HKG18-TR08: Upstreaming SVE in QEMU

Alex Bennée and Richard Henderson
Contents

● Introductions
  ○ Who we are
  ○ What QEMU is

● The QEMU Project
  ○ Development Process
  ○ Upstreaming Criteria

● SVE Work
  ○ Native Vectors for TCG
  ○ ARMv8.2 FP16
  ○ Remaining Prerequisites
  ○ SVE instructions

● Demo!
Who are we?

Alex Bennée
Linaro Virtualization Engineer
alex.bennee@linaro.org
IRC: ajb-linaro/stsqaud

Richard Henderson
Linaro Virtualization Engineer
richard.henderson@linaro.org
IRC: rth
What is QEMU?

“What QEMU is a generic and open source machine emulator and virtualizer”

● Device Emulation for HW virtualization
  ○ KVM
  ○ HAX
  ○ HVF
  ○ Xen

● Full System Emulation
  ○ 20-21 guest architectures
  ○ TCG JIT engine
  ○ 7 host architectures
QEMU’s Growth

Lines of Code
Steady Development

State of QEMU, Paolo Bonzini - KVM Forum 2017
QEMU Development Basics

- **Source Code**
  - GIT: https://git.qemu.org
  - Master repository with submodules
  - PULL Requests merged by Head Maintainer
    - Sub-maintainers send pull-requests via list

- **Discussion**
  - Main List: qemu-devel@nongnu.org
  - Sub-lists:
    - ARM specific: qemu-arm@nongnu.org
    - Security: qemu-security@nongnu.org
    - Trivial Patches: qemu-trivial@nongnu.org
  - IRC: #qemu on OFTC network

- **Documentation**
  - Source Tree: ./docs
  - Wiki: https://wiki.qemu.org/Main_Page
Maintainer/Contributor Retention since 2.0

State of QEMU, Paolo Bonzini - KVM Forum 2017
Upstreaming Criteria

● No Regressions
  ○ compiles!
  ○ make check
  ○ tested

● Acceptable Code
  ○ Follows CODING_STYLE
  ○ Signed-off-by: J Hacker <j.hacker@example.com>
  ○ Reviewed
  ○ Bisectable
ARM’s Scalable Vector Extensions

- New SIMD extension for Mobile to HPC
- IMPDEF Vector Size (128-2048* bit)
- Multiple Floating Point Precisions
  - N x 64 bit (double precision)
  - 2N x 32 bit (single precision)
  - 4N x 16 bit (half precision)
- For more details see:
  - ARM’s own blog post
  - The Challenge of SVE in QEMU (SFO17)
  - Vectors Meet Virtualization (FOSDEM18)
Tiny Code Generator (TCG)

Guest Instructions (ARM)

- `ldr q0, [x21, x0]`
- `ldr q1, [x20, x0]`
- `fmul v0.4s, v0.4s, v1.4s`
- `str q0, [x19, x0]`
- `add x0, x0, #0x10 (16)`
- `cmp x0, #0x400000`
- `b.ne loop`

TCG Micro Ops

- `mov i64 tmp2, x21`
- `mov i64 tmp3, x0`
- `add i64 tmp2, tmp2, tmp3`
- `mov i64 tmp7, $0x8`
- `add i64 tmp6, tmp2, tmp7`
- `qemu ld i64 tmp4, tmp2, leq, 0`
- `st i64 tmp4, env, $0x898`
- `...`

TCG Disas

Host Instructions (x86)

- `movl -0x14(%r14), %ebp`
- `testl %ebp, %ebp`
- `jl 0x5555559b87af`
- `movq 0xe8(%r14), %rbp`
- `movq 0x40(%r14), %rbx`
- `addq %rbx, %rbp`
- `leaq 8(%rbp), %12`
- `movq (%rbp), %rbp`
- `movq (%r12), %12`
- `movq %rbp, 0x898(%r14)`
- `movq %r12, 0x8a0(%r14)`
- `movq 0xe0(%r14), %rbp`
- `addq %rbx, %rbp`
- `leaq 8(%rbp), %rbx`
- `movq (%rbp), %rbp`
- `movq (%rbx), %rbp`
- `movq %rbp, 0x8a8(%r14)`
- `...`
Vectors in Tiny Code Generator (TCG)

- **Vector Sizes**
  - Larger registers (128+ bits)
  - Potentially Smaller Lanes (16 bit, 8 bit)

- **TCG Op Sizes**
  - TCGv_i32
  - TCGv_i64
128 bit Vector Register Layout

<table>
<thead>
<tr>
<th>1x128</th>
<th>.Q[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x64</td>
<td>.D[1]</td>
</tr>
<tr>
<td></td>
<td>.D[0]</td>
</tr>
<tr>
<td>4x32</td>
<td>.S[3]</td>
</tr>
<tr>
<td></td>
<td>.S[2]</td>
</tr>
<tr>
<td></td>
<td>.S[1]</td>
</tr>
<tr>
<td></td>
<td>.S[0]</td>
</tr>
<tr>
<td>8x16</td>
<td>.H[7]</td>
</tr>
<tr>
<td></td>
<td>.H[6]</td>
</tr>
<tr>
<td></td>
<td>.H[5]</td>
</tr>
<tr>
<td></td>
<td>.H[4]</td>
</tr>
<tr>
<td></td>
<td>.H[3]</td>
</tr>
<tr>
<td></td>
<td>.H[2]</td>
</tr>
<tr>
<td></td>
<td>.H[1]</td>
</tr>
<tr>
<td></td>
<td>.H[0]</td>
</tr>
<tr>
<td></td>
<td>.B[14]</td>
</tr>
<tr>
<td></td>
<td>.B[12]</td>
</tr>
<tr>
<td></td>
<td>.B[10]</td>
</tr>
<tr>
<td></td>
<td>.B[9]</td>
</tr>
<tr>
<td></td>
<td>.B[8]</td>
</tr>
<tr>
<td></td>
<td>.B[3]</td>
</tr>
<tr>
<td></td>
<td>.B[1]</td>
</tr>
<tr>
<td></td>
<td>.B[0]</td>
</tr>
</tbody>
</table>
Marshalling Costs

- For generated code
  - More TCGops per guest instruction
  - Backend restricted to scalar instructions
  - Optimiser has to be simple and fast

- Helper Functions
  - As above but
  - Creating C ABI frame and procedure call
  - Helpers don’t vectorise

- See also:
  
  [Vectoring in on QEMUs TCG Engine, KVM Forum 2017](#)
RFC Patches

● Request For Comment
  ○ “I’ve had this idea.. what do you think?”
  ○ Doesn’t have to be complete

● TCG_vec RFC
  ○ AJB: August 17th 2017
  ○ 9 files changed, 222 insertions(+), 10 deletions(-)
  ○ RTH: August 17th 2017
  ○ 11 files changed, 1817 insertions(+), 94 deletions(-)
Ready for Upstream

● Version 11
  ○ On List: Jan 26th 2018
  ○ 25 files changed, 6973 insertions(+), 496 deletions(-)
  ○ Existing AArch64 guest translator converted to API
  ○ Existing x86 and AArch64 host generators extended to API

● Pull Request via TCG
  ○ V1 On List: Feb 7th 2018
  ○ Failed pre-merge tests
  ○ V2 On List: Feb 8th 2018

● Merged in master
  ○ since: Feb 8th 2018
  ○ in upcoming v2.12
Half Precision Floating Point

● ARMv8.2 FP16
  ○ Optional feature for v8.2
  ○ Mandatory for SVE

● 16 bit width
  ○ 5 bit exponent
  ○ 10 bit fraction
  ○ 1 sign bit

● Trade Bandwidth for Accuracy
  ○ Twice as many calculations than single-precision
  ○ Often enough accuracy
    ■ Some graphics calculations
    ■ Machine learning

● Requires SoftFloat
Softfloat

- **Why?**
  - Differing number formats (e.g. ARM alternative format)
  - NaN propagation
  - Default NaN
  - Exception Mappings

- **Berkeley SoftFloat**
  - John R. Hauser, BSD Licensed
  - Tarball release
  - Current release 3c

- **QEMU uses**
  - Version 2a
  - Numerous tweaks
Softfloat Options

- **Initial Discussions**
  - Our SoftFloat heavily patched
  - On List: [May 2017](#)
  - Patch Existing, Use 3c (replace/incremental), Use glibc?

- **RFC: Import SoftFloat3?**
  - On List: [July 2017](#)
  - A lot of duplication and patching
  - Would upstream take patches?

- **RFC: Update 2a ourselves**
  - On List: [Oct 2017](#)
  - Included WIP FP16 work
  - Also a lot of error prone duplication
Cambridge Sprint

- Nov 2017
  - Pair programming (Alex & Rich)
  - New re-factored SoftFloat
- V1 re-factor softfloat and add fp16 functions
  - On List: Dec 2017
  - 4 files changed, 3066 insertions(+), 3850 deletions(-)
  - No ARM FP16 in series
- V3 re-factor softfloat and add fp16 functions
  - On List: Jan 2018
  - 49 files changed, 2188 insertions(+), 2940 deletions(-)
  - but….
Clearer, slower code
Packed Parts

/*
 * Structure holding all of the decomposed parts of a float. The exponent is unbiased and the fraction is normalized. All calculations are done with a 64 bit fraction and then rounded as appropriate for the final format.
 *
 * Thanks to the packed FloatClass a decent compiler should be able to fit the whole structure into registers and avoid using the stack for parameter passing.
 */

typedef struct {
   uint64_t frac;
   int32_t exp;
   FloatClass cls;
   bool sign;
} FloatParts;
addsub_floats

/*@
 * Returns the result of adding or subtracting the floating-point
 * values `a' and `b'. The operation is performed according to the
 * IEC/IEEE Standard for Binary Floating-Point Arithmetic.
 */

float16 __attribute__((flatten)) float16_add(float16 a, float16 b, float_status *status)
{
    FloatParts pa = float16_unpack_canonical(a, status);
    FloatParts pb = float16_unpack_canonical(b, status);
    FloatParts pr = addsub_floats(pa, pb, false, status);

    return float16_round_pack_canonical(pr, status);
}
Performance

Nbench score; higher is better

FOURIER  NEURAL NET  LU DECOMPOSITION  gmean

system-2.5  master  softfloat-v4  softfloat-v5
Other Instruction Patches

● ARMv8.1-SIMD
  ○ Additional Advanced SIMD instructions
  ○ Rounding Double Multiply Add/Subtract
  ○ Forms the base decode path for ARMv8.3-CompNum

● ARMv8.3-CompNum
  ○ Additional Advanced SIMD instructions
  ○ Complex add, multiply accumulate
  ○ Mandatory for SVE

● ARM v8.1 simd + v8.3 complex insns
  ○ Gated by SoftFloat FP16 work
  ○ V3 on list: [Feb 2018](#)
  ○ 9 files changed, 1133 insertions(+), 126 deletions(-)
  ○ Merged: [Mar 2018](#)
Other plumbing

● Decoder Script
  ○ Help generate regular decoder
  ○ Based on instruction bit patterns

● target/arm: Preparatory work for SVE
  ○ Expand SIMD registers for SVE
  ○ Access helpers
  ○ Migration Support

● linux-user support
  ○ Add extended signal frame
  ○ Important for testing with RISU
SVE Patches

● target/arm: Scalable Vector Extension
  ○ V2 on list: Feb 2018
  ○ 13 files changed, 11408 insertions(+), 91 deletions(-)
  ○ 67 patches
  ○ 99% complete linux-user support
  ○ No FFR for now

● Call for review
  ○ Now is the time to review and test
  ○ On course for 2.13 (est Aug 2018)
  ○ https://github.com/rth7680/qemu/tree/tgt-arm-sve-8
Demo

- Himeno Benchmark
  - Advanced Centre for Computing and Communication @ Riken
  - Incompressible Fluid Analysis

- LULESH Benchmark
  - Lawrence Livermore National Lab
  - Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)
Potential Future Work

● First Fault/Non-fault loads
  ○ Simple handling of page boundaries

● SVE System Mode
  ○ Mostly system registers

● Performance Work
  ○ Better host register use?
  ○ Use host FPU instead of SoftFloat?
    ■ I posted an experimental hack with v4
    ■ Emilio Cota @UC posted RFC yesterday
Thank You

#HKG18

HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org