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

Emilio G. Cota
Luca P. Carloni

Columbia University

VEE'19
April 14, 2019
Providence, RI

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

	Speed	**Cross-ISA**	**Full-system**
DynamoRIO	✔ Fast	✘	✘
Pin	✔ Fast	✘	✘
QEMU (& derivatives)	✘ Slow	✔	✔

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**

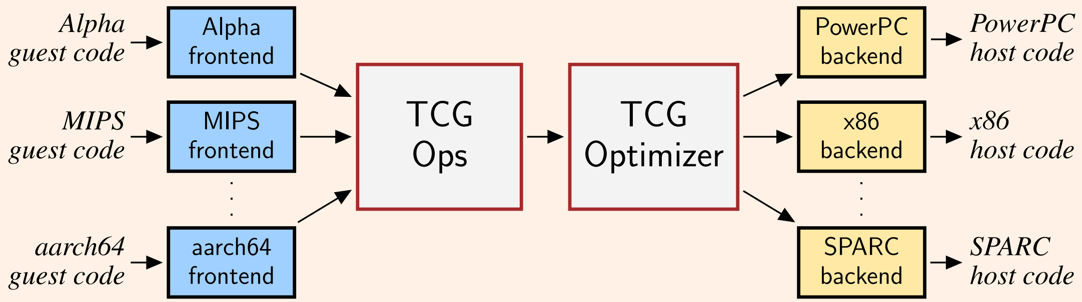
QEMU*

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)



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

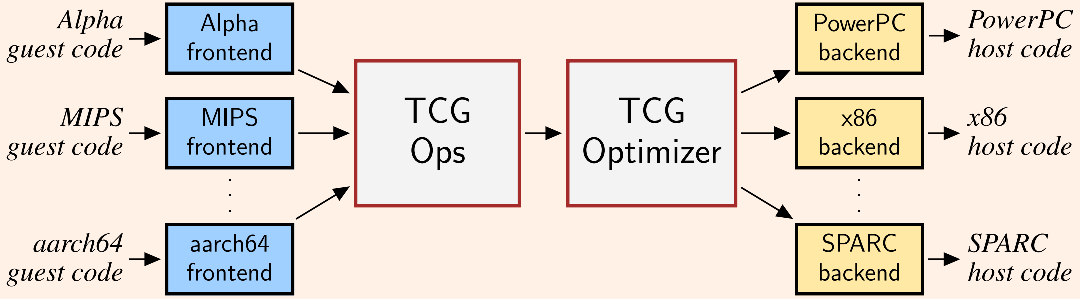
QEMU*

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

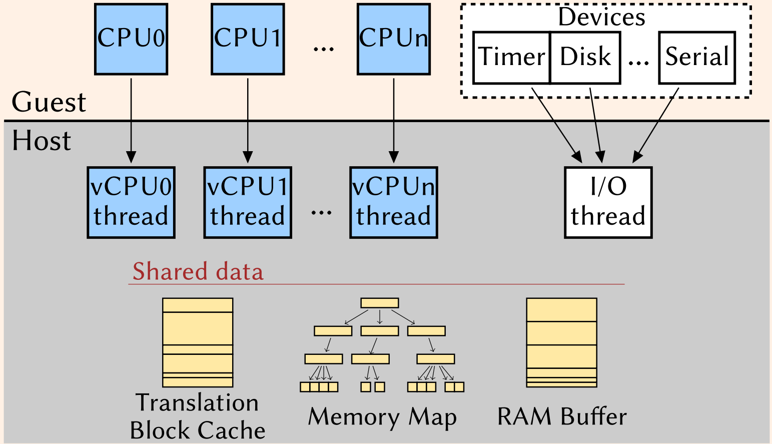
QEMU baseline

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



[*] Cota, Bonzini, Bennée, Carloni. "Cross-ISA Machine Emulation for Multicores", CGO, 2017

Qelt's contributions

Emulation Speed

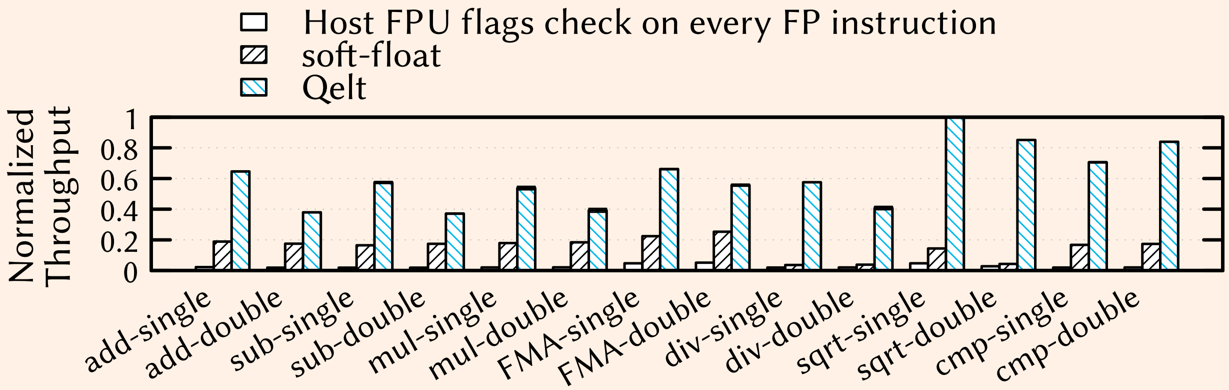
1. Correct cross-ISA **FP emulation** using the host FPU
2. Integration of two state-of-the-art optimizations:
   - indirect branch handling
   - dynamic sizing of the **software TLB**
3. Make the DBT engine **scale** under heavy **code *translation***
   - Not just during ***execution***

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



baseline (incorrect): always uses the host FPU and never reads excp. flags

- 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

1. Cross-ISA FP Emulation

```
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);
    }
  }
soft_fp:
  return soft_float64_mul(a, b, st);
}
```

Common case:

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

How common?

99.18%

of FP instructions in SPECfp06

.. and similarly for 32/64b + , - , × , ÷, √, ==

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

[A] Hong, Hsu, Chou, Hsu, Liu, Wu. "Optimizing Control Transfer and Memory Virtualization in Full System Emulators", ACM TACO, 2015

[B] Tong, Koju, Kawahito, Moshovos. "Optimizing memory translation emulation in full system emulators", ACM TACO, 2015

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*

[A] Hong, Hsu, Chou, Hsu, Liu, Wu. "Optimizing Control Transfer and Memory Virtualization in Full System Emulators", ACM TACO, 2015
[B] Tong, Koju, Kawahito, Moshovos. "Optimizing memory translation emulation in full system emulators", ACM TACO, 2015

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)

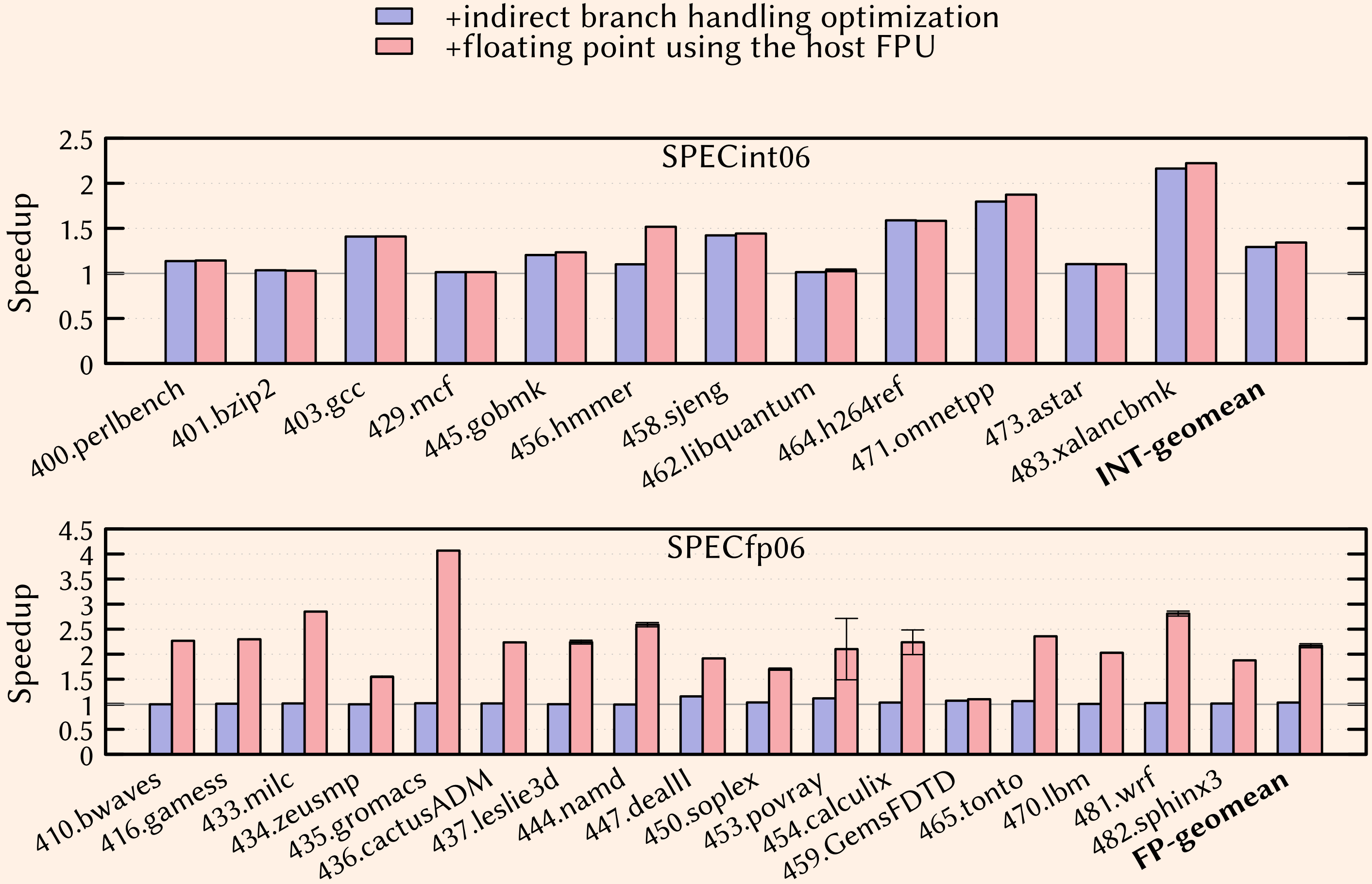
- Virtual memory is emulated with a *software TLB*
- Tong et al. [B] present TLB resizing based on TLB use rate at flush time
    - We improve on it by incorporating **history to shrink less aggressively**
        - Rationale: if a memory-hungry process was just scheduled out, it is likely that it will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

[A] Hong, Hsu, Chou, Hsu, Liu, Wu. "Optimizing Control Transfer and Memory Virtualization in Full System Emulators", ACM TACO, 2015

[B] Tong, Koju, Kawahito, Moshovos. "Optimizing memory translation emulation in full system emulators", ACM TACO, 2015

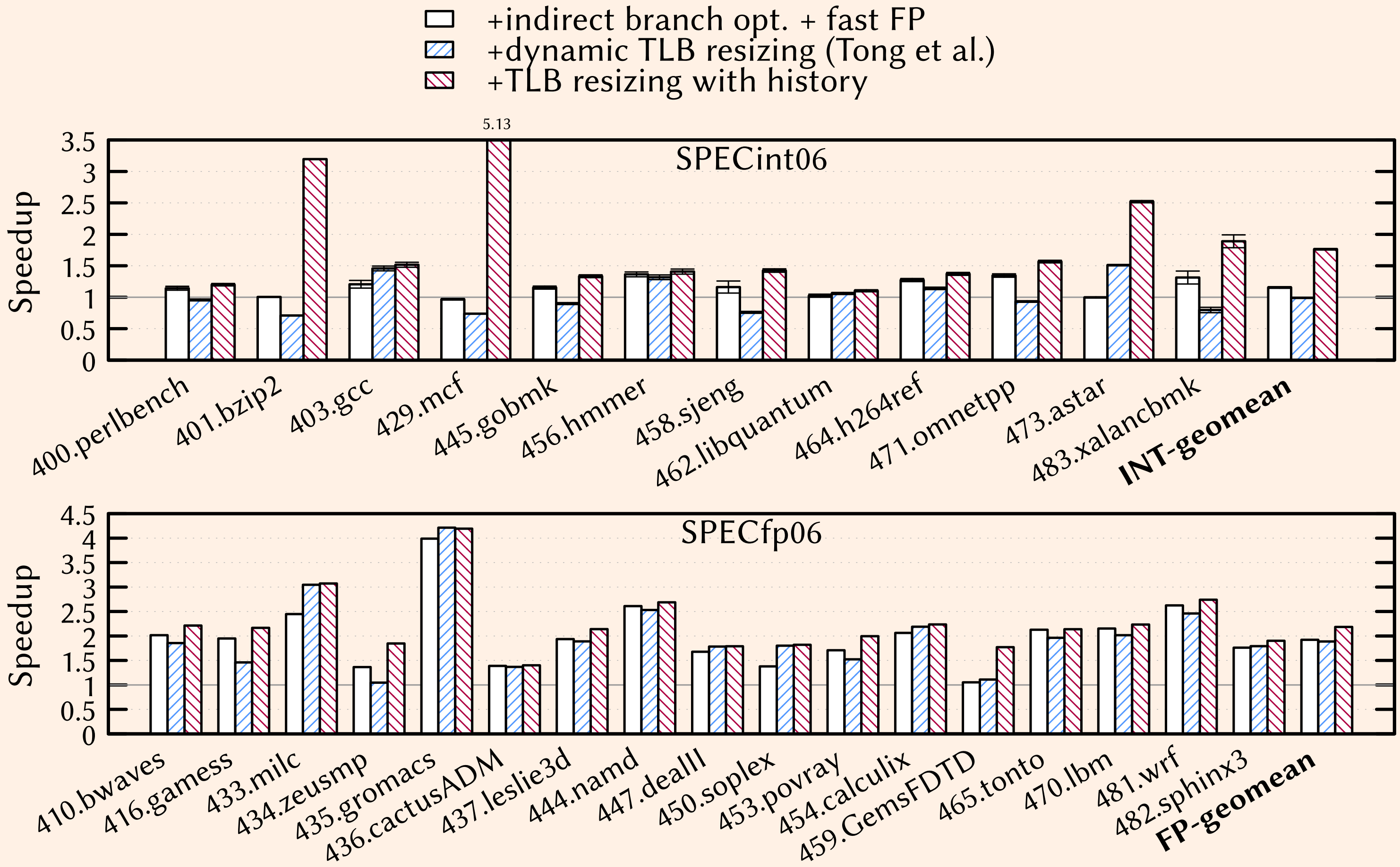
Indirect branch + FP improvements

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



TLB resizing

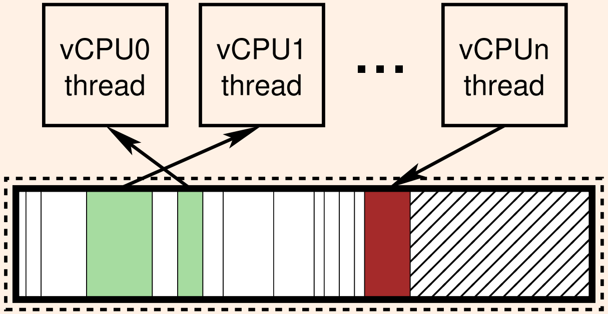
full-system x86_64-on-x86_64. Baseline: QEMU v3.1.0



- +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

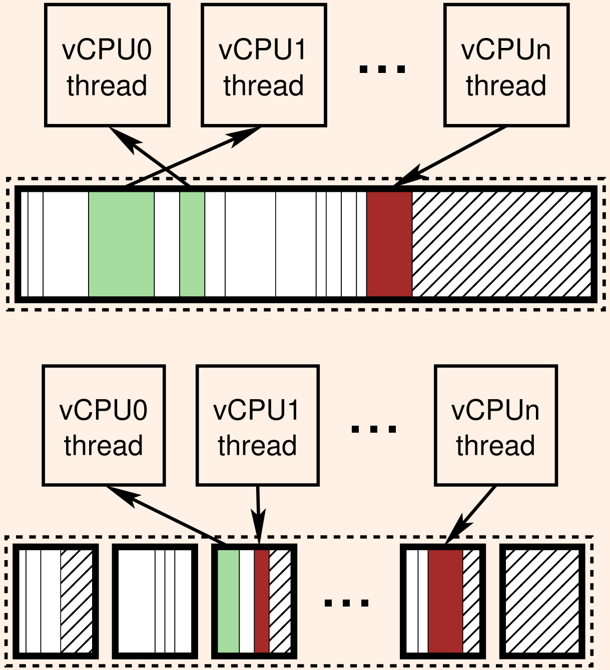


Monolithic TB cache (QEMU)

- ✅ 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)

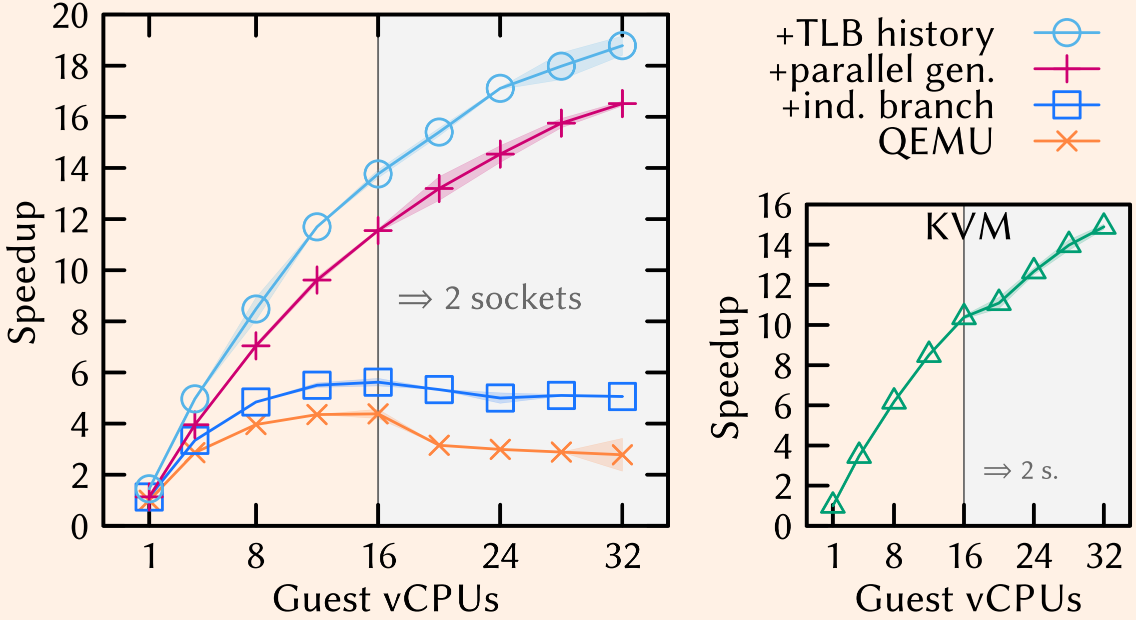
- Parallel TB execution
- Parallel TB generation (one region per vCPU)

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

Parallel code translation

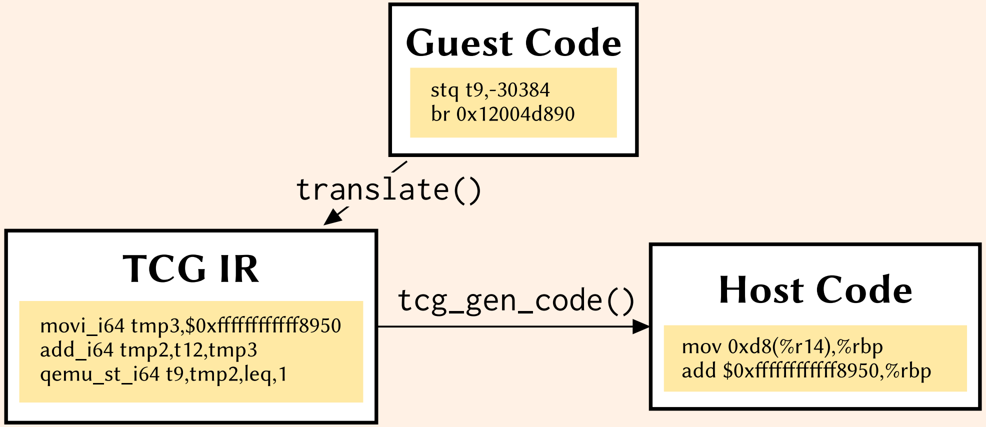
Guest VM performing parallel compilation of Linux kernel modules, x86_64-on-x86_64

- 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



[*] Cota, Bonzini, Bennée, Carloni. "Cross-ISA Machine Emulation for Multicores", CGO, 2017

4. Cross-ISA Instrumentation



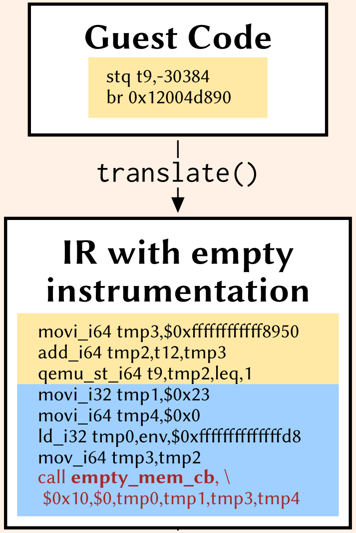
QEMU cannot instrument the guest

- 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

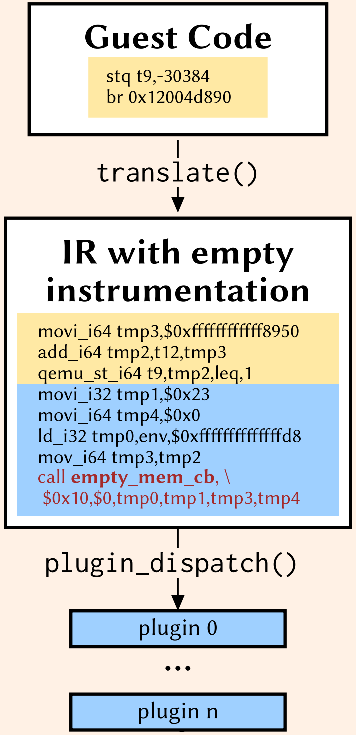
- Qelt first adds "empty" instrumentation in TCG, QEMU's IR



4. Cross-ISA Instrumentation

Instrumentation with Qelt

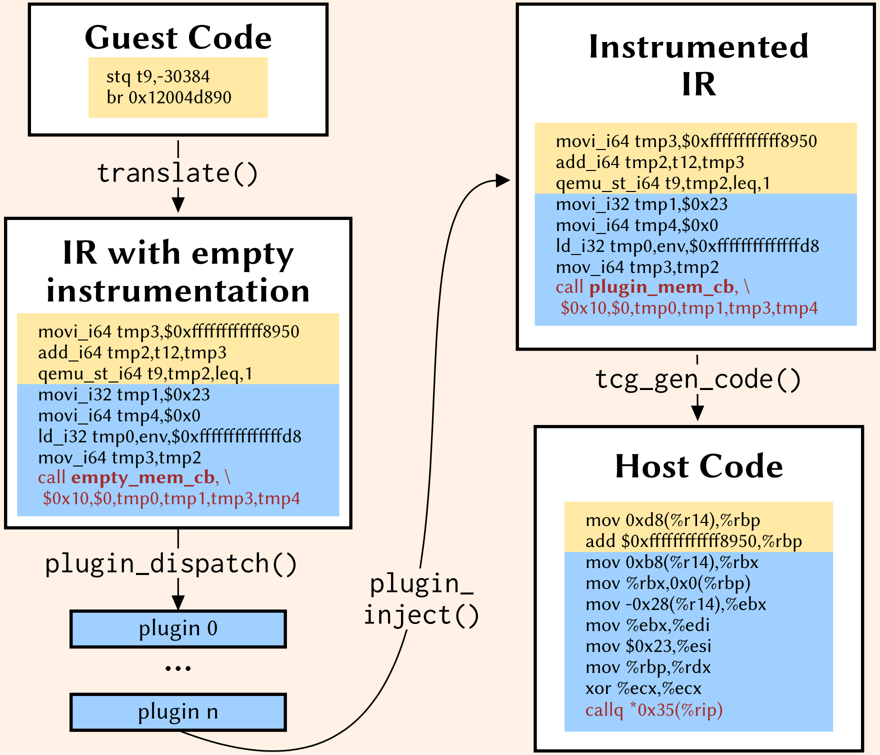
- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
  - They can use a decoder; Qelt only sees opaque insns/accesses



4. Cross-ISA Instrumentation

Instrumentation with Qelt

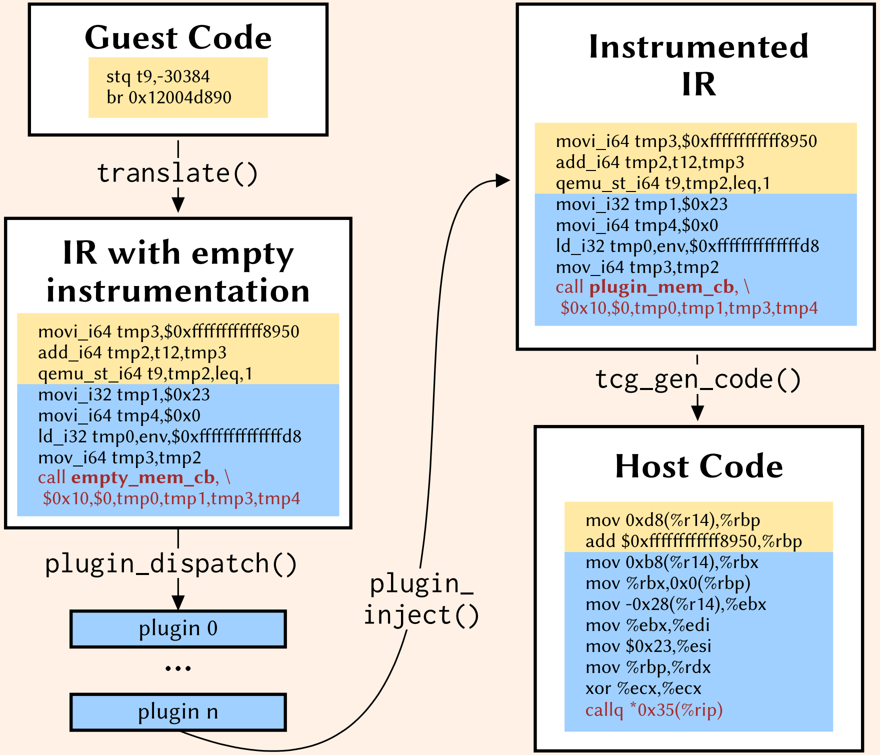
- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
    - They can use a decoder; Qelt only sees opaque insns/accesses
- Qelt then substitutes "empty" instrumentation with the actual calls to plugin *callbacks* (or removes it if not needed)



4. Cross-ISA Instrumentation

Instrumentation with Qelt

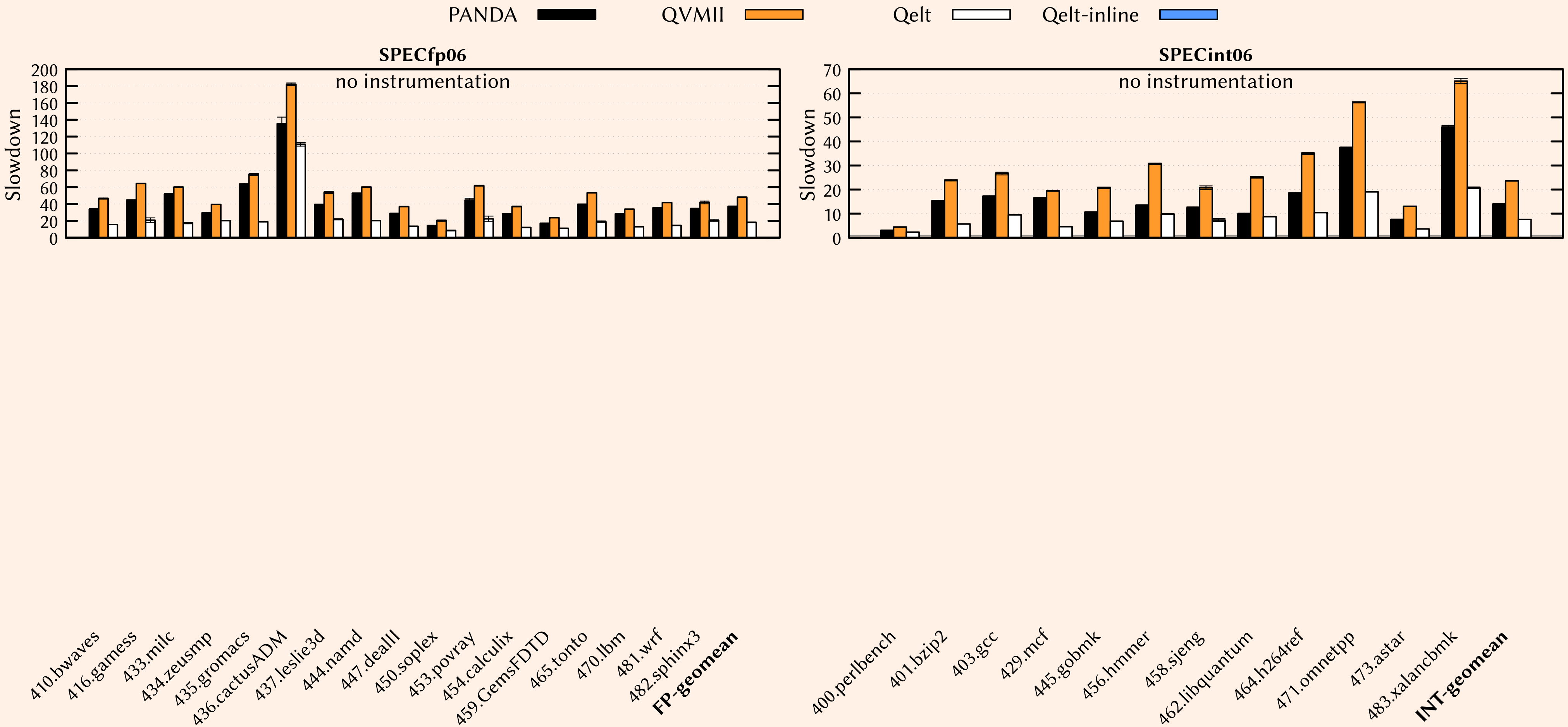
- Qelt first adds "empty" instrumentation in TCG, QEMU's IR
- Plugins subscribe to events in a TB
  - They can use a decoder; Qelt only sees opaque insns/accesses
- Qelt then substitutes "empty" instrumentation with the actual calls to plugin ***callbacks*** (or removes it if not needed)



- Other features (see paper): *direct* callbacks, inlining, helper instrumentation

Full-system instrumentation

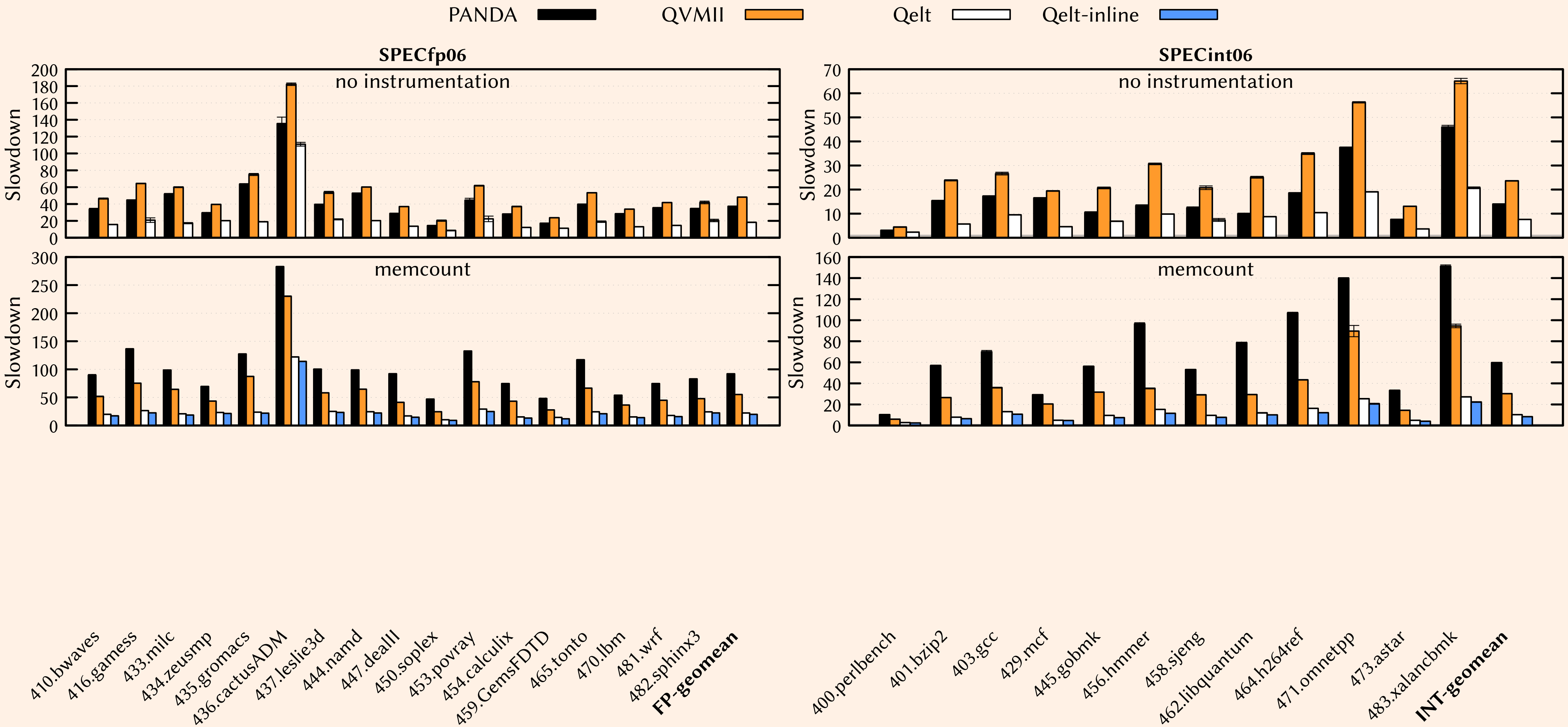
x86_64-on-x86_64 (**lower is better**). Baseline: KVM



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

Full-system instrumentation

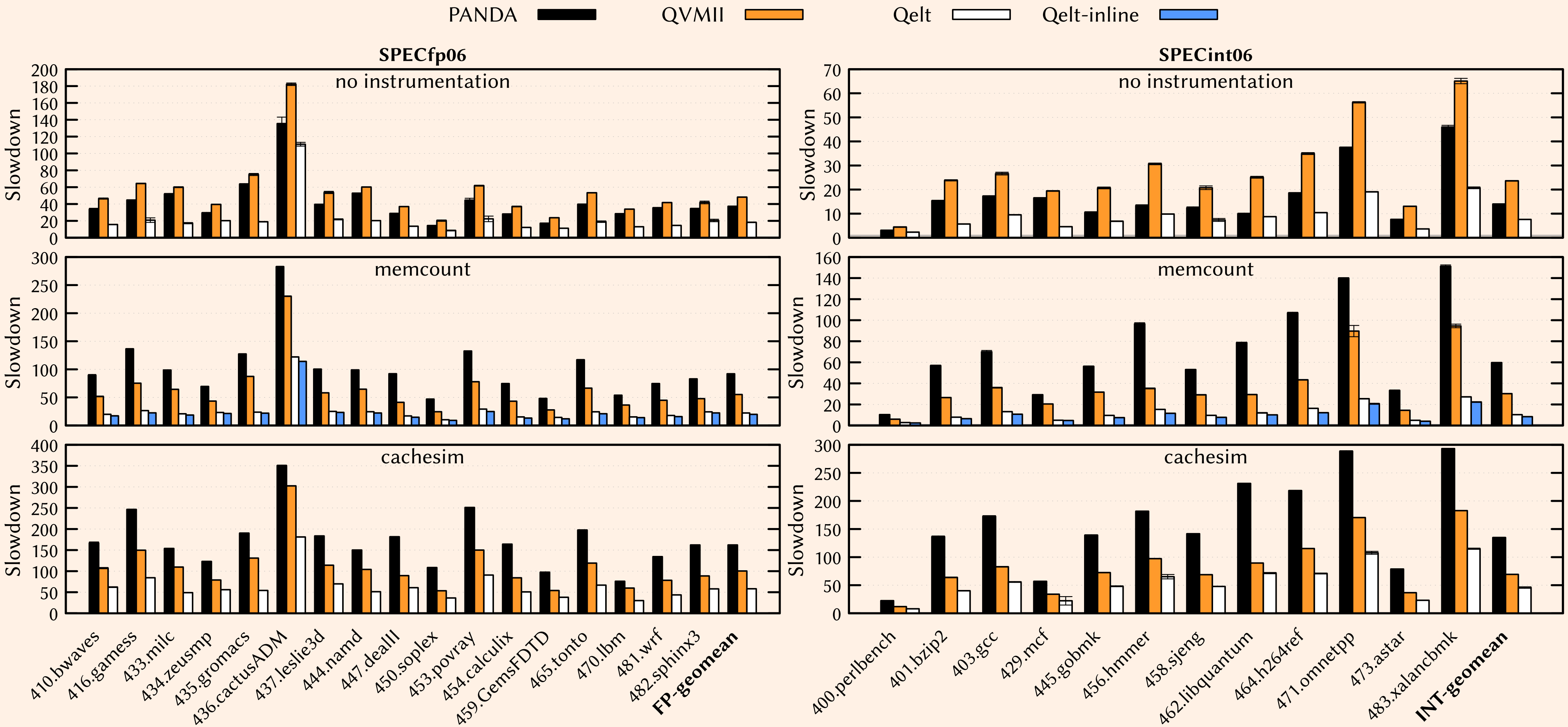
x86_64-on-x86_64 (**lower is better**). Baseline: KVM



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

Full-system instrumentation

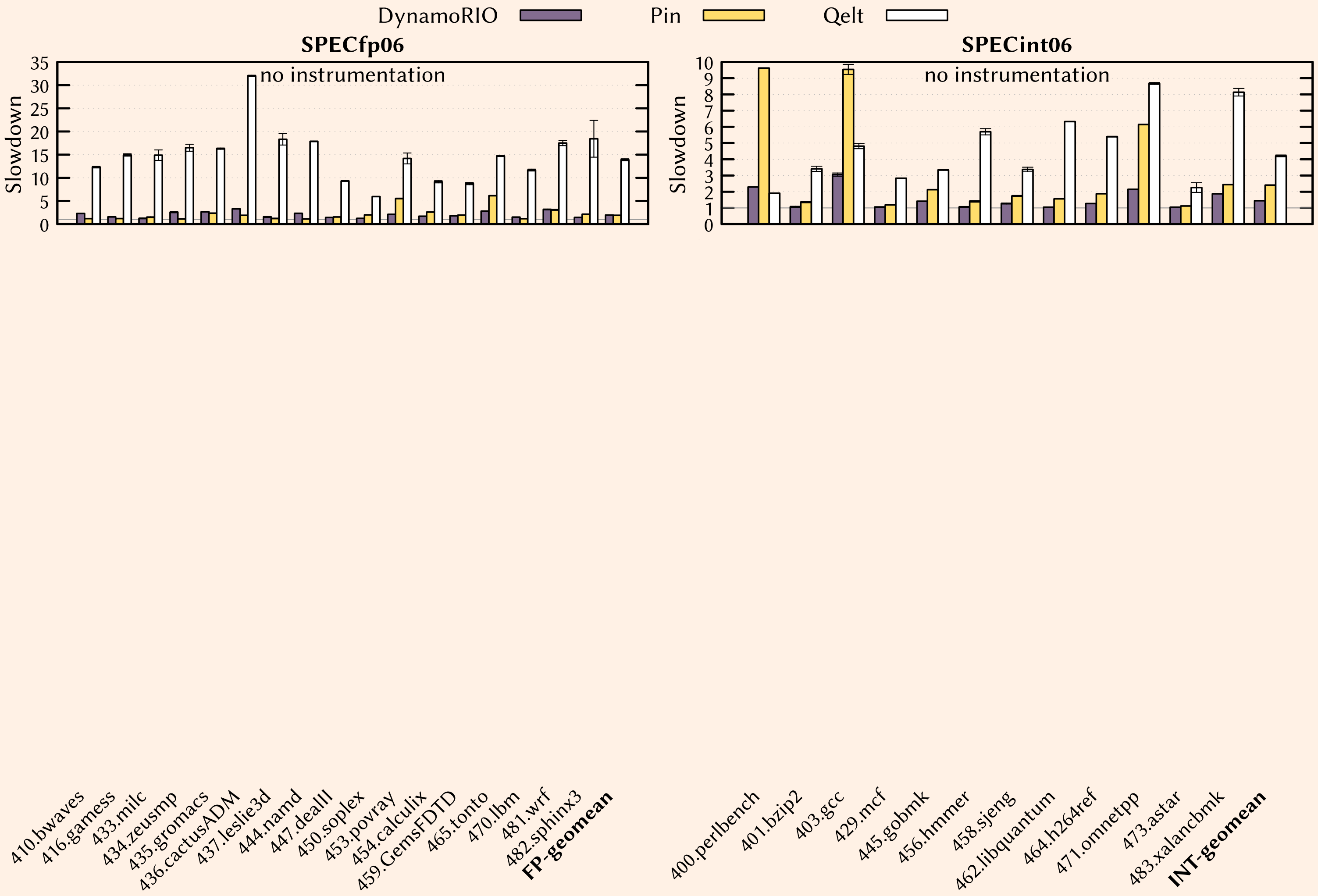
x86_64-on-x86_64 (**lower is better**). Baseline: KVM



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

User-mode instrumentation

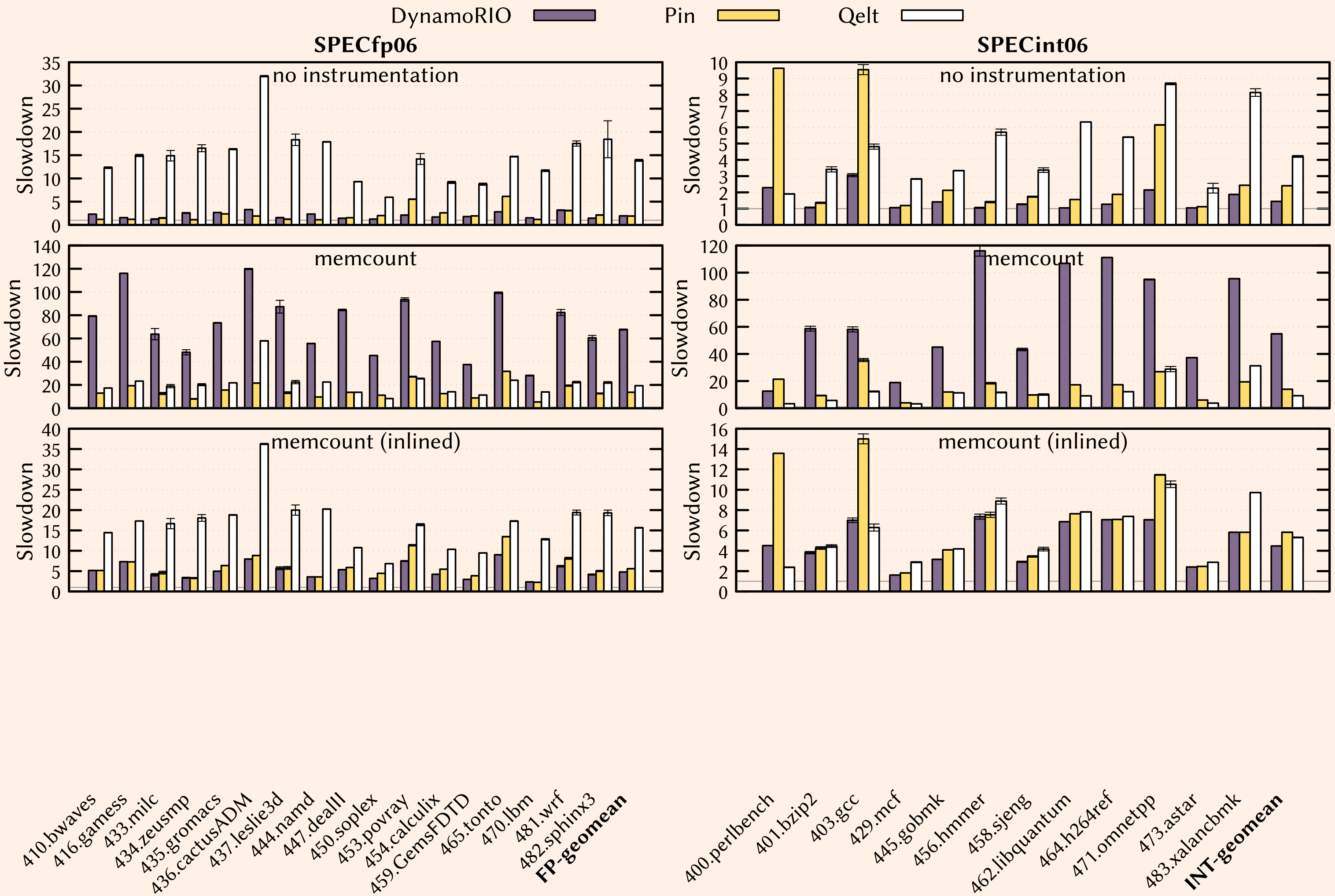
x86_64-on-x86_64 (**lower is better**). Baseline: native



- Qelt has narrowed the gap with Pin/DRIO for no instr., although for FP the gap is still significant

User-mode instrumentation

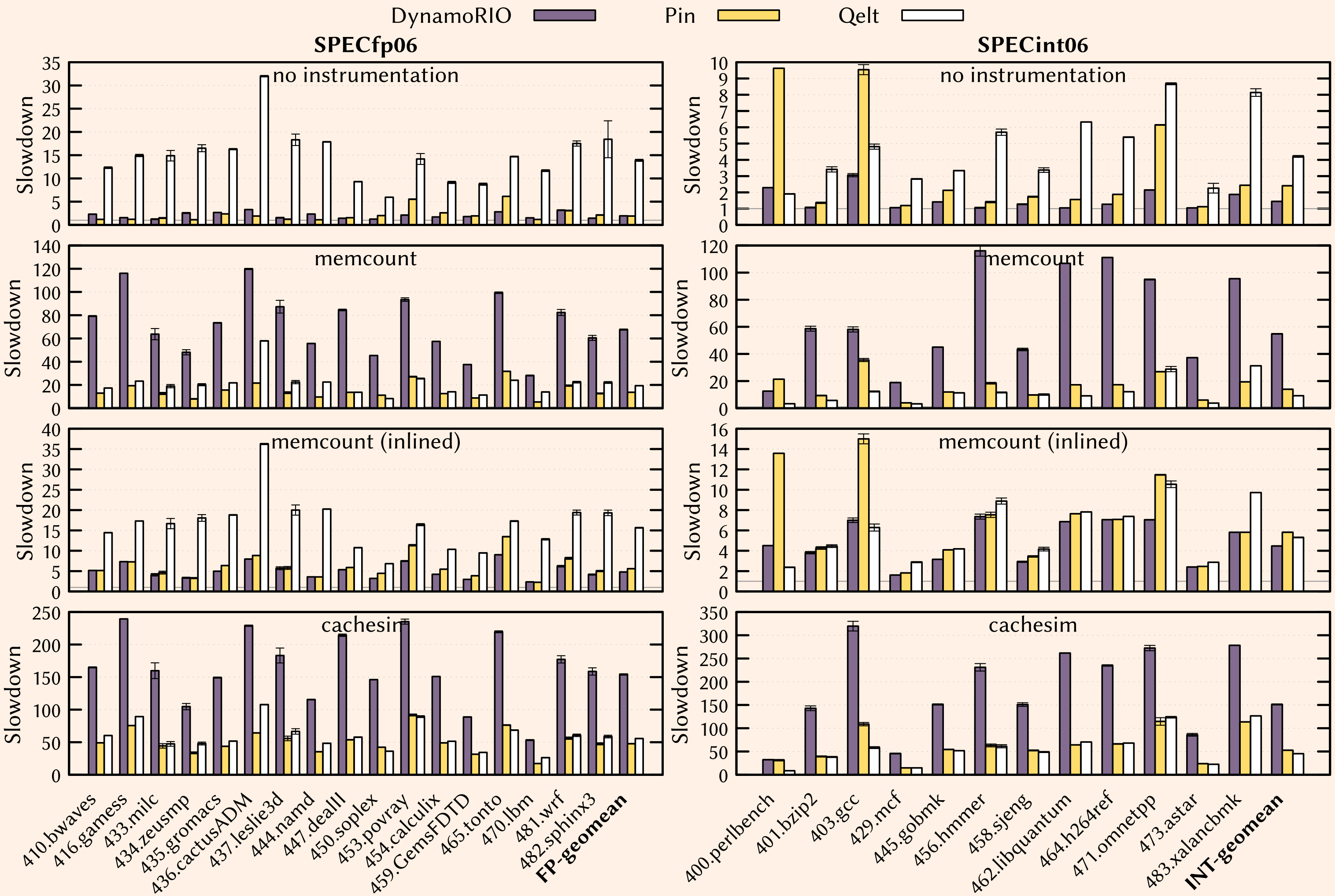
x86_64-on-x86_64 (**lower is better**). Baseline: native



- 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.

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

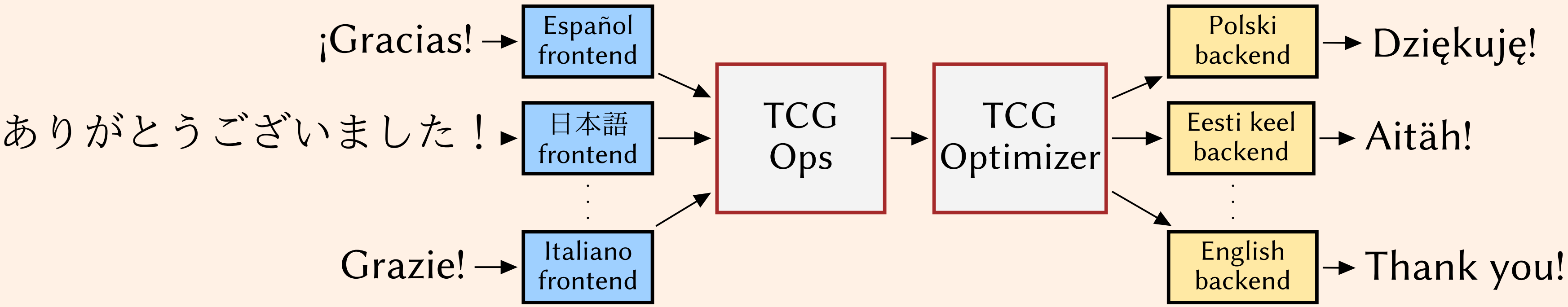
Conclusions

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

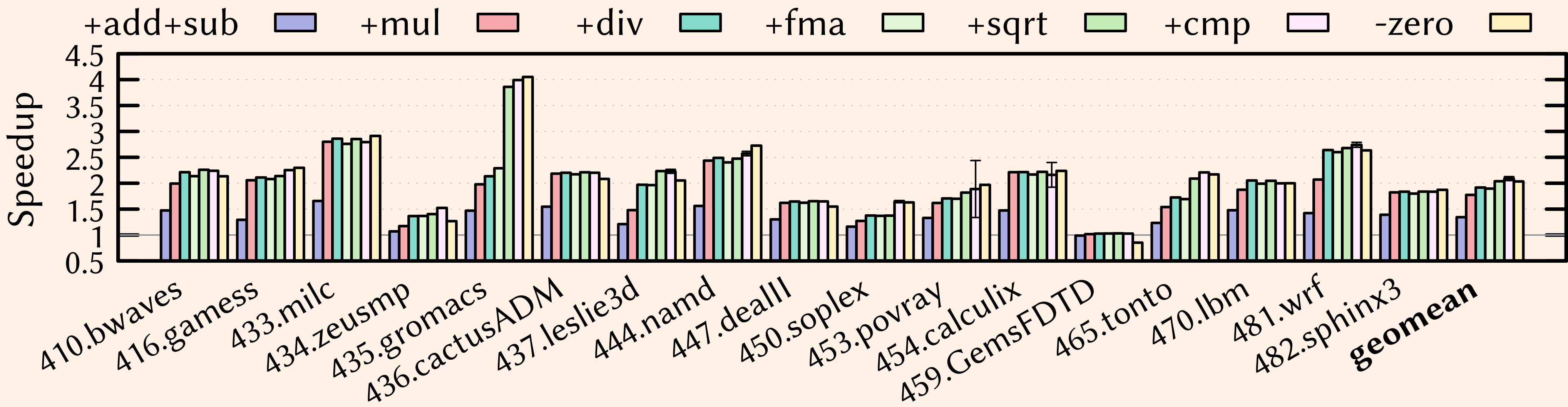


¡Gracias!
ありがとうございました！
Grazie!
Español frontend
日本語 frontend
Italiano frontend
TCG Ops
TCG Optimizer
Polski backend
Eesti keel backend
English backend
Dziękuję!
Aitäh!
Thank you!

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:

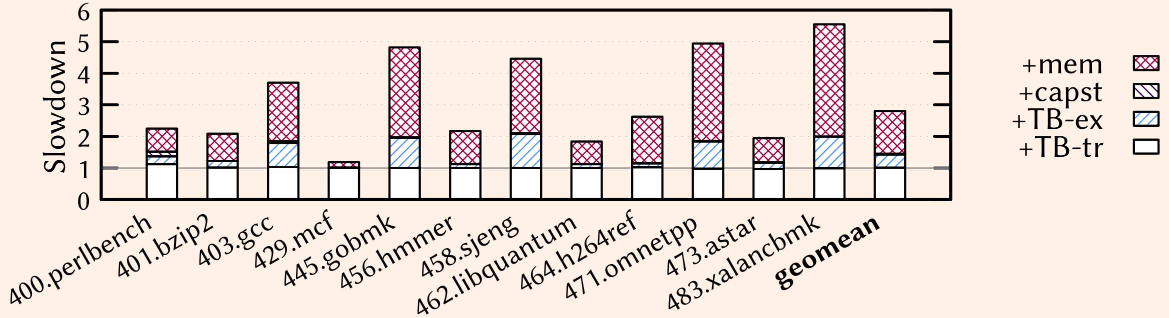
```
VOID Instruction(INS ins)
{
        if (INS_IsMemoryRead(ins))
                INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)MemCB, ...);
}
VOID Trace(TRACE trace, VOID *v)
{
        for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl))
                for (INS ins = BBL_InsHead(bbl); INS_Valid(ins); ins = INS_Next(ins))
                        Instruction(ins);
}
```

```
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);
        }
```

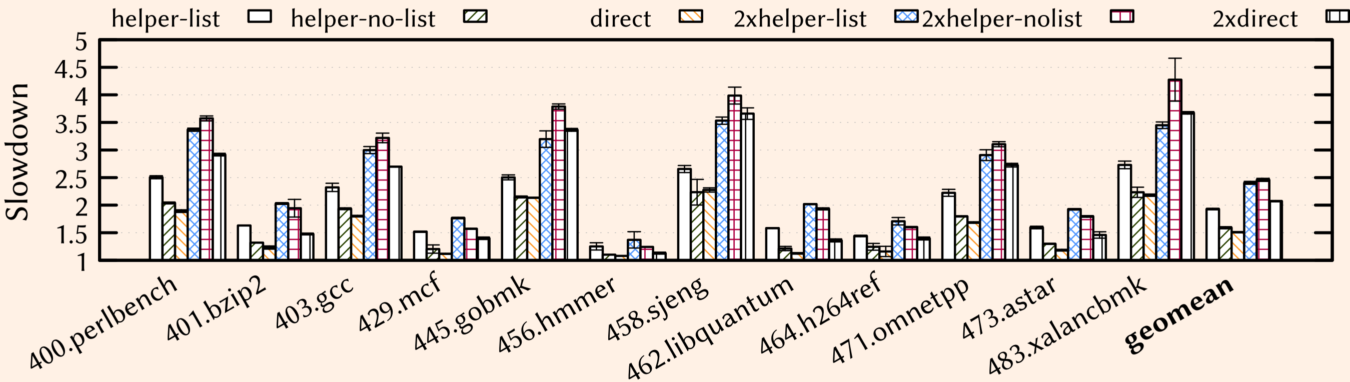
Instrumentation overhead

user-mode, x86_64-on-x86_64

- Typical overhead
  - Preemptive injection of instrumentation has negligible overhead

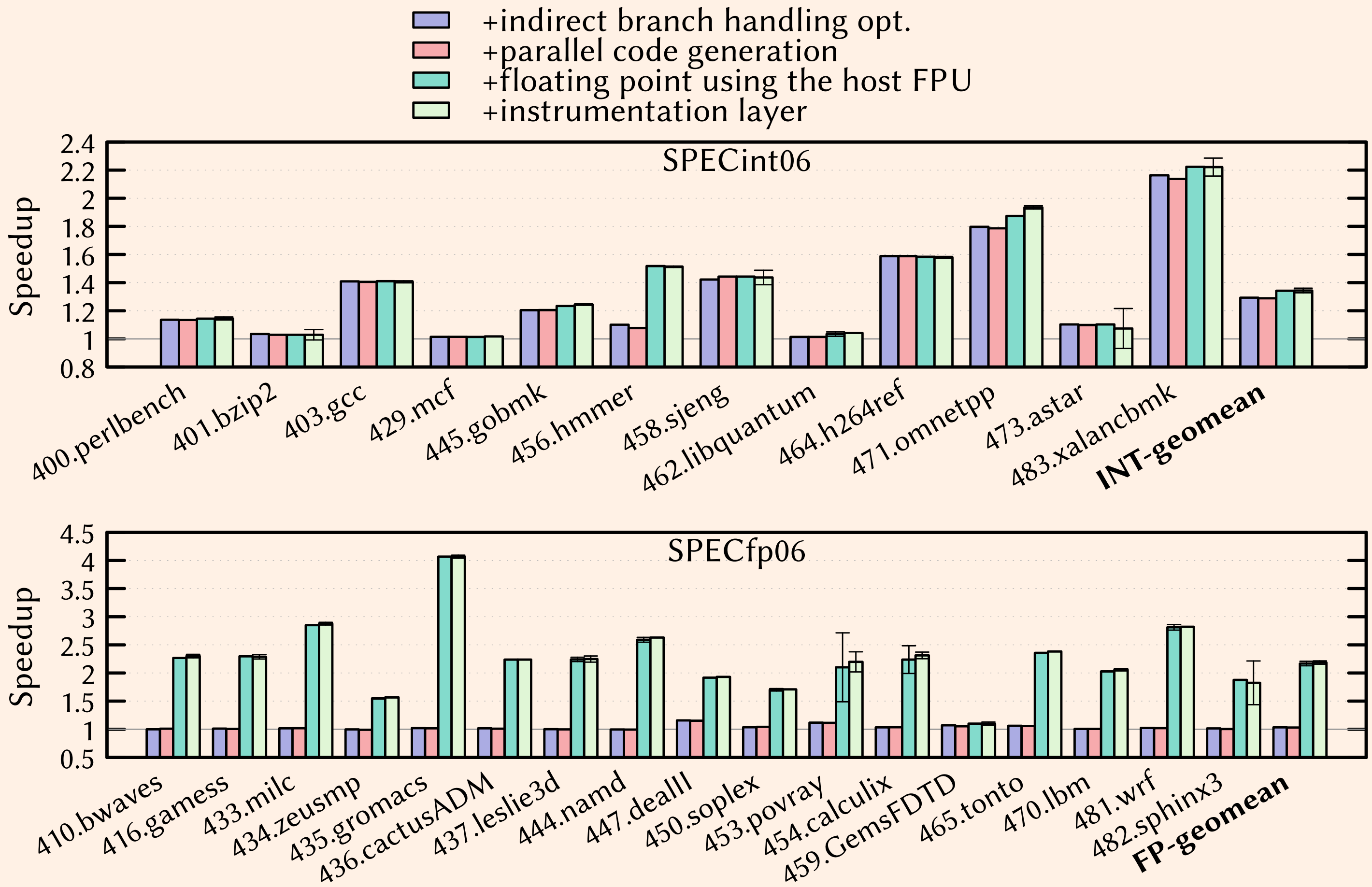


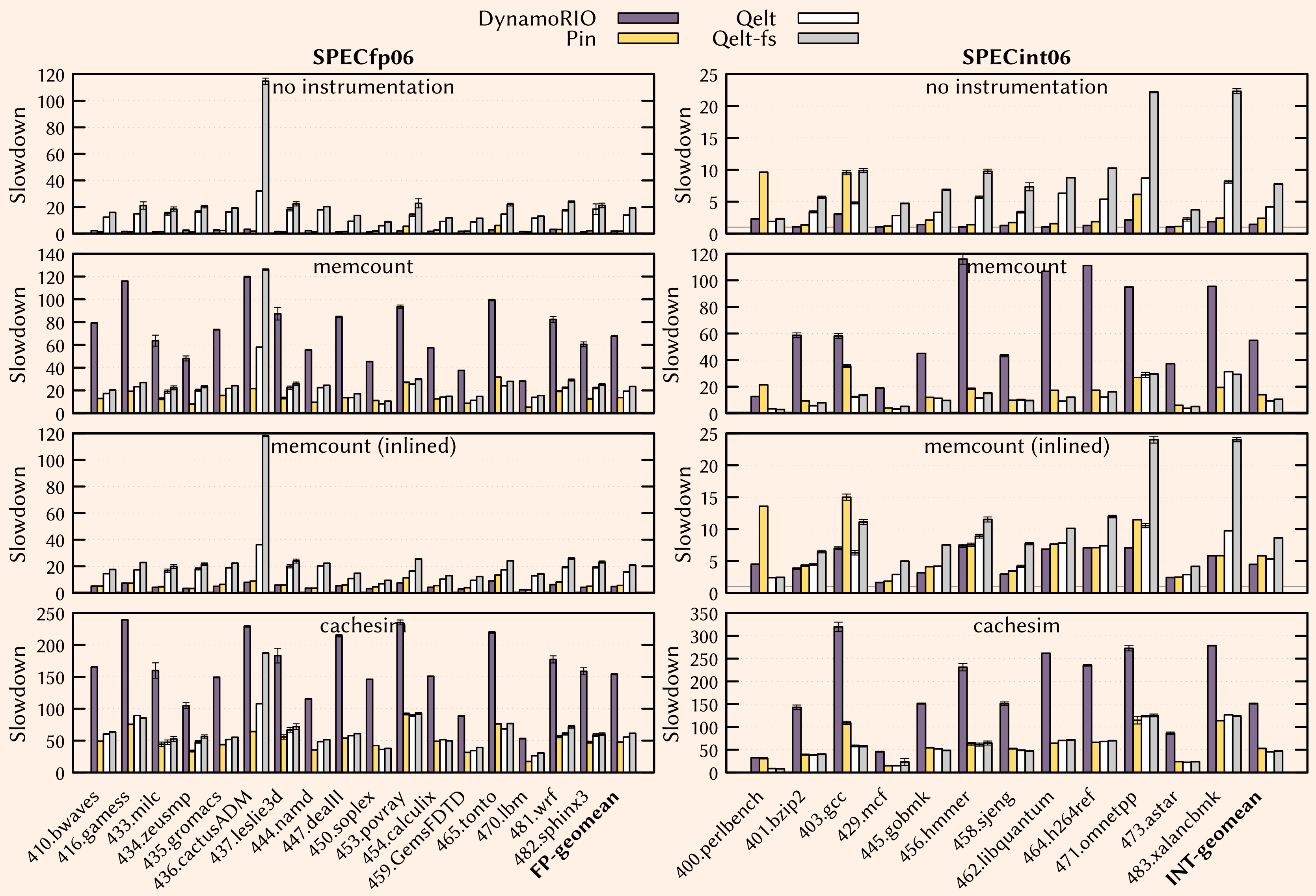
- Direct callbacks
  - Better than going via a helper (that iterates over a list) due to higher cache locality



All techniques put together

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

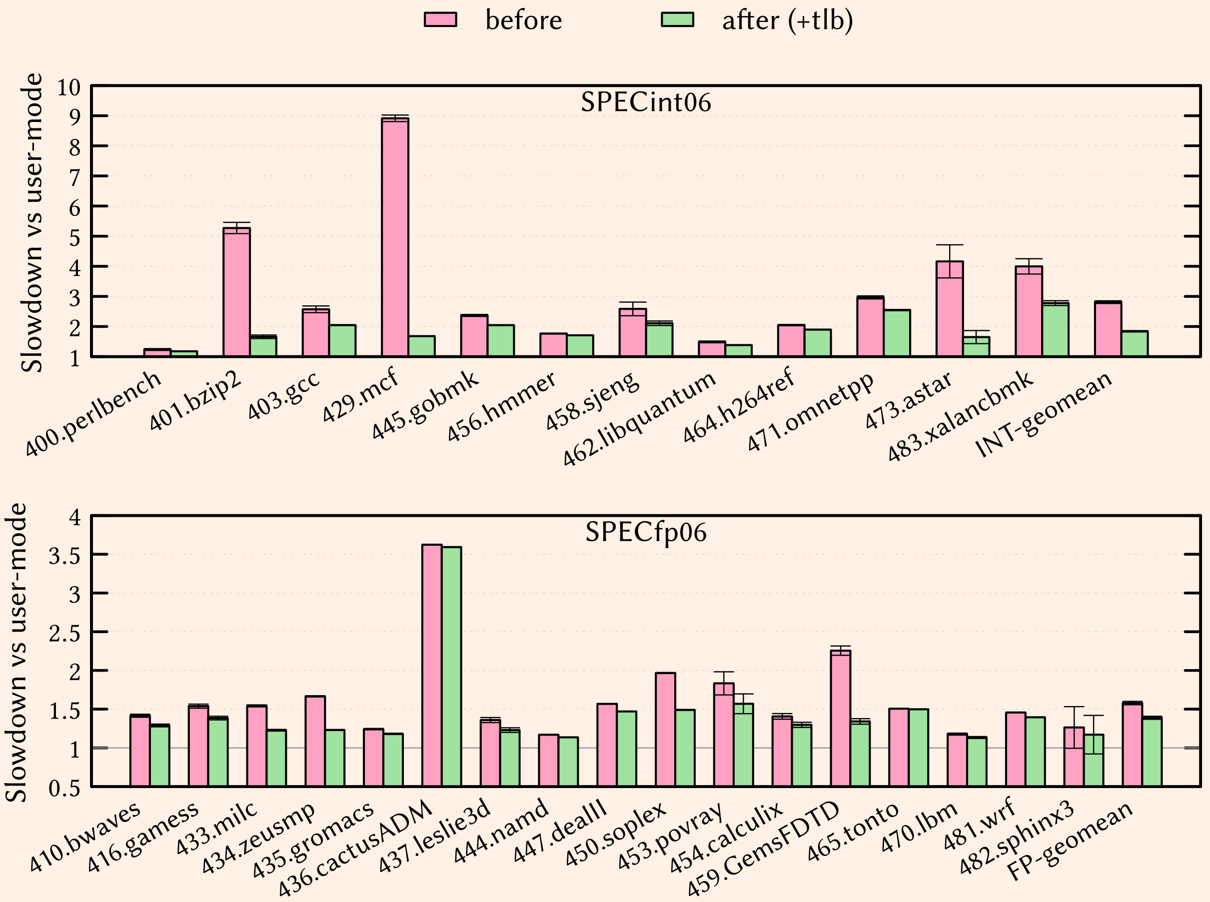




CactusADM: TLB resizing doesn't kick in often enough (we only do it on TLB flushes)

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

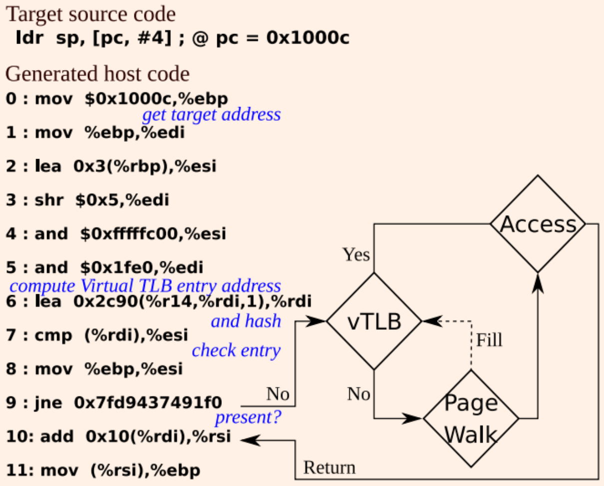


Fig. 1. QEMU target memory accesses translation

after:
only 2 insns thanks to
shadow page tables

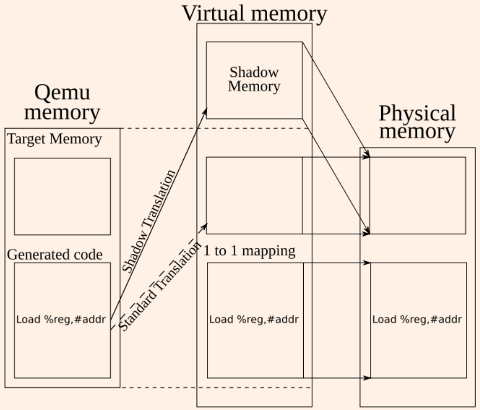


Fig. 2. Memory Layout

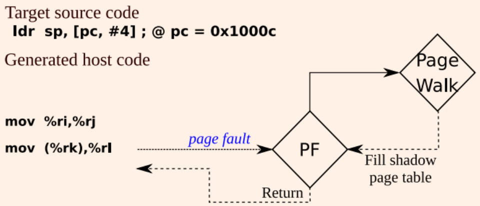


Fig. 3. QEMU target memory access with our solution

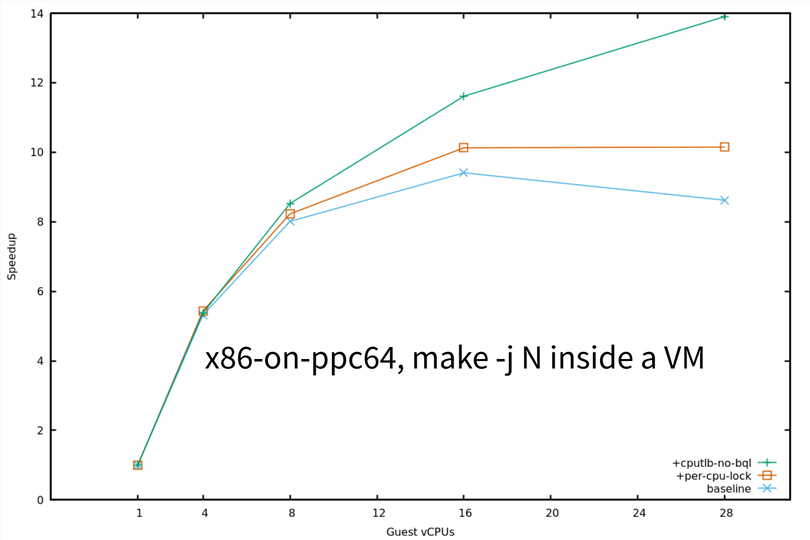
Advantages:

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

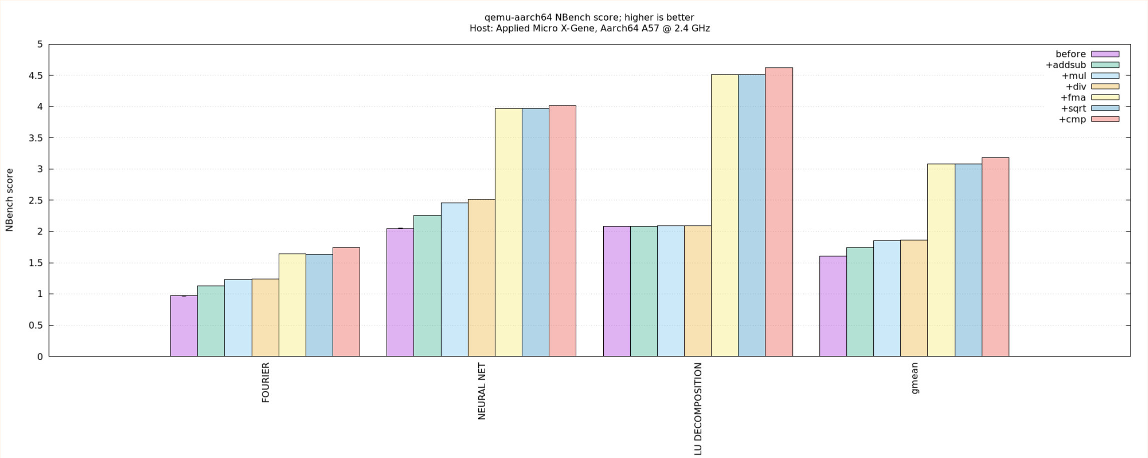
Disadvantages:

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

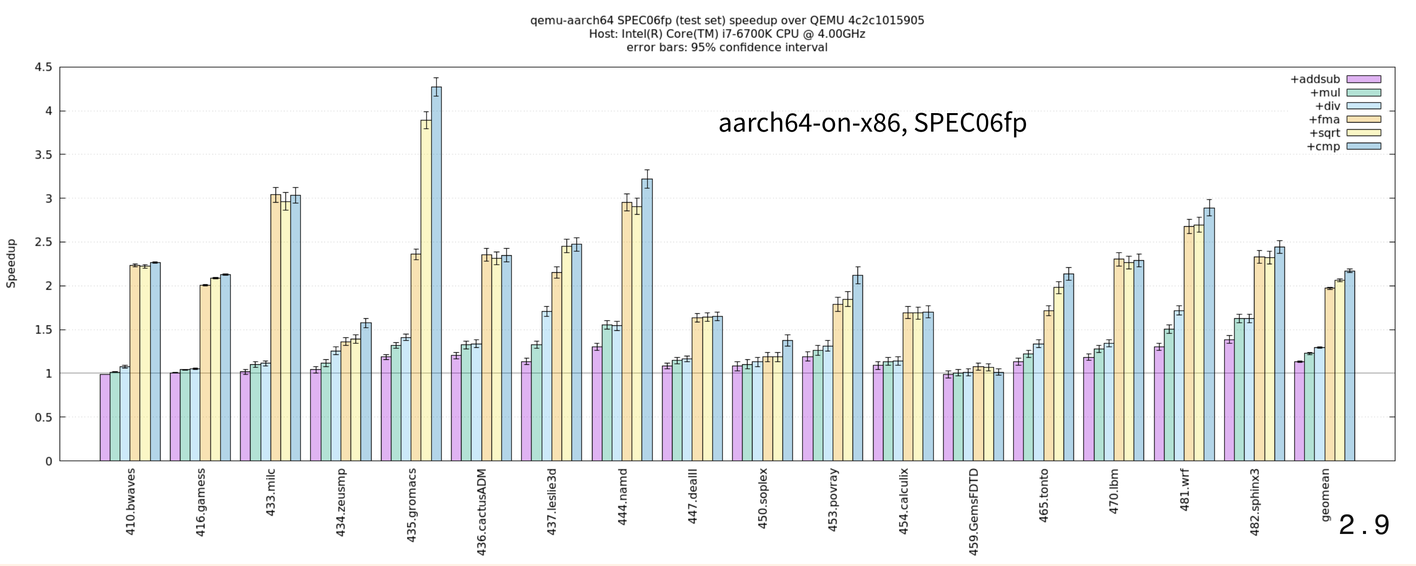
[^] Faravelon, Gruber, Pétrot. "Optimizing memory access performance using hardware assisted virtualization in retargetable dynamic binary translation. Euromicro Conference on Digital System Design (DSD), 2017.
[*] Belay, Bittau, Mashtizadeh, Terei, Mazieres, Kozyrakis. "Dune: Safe user-level access to privileged cpu features." OSDI, 2012



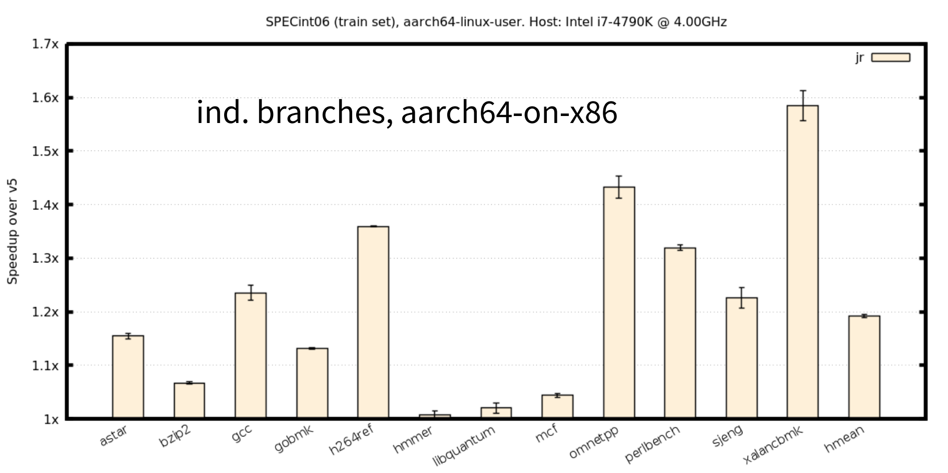
aarch64-on-aarch64, Nbench FP

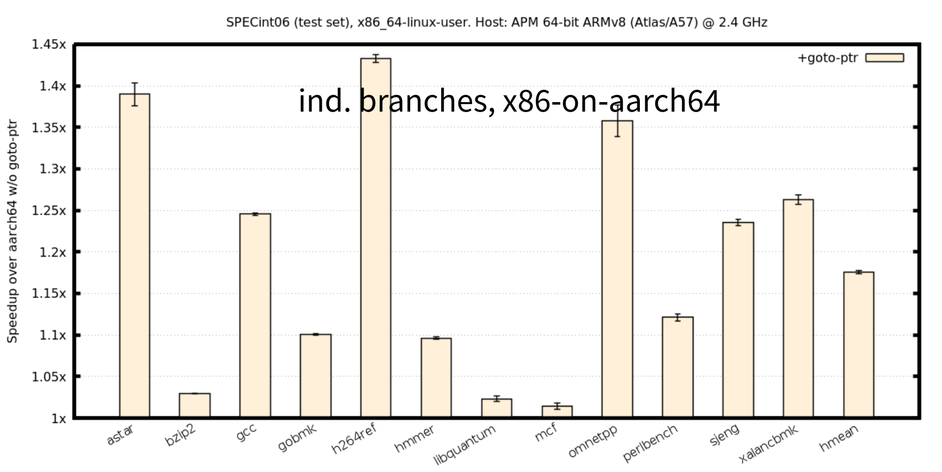


cross-ISA examples (1)



cross-ISA examples (2)





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

bench	before	after1	after2	after3	final_speedup
aes	1.12s	1.12s	1.10s	1.00s	1.12
bigint	0.78s	0.78s	0.78s	0.78s	1
dhryst	0.96s	0.97s	0.49s	0.49s	1.9591837
miniz	1.94s	1.94s	1.88s	1.86s	1.0430108
norx	0.51s	0.51s	0.49s	0.48s	1.0625
primes	0.85s	0.85s	0.84s	0.84s	1.0119048
qsort	4.87s	4.88s	1.86s	1.86s	2.6182796
sha512	0.76s	0.77s	0.64s	0.64s	1.1875

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

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