Cross-ISA Machine Instrumentation
using Fast and Scalable
Dynamic Binary Translation

Emilio G. Cota
Luca P. Carloni

Columbia University
Motivation

Dynamic Binary Translation (DBT) is widely used, e.g.

- Computer architecture simulation
- Software/ISA prototyping (a.k.a. emulation, virtual platforms)
- Dynamic analysis (security, correctness)
Motivation

Dynamic Binary Translation (DBT) is widely used, e.g.

- Computer architecture simulation
- Software/ISA prototyping (a.k.a. emulation, virtual platforms)
- Dynamic analysis (security, correctness)

DBT state of the art

<table>
<thead>
<tr>
<th></th>
<th>Speed</th>
<th>Cross-ISA</th>
<th>Full-system</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamoRIO</td>
<td>✔️ Fast</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Pin</td>
<td>✔️ Fast</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>QEMU (&amp; derivatives)</td>
<td>✗ Slow</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Motivation

- Pin/DynamoRIO are **instrumentation** tools
- Several QEMU-derived tools add **instrumentation** to QEMU
  - e.g. DECAF, PANDA, PEMU, QVMII, QTrace, TEMU
  - However, they widen the perf gap with DynamoRIO/Pin
Motivation

- Pin/DynamoRIO are **instrumentation** tools
- Several QEMU-derived tools add **instrumentation** to QEMU
  - e.g. DECAF, PANDA, PEMU, QVMII, QTrace, TEMU
  - However, they widen the perf gap with DynamoRIO/Pin

**Our goal:**

Fast, cross-ISA, full-system instrumentation
Fast, cross-ISA, full-system instrumentation

*How fast?*

- Goal: match Pin's speed when using it for simulation
  - Note that Pin is same-ISA, user-only
Fast, cross-ISA, full-system instrumentation

How fast?

- Goal: match Pin's speed when using it for simulation
  - Note that Pin is same-ISA, user-only

How to get there? Need to:

- Increase **emulation speed and scalability**
  - QEMU is slower than Pin, particularly for full-system and floating point (FP) workloads
  - QEMU does not scale for workloads that translate a lot of code in parallel, e.g. parallel compilation in the guest
- Support fast, **cross-ISA instrumentation of the guest**
Open source: https://www.qemu.org
Widely used in both industry and academia
Supports many ISAs through DBT via TCG, its Intermediate Representation (IR)

• Complex instructions are emulated in "helper" functions (not pictured)
QEMU*

Open source: https://www.qemu.org
Widely used in both industry and academia
Supports many ISAs through DBT via TCG, its Intermediate Representation (IR)
  - Complex instructions are emulated in "helper" functions (not pictured)

Our contributions are not QEMU-specific
They are applicable to cross-ISA DBT tools at large

[*] Bellard. "QEMU, a fast and portable dynamic translator", ATC, 2005
QEMU baseline

User-mode (QEMU-user)
- DBT of user-space code only
- System calls are run natively on the host machine

System-mode (QEMU-system)
- Emulates an entire machine, including guest OS + devices
- QEMU uses one host thread per guest vCPU ("multi-core on multi-core") [*]
  - Parallel code execution, serialized code translation with a global lock

Qelt's contributions

Emulation Speed

1. Correct cross-ISA FP \textit{emulation} using the host FPU
2. Integration of two state-of-the-art optimizations:
   - indirect branch handling
   - dynamic sizing of the \textit{software} TLB
3. Make the DBT engine \textit{scale} under heavy \textit{code translation}
   - Not just during \textit{execution}

Instrumentation

4. Fast, ISA-agnostic instrumentation layer for QEMU
1. Cross-ISA FP Emulation

- Rounding, NaN propagation, exceptions, etc. have to be emulated correctly
- Reading the host FPU flags is *very* expensive
  - soft-float is faster, which is why QEMU uses it

Qelt uses the host FPU for a **subset of FP operations**, *without ever reading the host FPU flags*
  - Fortunately, this subset is *very common*
  - defers to soft-float otherwise

![Graph showing normalized throughput for different operations.](baseline (incorrect): always uses the host FPU and never reads excp. flags)
1. Cross-ISA FP Emulation

```c
float64 float64_mul(float64 a, float64 b, fp_status *st) {
    float64 input flush2(&a, &b, st);
    if (likely(float64_is_zero_or_normal(a) &&
              float64_is_zero_or_normal(b) &&
              st->exception_flags & FP_INEXACT &&
              st->round_mode == FP_ROUND_NEAREST_EVEN)) {
        if (float64_is_zero(a) || float64_is_zero(b)) {
            bool neg = float64_is_neg(a) ^ float64_is_neg(b);
            return float64_set_sign(float64_zero, neg);
        } else {
            double ha = float64_to_double(a);
            double hb = float64_to_double(b);
            double hr = ha * hb;
            if (unlikely(isinf(hr))) {
                st->float_exception_flags |= float_flag_overflow;
            } else if (unlikely(fabs(hr) <= DBL_MIN)) {
                goto soft_fp;
            }
            return double_to_float64(hr);
        }
    } else if (unlikely(fabs(hr) <= DBL_MIN)) {
        goto soft_fp;
    }
    return double_to_float64(hr);
}
```

Common case:
- A, B are normal or zero
- Inexact already set
- Default rounding

How common?

99.18%

of FP instructions in SPECfp06
2. Other Optimizations

derived from state-of-the-art DBT engines

A. Indirect branch handling

- We implement Hong et al.'s [A] technique to speed up indirect branches
  - We add a new TCG operation so that all ISA targets can benefit


2. Other Optimizations

derived from state-of-the-art DBT engines

A. Indirect branch handling

- We implement Hong et al.'s [A] technique to speed up indirect branches
  - We add a new TCG operation so that all ISA targets can benefit

B. Dynamic TLB resizing (full-system)

- Virtual memory is emulated with a software TLB

2. Other Optimizations

derived from state-of-the-art DBT engines

A. Indirect branch handling

- We implement Hong et al.'s [A] technique to speed up indirect branches
  - We add a new TCG operation so that all ISA targets can benefit

B. Dynamic TLB resizing (full-system)

- Virtual memory is emulated with a *software TLB*
- Tong et al. [B] present TLB resizing based on TLB use rate at flush time
  - We improve on it by incorporating *history to shrink less aggressively*
    - Rationale: if a memory-hungry process was just scheduled out, it is likely that it will be scheduled in in the near future

Indirect branch + FP improvements

user-mode x86_64-on-x86_64. Baseline: QEMU v3.1.0

+indirect branch handling optimization
+floating point using the host FPU

SPECint06

SPECfp06

INT-geomean

FP-geomean
TLB resizing
full-system x86_64-on-x86_64. Baseline: QEMU v3.1.0

- +TLB history: takes into account recent usage of the TLB to shrink less aggressively, improving performance.
3. Parallel code translation
with a shared translation block (TB) cache

Monolithic TB cache (QEMU)

- Parallel TB execution (green blocks)
- Serialized TB generation (red blocks) with a global lock
3. Parallel code translation with a shared translation block (TB) cache

Monolithic TB cache (QEMU)
- Parallel TB execution (*green* blocks)
- **Serialized TB generation (*red* blocks) with a global lock**

Partitioned TB cache (Qelt)
- Parallel TB execution
- Parallel TB generation (one region per vCPU)

- vCPUs generate code at different rates
  - Appropriate region sizing ensures low code cache waste
Parallel code translation

Guest VM performing parallel compilation of Linux kernel modules, x86_64-on-x86_64

- QEMU scales for parallel workloads that rarely translate code, such as PARSEC [*]

- However, QEMU does not scale for this workload due to contention on the **lock serializing code generation**

- +parallel generation **removes the scalability bottleneck**
  - Scalability is similar (or better) to KVM's

QEMU cannot instrument the guest

- Would like **plugin** code to receive **callbacks** on **instruction-grained events**
  - e.g. memory accesses performed by a particular instruction in a translated block (TB), as in Pin
4. Cross-ISA Instrumentation

Instrumentation with Qelt

- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
4. Cross-ISA Instrumentation

Instrumentation with Qelt

- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
  - They can use a decoder; Qelt only sees opaque insns/accesses
4. Cross-ISA Instrumentation

Instrumentation with Qelt

- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
  - They can use a decoder; Qelt only sees opaque insns/accesses
- Qelt then substitutes "empty" instrumentation with the actual calls to plugin **callbacks** (or removes it if not needed)
Instrumentation with Qelt

- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
  - They can use a decoder; Qelt only sees opaque insns/accesses
- Qelt then substitutes "empty" instrumentation with the actual calls to plugin **callbacks** (or removes it if not needed)
- Other features (see paper): **direct** callbacks, inlining, helper instrumentation

**DIAGRAM:**

```
Guest Code
store 19, 30384
br 0x12004d090

translate()

IR with empty instrumentation
movi64 tmp3, 0x8000000000000000
movi32 tmp1, 0x23
movi64 tmp4, 0x00
ldi32 tmp0, env, 0x8000000000000000
movi64 tmp3, tmp2
call empty_mem_cb, \n$0x10,$0, tmp0, tmp1, tmp3, tmp4

plugin_dispatch()

plugin_inject()

plugin 0
...
plugin n

Instrumented IR
movi64 tmp3, 0x8000000000000000
addi64 tmp2, 112, tmp3
movi64 tmp2, 0x19, tmp2, leq1
movi32 tmp1, 0x23
movi64 tmp4, 0x00
ldi32 tmp0, env, 0x8000000000000000
movi64 tmp3, tmp2
... call plugin_mem_cb, \n$0x10,$0, tmp0, tmp1, tmp3, tmp4
```

**1.16**
Full-system instrumentation

x86_64-on-x86_64 (lower is better). Baseline: KVM

Qelt faster than the state-of-the-art, even for heavy instrumentation (cachesim)
Full-system instrumentation

x86_64-on-x86_64 (lower is better). Baseline: KVM

Qelt faster than the state-of-the-art, even for heavy instrumentation (cachesim)
Full-system instrumentation

x86_64-on-x86_64 (lower is better). Baseline: KVM

PANDA | QVMII | Qelt | Qelt-inline

Qelt faster than the state-of-the-art, even for heavy instrumentation (cachesim)
• Qelt has narrowed the gap with Pin/DRIO for no instr., although for FP the gap is still significant.
Qelt has narrowed the gap with Pin/DRIO for no instr., although for FP the gap is still significant.

DRIO is not designed for non-inline instr.
User-mode instrumentation

x86_64-on-x86_64 (lower is better). Baseline: native

- Qelt has narrowed the gap with Pin/DRIO for no instr., although for FP the gap is still significant
- DRIO is not designed for non-inline instr.
- Qelt is competitive with Pin for heavy instrumentation (cachesim), while being cross-ISA
Conclusions

Qelt's contributions

- Fast FP emulation leveraging the host FPU
- Scalable DBT-based code generation
- Fast, ISA-agnostic instrumentation layer
  - Performance for simulator-like instrumentation is competitive with state-of-the-art same-ISA, user-mode emulators such as Pin
Conclusions

Qelt's contributions

- Fast FP emulation leveraging the host FPU
- Scalable DBT-based code generation
- Fast, ISA-agnostic instrumentation layer
  - Performance for simulator-like instrumentation is competitive with state-of-the-art same-ISA, user-mode emulators such as Pin

Qelt's impact

- Instrumentation layer: under review by the QEMU community
- Everything else: **merged upstream**, to be released in QEMU v4.0 (April'19)
  - Contributions well-received (and improved!) by the QEMU community
- We hope our work will enable further adoption of QEMU to perform cross-ISA emulation and instrumentation
¡Gracias! → Español frontend

ありがとうございます！→ 日本語 frontend

Grazie! → Italiano frontend

TCG Ops → TCG Optimizer

→ Dziękuję!

→ Aitäh!

→ Thank you!
Backup slides
FP per-op contribution

user-mode x86-on-x86
Qelt Instrumentation

- Fine-grained event subscription when guest code is translated
  - e.g. subscription to memory reads in Pin vs Qelt:

```c
VOID Instruction(INS ins)
{
    if (INS_IsMemoryRead(ins))
        INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)MemCB, ...);
}

VOID Trace(TRACE trace, VOID *v)
{
    for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl))
        for (INS ins = BBL_InsHead(bbl); INS_Valid(ins); ins = INS_Next(ins))
            Instruction(ins);
}
```

```c
static void vcpu_tb_trans(qemu_plugin_id_t id, unsigned int cpu_index, struct qemu_plugin_tb *tb)
{
    size_t n = qemu_plugin_tb_n_insns(tb);
    size_t i;

    for (i = 0; i < n; i++) {
        struct qemu_plugin_insn *insn = qemu_plugin_tb_get_insn(tb, i);

        qemu_plugin_register_vcpu_mem_cb(insn, vcpu_mem, QEMU_PLUGIN_CB_NO_REGS, QEMU_PLUGIN_MEM_R);
    }
} 2.3
```
Instrumentation overhead
user-mode, x86_64-on-x86_64

• Typical overhead
  ▪ Preemptive injection of instrumentation has negligible overhead

• Direct callbacks
  ▪ Better than going via a helper (that iterates over a list) due to higher cache locality
All techniques put together

user-mode x86_64-on-x86_64. Baseline: QEMU v3.1.0
CactusADM: TLB resizing doesn't kick in often enough (we only do it on TLB flushes)
SoftMMU overhead
lower is better

CactusADM: TLB resizing doesn't kick in often enough (we only do it on TLB flushes)
SoftMMU using shadow page tables[^]

Before:
softMMU requires many insns

Generated host code

```
ldr sp, [pc, #4] ; @ pc = 0x1000c

mov %0x1000c,%ebp
get target address
mov %ebp,%edi
lea 0x3(%ebp),%esi
shr $0x5,%edi
and $0xffffffff, %esi
lea 0x2c90(%r14,%rdi,1),%rdi
and hash check entry
jmp 0x1fd913745f0
add 0x10(%rdi),%rsi
mov (%rsi),%ebp
```

Advantages:

- High performance (almost 0 overhead for MMU emulation)
- Minimal modifications to QEMU compared to other options in the literature

Disadvantages:

- Requires dune*, which means QEMU must be statically compiled
- Cannot work when target address space => host address space

[^

cross-ISA examples (1)

x86-on-ppc64, make -j N inside a VM

aarch64-on-aarch64, Nbench FP

aarch64-on-x86, SPEC06fp
## cross-ISA examples (2)

### Ind. branches, RISC-V on x86, user-mode

<table>
<thead>
<tr>
<th>bench</th>
<th>before</th>
<th>after1</th>
<th>after2</th>
<th>after3</th>
<th>final speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes</td>
<td>1.12s</td>
<td>1.12s</td>
<td>1.10s</td>
<td>1.00s</td>
<td>1.12</td>
</tr>
<tr>
<td>bigint</td>
<td>0.78s</td>
<td>0.78s</td>
<td>0.78s</td>
<td>0.78s</td>
<td>1</td>
</tr>
<tr>
<td>dhryst</td>
<td>0.96s</td>
<td>0.97s</td>
<td>0.49s</td>
<td>0.49s</td>
<td>1.9591837</td>
</tr>
<tr>
<td>miniz</td>
<td>1.94s</td>
<td>1.94s</td>
<td>1.88s</td>
<td>1.86s</td>
<td>1.0430108</td>
</tr>
<tr>
<td>norx</td>
<td>0.51s</td>
<td>0.51s</td>
<td>0.49s</td>
<td>0.48s</td>
<td>1.0625</td>
</tr>
<tr>
<td>primes</td>
<td>0.85s</td>
<td>0.85s</td>
<td>0.84s</td>
<td>0.84s</td>
<td>1.0119048</td>
</tr>
<tr>
<td>qsort</td>
<td>4.87s</td>
<td>4.88s</td>
<td>1.86s</td>
<td>1.86s</td>
<td>2.6182796</td>
</tr>
<tr>
<td>sha512</td>
<td>0.76s</td>
<td>0.77s</td>
<td>0.64s</td>
<td>0.64s</td>
<td>1.1875</td>
</tr>
</tbody>
</table>

### Ind. branches, RISC-V on x86, full-system

<table>
<thead>
<tr>
<th>bench</th>
<th>before</th>
<th>after1</th>
<th>after2</th>
<th>after3</th>
<th>final speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes</td>
<td>2.68s</td>
<td>2.54s</td>
<td>2.60s</td>
<td>2.34s</td>
<td>1.1452991</td>
</tr>
<tr>
<td>bigint</td>
<td>1.61s</td>
<td>1.56s</td>
<td>1.55s</td>
<td>1.64s</td>
<td>0.98170732</td>
</tr>
<tr>
<td>dhryst</td>
<td>1.78s</td>
<td>1.67s</td>
<td>1.25s</td>
<td>1.24s</td>
<td>1.4354839</td>
</tr>
<tr>
<td>miniz</td>
<td>3.53s</td>
<td>3.35s</td>
<td>3.28s</td>
<td>3.35s</td>
<td>1.0537313</td>
</tr>
<tr>
<td>norx</td>
<td>1.13s</td>
<td>1.09s</td>
<td>1.07s</td>
<td>1.06s</td>
<td>1.0660377</td>
</tr>
<tr>
<td>primes</td>
<td>15.37s</td>
<td>15.41s</td>
<td>15.20s</td>
<td>15.37s</td>
<td>1</td>
</tr>
<tr>
<td>qsort</td>
<td>7.20s</td>
<td>6.71s</td>
<td>3.85s</td>
<td>3.96s</td>
<td>1.8181818</td>
</tr>
<tr>
<td>sha512</td>
<td>1.07s</td>
<td>1.04s</td>
<td>0.90s</td>
<td>0.90s</td>
<td>1.1888889</td>
</tr>
</tbody>
</table>