptr i8, i8* %cpu, i64 40
  %rsp_p = bitcast i8* %rsp_p_i8 to i64*
  %rsp = load i64, i64* %rsp_p
  ; ... load other registers ...
  br label %bb_40061e

bb_40061e:
  ; ...
epilogue:
  ; ...
}
```

Construct ptrs. into CPU struct

Load registers into SSA variables

Rellume: Example

```
define void @func_40061e(i8* %cpu) {
prologue:
  ; ...


bb_40061e:
  ; ...

epilogue:
  %rsp_4 = phi i64 [%rsp_3, %bb_40061e]
  store i64 %rsp_4, i64* %rsp_p
  ; ... store flags ...
  store i64 0x400625, i64* %rip_p
  ret void
}
```

Store new values

Store new RIP

The good

Functional!

Fast!

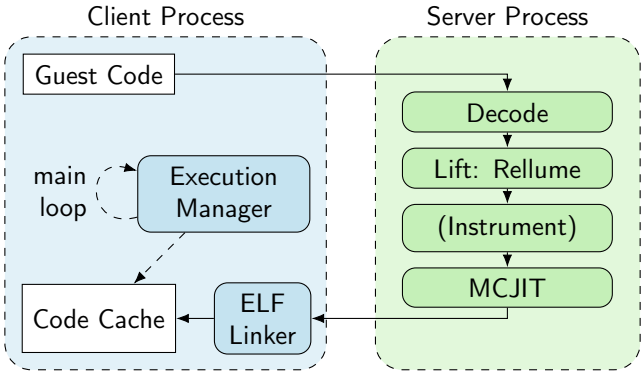
The bad: LLVM's limitations

- Better use of host registers possible
  - Keep more guest registers in host registers, less memory accesses
  - x86-64 has *(slightly broken)* HHVM calling convention
  - Need for general all-regs CC
- LLVM 11 is 40% slower than LLVM 9
  - Not investigated yet

The ugly: hard problems

- Floating-point semantics
  - Rounding mode depends on register, impossible to model in LLVM-IR
  - Current state: ignore rounding, except for FP→int (emulate in software)
- Load-Locked/Store-Conditional atomics
  - Can't be represented in general LLVM-IR
  - Ideas: hardware transactional memory, global mutex, stop-the-world
- Memory Consistency: multi-threaded x86-64 on ⟨*something else*⟩
  - Two approaches: fences everywhere or hardware support
  - Current state: single-threaded only ☺

Instrew: Client-Server Architecture



- More flexible, options:
- Different server machine
- Permanent server
- Transparent caching
- ...

Instrumentation

- Instrumentation tool is a shared library on server-side
- Loads initial library functions into client
  - Compile LLVM code for target architecture, send to client
- Modify lifted code prior to compilation
  - Optionally, lifter adds instruction information to LLVM-IR
- Memory management currently rather simple
  - 48 bytes storage accessible in CPU state
  - Further memory $\rightsquigarrow$ custom allocation with `mmap`

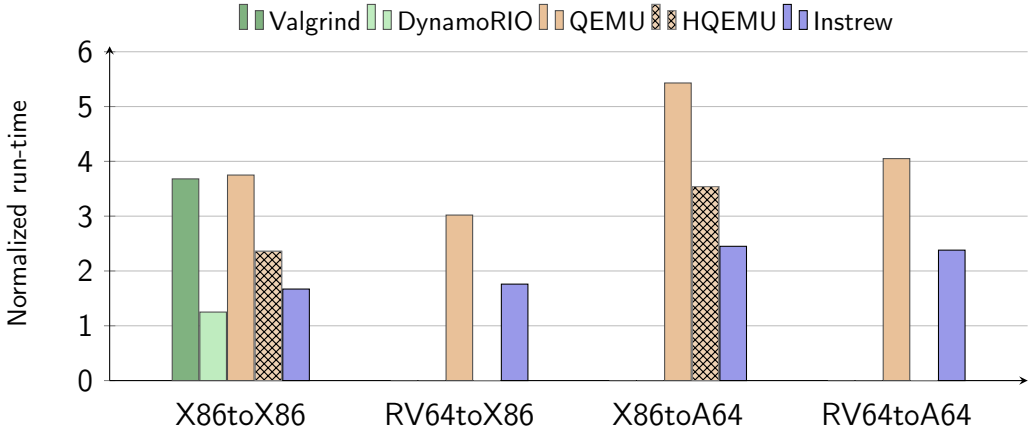
Evaluation

- Run on SPEC CPU2017 benchmarks
- Source architectures: x86-64, RISC-V64
- Target architectures: x86-64, AArch64
- LLVM 9
- Comparison with Valgrind, DynamoRIO, QEMU, HQEMU; base: host run

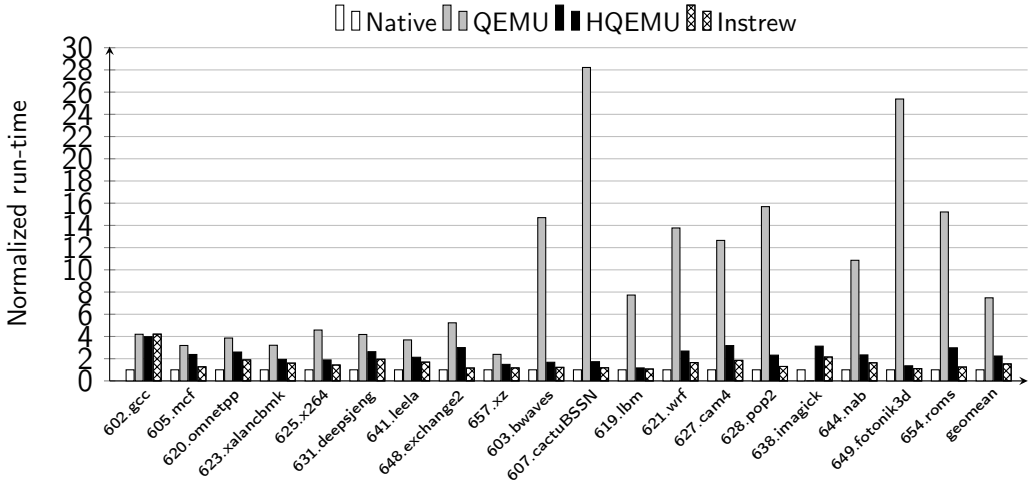
System x86-64: 2×Intel Xeon CPU E5-2697 v3 (Haswell) @ 2.6 GHz (3.6 GHz Turbo), 17 MiB L3 cache; 64 GiB main memory; SUSE Linux 15.1 SP1; Linux kernel 4.12.14-95.32; 64-bit mode. Compiler: GCC 9.2.0 with `-O3 -march=x86-64`, implies SSE/SSE2 but no SSE3+/AVX. Libraries: glibc 2.32; LLVM 9.0.
System AArch64: 2×Cavium ThunderX2 99xx @ 2.5 GHz, 32 MiB L3 cache; 512 GiB main memory; CentOS 8; Linux kernel 4.18.0-193.14.2; 64-bit mode. Compiler: GCC 10 with `-O3`. Libraries: glibc 2.32; LLVM 9.0.
SPEC CPU2017 intspeed+fpspeed benchmarks, ref workload, single thread.

SPEC CPU2017int Results

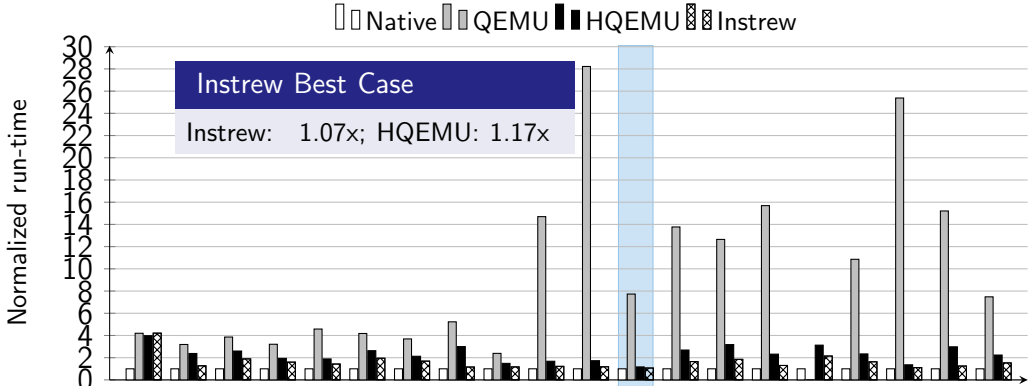
Results normalized to native execution on host



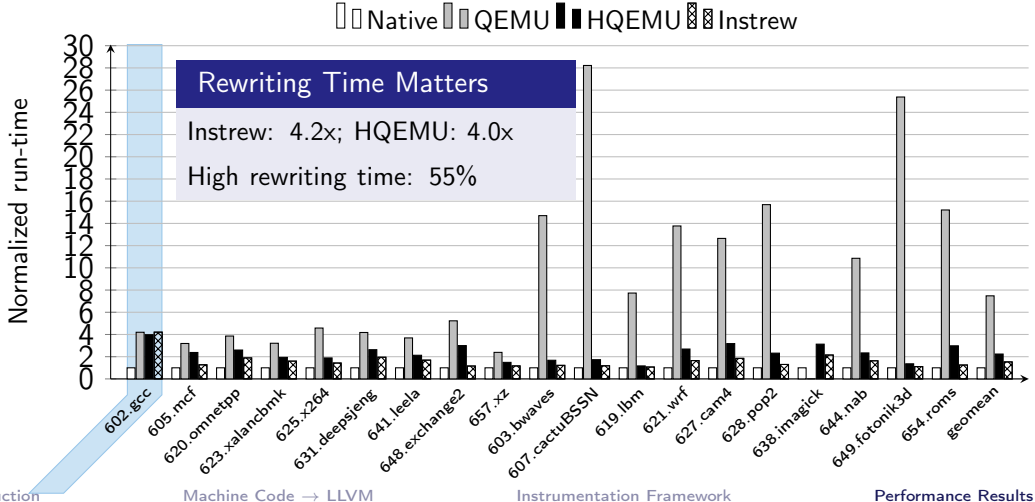
SPEC CPU2017int+fp Results (x86-64→x86-64)



SPEC CPU2017int+fp Results (x86-64→x86-64)

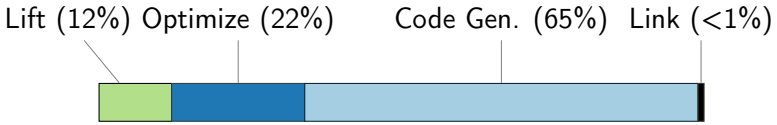


SPEC CPU2017int+fp Results (x86-64→x86-64)



Rewriting Overhead

- Mean Rewriting Time: $<2\%$
  - Notable exception: 602.gcc with 55%
  - HHVM-CC on x86 hosts: codegen.-time increase 12%–78%
- Mean Rewriting Time Breakdown:
  - Most time spent for machine code generation
  - SelectionDAG instruction selector known to be slow
  - Replacement GlobalISel not yet ready



Instrew: LLVM-based DBI/DBT

- Fast Dynamic Binary Instrumentation/Translation based on LLVM
- Lift whole functions directly to LLVM-IR
- Client-server approach enabling further optimizations
- 50% less overhead compared to current LLVM-based DBT

Instrew is **Free Software**!

`https://github.com/aengelke/instrew`