TOTAL SYSTEM AWARENESS IN TCG

ALEX BENNÉE

KVM FORUM 2019
INTRODUCTION
WHO/WHERE

stsquad  #qemu
@diasp.eu
@mastodon.org.uk

alex.bennee  @linaro.org

ARM, TCG, Testing, KVM
QEMU TCG plugins provide a way for users to run experiments taking advantage of the total system control emulation can have over a guest.
HISTORY OF PLUGINS
There will be no plugin system in the near future. Such systems are mainly useful for closed source projects, which QEMU is not. Moreover, as in ffmpeg, I don't want to bother about binary compatibility and API stability at this stage of the project.

Fabrice Bellard, qemu-devel, 2004
..what we do is enable loading a dynamic shared object as a 'plugin'... We do not use any additional header files or anything like that in our closed source bits, since that would of course violate the GPL.... so that we could 'replace' the initialization of certain built-in peripherals in QEMU with our own proprietary versions that live in the plugin.

Sponsorship for QEMU developers, qemu-devel, 2005
files will be dlopened by qemu at run time, and will register themselves as hardware to the appropriate hardware controller (ie a PCI device hardware plugin registers itself with the PCI bus).
project to create new ARM machine emulations using the
Python programming language.

{ANN} PyQemu 1.0 (and machine plugin patches), qemu-devel, 2007
PyQemu project to create new ARM machine emulations using the Python programming language.

“Sorry to ruin your GSoC project, but the plugin system was discussed last year, please see..”

{ANN} PyQemu 1.0 (and machine plugin patches), qemu-devel, 2007
LESSONS FROM HISTORY

- Wary of license evasion
- Worries about API Stability
- Solved!
LESSONS FROM HISTORY

- Wary of license evasion
- Worries about API Stability
- Solved!

(at least for upstream)
The problem is there currently are at least 3 users of Qemu:

1. *People who want fast simulation*

2. *People who are doing virtualization*

3. *People trying to do instrumentation/research*

...three groups have conflicting interests...adding instrumentation infrastructure will either slow down the common case, or ... introduce lots of #ifdefs
Instruction counting instrumentation for ARM + patch, qemu-devel, 2009
NEVER UP-STREAMED

- TEMU (~ QEMU 0.9.1)
- DECAF (~ QEMU 1.0)
- PEMU (~ QEMU 1.5)
- QTRACE (~ QEMU 1.7.1)
1ST QEMU USERS FORUM

- Cycle Accurate Simulation
- Program Instrumentation
- Cache/Pipeline Modelling

Trip Report, 1st QEMU Users Forum, qemu-devel, 2011
ATOS-TOOLS QEMU-PLUGINS

• Actively developed
• Can simulate HW in userspace
• Can wrap DineroIV
• Generates TCG ops in plugins

stable-3.1 @ github.com/atos-tool/qemu
CONCLUSIONS

- Plenty of demand
- Write-once forks
- Never to be up-streamed
THE PATH TO UPSTREAM

The Bolton Strid, Yorkshire, Alex Bennée, 2018
TCG TRACING -> INSTRUMENTATION

- trace_<eventname>_tcg
- Common Translator Loop
- Final instrumentation series never merged
USING LOGGING

qemu-aarch64 -d cpu,nochain -D sha1.trace \
  ./tests/tcg/aarch64-linux-user/sha1

PC=000000000000004002b4  X00=00000000000000000000000000000000  X01=00000000000000000000000000000000  X02=00000000000000000000000000000000  X03=00000000000000000000000000000000  X04=00000000000000000000000000000000  X05=00000000000000000000000000000000  X06=00000000000000000000000000000000  X07=00000000000000000000000000000000  X08=00000000000000000000000000000000  X09=00000000000000000000000000000000  X10=00000000000000000000000000000000  X11=00000000000000000000000000000000  X12=00000000000000000000000000000000  X13=00000000000000000000000000000000  X14=00000000000000000000000000000000  X15=00000000000000000000000000000000  X16=00000000000000000000000000000000  X17=00000000000000000000000000000000  X18=00000000000000000000000000000000  X19=00000000000000000000000000000000  X20=00000000000000000000000000000000  X21=00000000000000000000000000000000  X22=00000000000000000000000000000000  X23=00000000000000000000000000000000  X24=00000000000000000000000000000000  X25=00000000000000000000000000000000  X26=00000000000000000000000000000000  X27=00000000000000000000000000000000  X28=00000000000000000000000000000000
Trace points already exist as a series of interesting places in QEMU exposing information that can be used for analysis. By re-using them we avoid potential duplication of concerns. Adding new hook points becomes a simple case of adding a new trace point.

Trace updates and plugin RFC, qemu-devel, 2018
PROBLEMS

- Very wide API
- Helper per-operation
PAVEL'S SERIES

- Iteration of ISP RAS plugins tree
- *_needs filter
- directly calls helpers

Instrumentation, Introspection and Debugging with QEMU, KVM Forum 2017
RFC v2 QEMU binary Instrumentation Prototype, qemu-devel, 2018
EMILIO'S SERIES

- Direct helpers & inline ops
- Instruction granularity
- Required 2 pass translation
- Time control/lockstep vCPUS
- Guest hooks

RFC Plugin Support, qemu-devel, 2018
WHAT HAVE WE LEARNT

- APIs are hard
- Don't leak internals
TCG PLUGINS
DESIGN PRINCIPLES

- Low impact
- Simple non-leaky API
- No state modification
- Minimal viable plugin
VCPURun Loop (TCG)

- Run Loop: `qemu_tcg_cpu_thread_fn()`
- Find Next Block
- Generate Block
- Run Block
- Add new code to buffer
- Translated Code Buffer
TCG OPS

**ARM Instruction**

```
add x0, sp, #0x120
```

**TCG Ops**

```
movi_i64 tmp3,$0x120
add_i64 tmp2,sp,tmp3
mov_i64 x0,tmp2
```
INSERTING DUMMY OPS

**ARM Instruction**

- `add x0, sp, #0x120`

**TCG Ops**

1. `Dummy Helper Call (pre)`
2. `movi_i64 tmp3, $0x120`
3. `add_i64 tmp2, sp, tmp3`
4. `movi_i64 x0, tmp2`
5. `Dummy Helper Call (post)`

**End of Instruction**
FINAL SETUP

Translate → Optimize → Generate

GuestInsn
Instrumentation
Implementation

Host Code
TCG PLUGIN API
RULES

- Threading aware
- Opaque Handles
  - valid during callback only
- Do own housekeeping
int qemu_plugin_install(qemu_plugin_id_t id, const qemu_info_t *info, int argc, char **argv)
{
    if (argc) { /* process args */
        /* setup plugin bits... */

        /* register initial callbacks */
        qemu_plugin_register_vcpu_tb_trans_cb(id, vcpu_tb_trans);
        qemu_plugin_register_atexit_cb(id, plugin_exit, NULL);
        return 0;
    }
}
static void vcpu_tb_trans(qemu_plugin_id_t id, struct qemu_plugin_tb *tb) {
    /* query details */
    uint64_t pc = qemu_plugin_tb_vaddr(tb);
    unsigned long insns = qemu_plugin_tb_n_insns(tb);

    /* register execution callback */
    if (do_inline) {
        qemu_plugin_register_vcpu_tb_exec_inline(tb, QEMU_PLUGIN_INLINE_ADDR
            &count, insns);
    } else {
        qemu_plugin_register_vcpu_tb_exec_cb(tb, vcpu_tb_exec,
            QEMU_PLUGIN_CB_NO_REGS,
            (void *)insns);
    }
}
**BLOCK HELPERS**

qemu_plugin_tb_vaddr
qemu_plugin_tb_n_insns
static void vcpu_tb_trans(qemu_plugin_id_t id, struct qemu_plugin_tb *tb)
{
    size_t i, n = qemu_plugin_tb_n_insns(tb);
    for (i = 0; i < n; i++) {
        struct qemu_plugin_insn *insn = qemu_plugin_tb_get_insn(tb, i);

        if (do_inline) {
            qemu_plugin_register_vcpu_insn_exec_inline(
                insn, QEMU_PLUGIN_INLINE_ADD_U64, &insn_count, 1);
        } else {
            qemu_plugin_register_vcpu_insn_exec_cb(
                insn, vcpu_insn_exec_before, QEMU_PLUGIN_CB_NO_REGS, NULL);
        }
    }
}
INSTRUCTION HELPERS

qemu_pluginInsn_data  readable buffer
qemu_pluginInsn_size  size
qemu_pluginInsn_vaddr  virtual address
qemu_pluginInsn_haddr  hardware address
qemu_pluginInsn_disas  allocated string
INSTRUMENTING MEMORY ACCESSES

```c
static void vcpu_tb_trans(qemu_plugin_id_t id, struct qemu_plugin_tb *tb)
{
    size_t i, n = qemu_plugin_tb_n_insns(tb);
    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_haddr,
                                        QEMU_PLUGIN_CB_NO_REGS,
                                        rw, NULL);
    }
}
```
static void vcpu_haddr(unsigned int cpu_index, qemu_plugin_meminfo_t meminfo,
                      uint64_t vaddr, void *udata)
{
    ...
}

MEMORY CALLBACK
MEMORY INFO HELPERS

qemu_plugin_mem_size_shift
qemu_plugin_mem_is_sign_extended
qemu_plugin_mem_is_big_endian
qemu_plugin_mem_is_store
HW ADDRESS HELPERS

```c
struct qemu_plugin_hwaddr *hwaddr = qemu_plugin_get_hwaddr(meminfo, vaddr);

qemu_plugin_hwaddr_is_io        IO?
qemu_plugin_hwaddr_device_offset offset
    (including
     RAM)
```
OTHER APIS AND HELPERS

qemu_plugin_outs -d plugin output
plugin_reset/uninstall un-register callbacks
syscall/syscall_ret user syscall tracking
vcpu_[init/idle/exit] vcpu state
EXAMPLE PLUGINS

empty  measure overhead
bb     count translations
insn   count instructions
mem    count mem transactions
hotblocks profile execution
hotpages profile memory patterns
howvec profile instruction patterns
FUTURE WORK
MORE SYSTEM STATE

• System memory
• Registers
• Integrate with gdbstub?
• Device State?
DEVELOPER TOOLS
TIME

• Currently based on Host
  ▪ emulation overhead visible
  ▪ use icount
  ▪ but icount not MTTCG
• Expose to plugins
  ▪ read only, or
  ▪ allow plugins to drive timers?
SUMMARY

- A long journey
- Common non-invasive interface
- Efficient in the null case
- Can be extended
  - in and out-of-tree
QUESTIONS?
EXTRA SLIDES
SUMMARY

- Translation Phase
  - Guest Insn -> TCGops
  - Optimise TCG Ops
  - TCGOps -> Host Instructions
- Execution Phase