Towards multi-threaded TCG

Alex Bennée

alex.bennee@linaro.org

KVM Forum 2015
Introduction
Hello!

- Alex Bennée
- Works for Linaro
- IRC: stsqaud/ajb-linaro
- Mostly ARM emulation, a little KVM on the side
- Uses Emacs
What is multi-threaded TCG?
TCG?

- Tiny Code Generator
- Running non-native code on your desktop
Current process model
How it looks

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>22953</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>742M</td>
<td>29112</td>
<td>10288</td>
<td>S</td>
<td>100.0</td>
<td>0.1</td>
<td>0:09:29</td>
<td>/home/alex/src/qemu/qemu.git/arm-softmmu/qemu-system-arm -machine virt,accel=tcg -cpu cortex-a15 -device virtio-serial-device -device virtio console,cmdline=init=/bin/sh console=ttym0 console=tty0 console=ttyS0 initrd=/initrd-lzimage-2.6.32-771.1.0.20120625.1 arch=arm32v7 output=-hda=/dev/vg_qemu/rootfs -smp 4 -append &quot;excl&quot;</td>
</tr>
<tr>
<td>18049</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>2715M</td>
<td>1723M</td>
<td>86292</td>
<td>S</td>
<td>14.5</td>
<td>5.4</td>
<td>5h57:08</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enajal</td>
</tr>
<tr>
<td>18607</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>1066M</td>
<td>97M</td>
<td>7588</td>
<td>S</td>
<td>6.6</td>
<td>0.3</td>
<td>3h10:31</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=ppapi --channel=channel</td>
</tr>
<tr>
<td>29724</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>27754</td>
<td>3928</td>
<td>1368</td>
<td>S</td>
<td>2.0</td>
<td>0.0</td>
<td>2h20:33</td>
<td>htot</td>
</tr>
<tr>
<td>430</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>2764</td>
<td>3872</td>
<td>1372</td>
<td>S</td>
<td>2.0</td>
<td>0.0</td>
<td>1h39:21</td>
<td>htot</td>
</tr>
<tr>
<td>15859</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>27052</td>
<td>3366</td>
<td>1452</td>
<td>R</td>
<td>1.3</td>
<td>0.0</td>
<td>0:07:51</td>
<td>htot</td>
</tr>
<tr>
<td>21405</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>1354M</td>
<td>358M</td>
<td>70676</td>
<td>S</td>
<td>0.7</td>
<td>1.1</td>
<td>20:17:47</td>
<td>/opt/google/chrome/chrome --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>23665</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>161M</td>
<td>23164</td>
<td>23164</td>
<td>S</td>
<td>0.7</td>
<td>0.5</td>
<td>8:28:34</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>23681</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>999M</td>
<td>151M</td>
<td>31936</td>
<td>S</td>
<td>0.7</td>
<td>0.5</td>
<td>8:20:12</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>8975</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>2386M</td>
<td>1160M</td>
<td>76724</td>
<td>S</td>
<td>0.7</td>
<td>3.6</td>
<td>1h02:53</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>19612</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>50824</td>
<td>4316</td>
<td>1444</td>
<td>S</td>
<td>0.7</td>
<td>0.0</td>
<td>0:44:13</td>
<td>nosh-server new -c 256 -t LANG=en_GB.UTF-8 -l LANGUAGE=en_GB,</td>
</tr>
<tr>
<td>22381</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>877M</td>
<td>138M</td>
<td>39428</td>
<td>S</td>
<td>0.6</td>
<td>0.4</td>
<td>7:28:09</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>23188</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>805M</td>
<td>291M</td>
<td>235M</td>
<td>S</td>
<td>0.6</td>
<td>0.9</td>
<td>17:09:42</td>
<td>/usr/lib/chromium-browser/chromium-browser --type=renderer --enable-deferred-image</td>
</tr>
<tr>
<td>5505</td>
<td>alex</td>
<td>20</td>
<td>0</td>
<td>808M</td>
<td>127M</td>
<td>10632</td>
<td>S</td>
<td>0.0</td>
<td>0.4</td>
<td>21:13:18</td>
<td>emacs --daemon</td>
</tr>
</tbody>
</table>

Load average: 0.85 0.85 0.71
Multi-threaded TCG

Threads

vCPU1 → vCPU2 → vCPU3 → monitor

CPU1 → CPU2 → CPU3 → … → CPUn
Multithreaded programming

Reality?

Theory

Actual
Why do we want it?
Living in a Multi-core world
Raspberry Pi 2

Quad-core Cortex A7 @900Mhz

$25
Dragonboard 410c

Quad-core Cortex A53 @ 1.4Ghz

$75
Nexus 5

Quad Core Krait 400 @ 2.26Ghz

$339
My Desktop

Intel i7 (4 core + 4 hyperthreads) @ 3.4 Ghz

$600
Build Server

2 x Intel Xeon (6+6 hyperthreads) @ 3.46 Ghz

$2-3k
Android Emulation

- Android emulator uses QEMU as base
- Most modern Android devices are multi-core
Per-core performance

via @HenkPoly
Other reasons to care
Using QEMU for System bring up

• Increasingly used for prototyping
  ▪ new multi-core systems
  ▪ new heterogeneous systems
• Want concurrent behaviour
  ▪ Bad software should fail in QEMU!
As a development tool

• Instrumentation and inspection
• Record and playback
• Reverse debugging
Cross Tooling
Building often complex

http://lukeluo.blogspot.co.uk/2014/01/linux-from-scratch-for-cubietruck-c4.html
Just use qemu-linux-user?

- Make sure binfmt_misc setup
- Mess around with multilib/chroots
- Hope threads/signals not used
Or boot a multi-core system
Things in our way

- Global State in QEMU
- Guest Memory Models
Global State

- Numerous globals in TCG generation
- TCG Runtime Structures
- Device emulation structures
Guest Memory models

- Atomic behaviour
- LL/SC Semantics
- Memory barriers
How can we do it?
3 broad approaches
Use threads/locks
Use processes/IPC

Re-write from scratch
## Pros/Cons of each approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Threads/Locks</th>
<th>Process/IPC</th>
<th>Re-write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Performance</td>
<td>Correctness</td>
<td>Shiny and New!</td>
</tr>
<tr>
<td>Cons</td>
<td>Performance, Complexity</td>
<td>Performance, Invasive</td>
<td>Wasted Legacy, New problems</td>
</tr>
</tbody>
</table>
What we have done

- Protected code generation
- Serialised the run loop
  - translated code multi-threaded
- New memory semantics
- Multi-threaded device emulation
Things in our way

- Global State in QEMU
- Guest Memory Models
Code generator globals

Threads

vCPU 1

write

vCPU 2

read

TCG Variables

write

cpu_V0

read

read
TCG Runtime structures

- SoftMMU TLB
- Translation Buffer Jump Cache
- Condition Variables (tcg_halt_cond)
- Flags (exit_request)
per-CPU variables

- `tcg_halt_cond -> cpu->halt_cond`
- `exit_request -> cpu->exit_request`
Quick reminder of how TCG works
Code Generation

- target machine code
- intermediate form (TCG ops)
- generate host binary code
## Input Code

<table>
<thead>
<tr>
<th>ldr</th>
<th>r2, [r3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>r2, r2, #1</td>
</tr>
<tr>
<td>str</td>
<td>r2, [r3]</td>
</tr>
<tr>
<td>bx</td>
<td>lr</td>
</tr>
</tbody>
</table>
TCG Ops

mov_i32 tmp5,r3
gemu_ld_i32 tmp6,tmp5,leul,3
mov_i32 r2,tmp6

mov_i32 tmp5,$0x1
mov_i32 tmp6,r2
add_i32 tmp6,tmp6,tmp5
mov_i32 r2,tmp6

mov_i32 tmp5,r3
mov_i32 tmp6,r2
gemu_st_i32 tmp6,tmp5,leul,3

exit_tb $0x7ff368a0baab
Output Code

mov (%rsi),%ebp
inc %ebp
mov %ebp,(%rsi)
Basic Block

- `prologue`
- `code`
- `exit 1` and `exit 2`
Block Chaining
TCG Global State

- Code generation globals
- Global runtime
Translated code is safe

- Only accesses vCPU structures
- We need to careful leaving the translated code
Exit Destinations

• Back to Run Loop
• Helper Function
Exit to run loop

Enter JIT Code

Return to runloop
Simplified Run Loop

Run Loop

Find Next Block

Generate Block

Run Block

Translated Code Buffer

Add new code to buffer
Types of Helper

- **Complex Operations**
  - should only touch private vCPU state
  - no locking required*

- **System Operations**
  - locking for cross-cpu things
  - some operations affect all vCPUs
Stop the World!

- Using locks
  - expensive for frequently read vCPU structures
  - complex when modifying multiple vCPUs data
- Ensure relevant vCPUs halted, modify at "leisure"
Deferred Work

- Existing queued_work mechanism
  - add work to queue
  - signal vCPU to exit
- New queued_safe_work
  - waits for all vCPUs to halt
  - no lock held when run
TCG Summary

- Move global vars to per-CPU/Thread
  - exit and condition variables
- Make use of tb_lock
  - uses existing TCG context tb_lock
  - protects all code generation/patching
  - protects all manipulation of tb_jump_cache
- Add async safe work mechanism
  - Defer tasks until all vCPUs halted
Things in our way

- Global State in QEMU
- Guest Memory Models
No Atomic TCG Ops
Atomic Behaviour is easy when Single Threaded
Considerably harder when Multi-threaded
Load-link/Store-conditional (LL/SC)

- RISC alternative to atomic CAS
- Multi-instruction sequence
- Store only succeeds if memory not touch since link
- LL/SC can emulate other atomic operations
LL/SC in QEMU

- Introduce new TCG ops
  - qemu_ldlink_i32/64
  - qemu_stcond_i32/64
- Can be used to emulate
  - load/store exclusive
  - atomic instructions
SoftMMU
What it does

- Maps guest loads/stores to host memory
  - uses an addend offset
- Fast path in generated code
- Slow path in C code
  - Victim cache lookup
  - Target page table walk
How it works: Stage one
How it works: Stage two
How it works: Stage three
How does this help with LL/SC?

- Introduced new TCG ops
  - qemu_ldlink_i32/64
  - qemu_stcond_i32/64

Using the SoftMMU slow path we can implement the backend in a generic way
LL/SC in Pictures
LL/SC Summary

- New TLB_EXCL flag marks page
- All access now follows slow-path
  - trip exclusive flag
- Store conditional always slow-path
  - Will fail if flag tripped
Memory Model Summary

- Multi-threading brings a number of challenges
- New TCG ops to support atomic-like operations
- SoftMMU allows fairly efficient implementation
- Memory barriers still an issue.
Device Emulation
KVM already done it ;-)  

- added thread safety to a number of systems  
- introduced memory API  
- introduced I/O thread
TCG access to device memory

- All MMIO pages are flagged in the SoftMMU TLB
- The slowpath helper passes the access to the memory API
- The memory API defines regions of memory as:
  - lockless (the eventual driver worries about concurrency)
  - locked with the BQL
Thanks KVM!
Current state
Performance & Demo

- Hand over to Frederic
What’s left

- LL/SC Patches
- MTTCG Patches
- Memory Barriers
- Enabling all front/back ends
- Testing & Documentation
LL/SC Patches

- Majority of patch set independent from MTTCG
- Been through a number of review cycles
- Hope to get merged soonish now tree is open

Who/where?

- Alvise Rigo of Virtual Open Systems
- https://git.virtualopensystems.com/dev/qemu-mt.git
- Latest branch: slowpath-for-atomic-v4-no-mttcg
MTTCG Patches

- Clean-up and rationalisation patches
  - starting to go into maintainer trees
- Delta to full MTTCG reducing

Who/where?

- Frederic Konrad of Greensocs
- [http://git.greensocs.com/fkonrad/mttcg.git](http://git.greensocs.com/fkonrad/mttcg.git)
- Latest branch: multi_tcg_v7
Memory Barriers

- No code yet
- Current proposal is one (or two) barrier TCG ops
- Hard to trigger barrier issues on x86 backend
Enabling all front/back ends

- Current testing is ARM32 on x86
- Aim to enable MTTCG on all front/backends
- Front-ends need to use new TCG ops
- Back-ends need to support new TCG ops
  - may require incremental updates
Testing & Documentation

• Both important for confidence in design
• Torture tests
  ▪ hand-rolled
  ▪ using kvm-unit-tests
• Want to have reference in docs/ on how it should work
Questions?
The End

Thank you
Extra Material
Full TLB Walk Diagram
Annotated TLB Walk Code (In)

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Operation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000000</td>
<td>e3a00000</td>
<td>mov r0, #0</td>
<td>; 0x0</td>
</tr>
<tr>
<td>0x40000004</td>
<td>e59f1004</td>
<td>ldr r1, [pc, #4]</td>
<td>; 0x40000010</td>
</tr>
</tbody>
</table>
Annotated TLB Walk Code (Ops)

---- prologue
ld_i32 tmp5,env,$0xfffffffffffffffff4
movi_i32 tmp6,$0x0
brcond_i32 tmp5,tmp6,ne,$L0

---- 0x40000000
movi_i32 tmp5,$0x0
mov_i32 r0,tmp5

---- 0x40000004
movi_i32 tmp5,$0x40000000c
movi_i32 tmp6,$0x4
add_i32 tmp5,tmp5,tmp6
qemu_ld_i32 tmp6,tmp5,leu1,1
mov_i32 r1,tmp6
Annotated TLB Walk Code (Opt Op)

OP after optimization and liveness analysis:

```plaintext
---- prologue
ld_i32 tmp5, env, $0xffffffffffffff4
movi_i32 tmp6, $0x0
brcond_i32 tmp5, tmp6, ne, $L0

---- 0x40000000
movi_i32 r0, $0x0

---- 0x40000004
movi_i32 tmp5, $0x40000010
qemu_ld_i32 tmp6, tmp5, leu1, 1 (val, addr, index, opc)
mov_i32 r1, tmp6
```
Annotated TLB Walk Code (Out Asm)

---- prologue
0x7fffe1ba1000: mov  -0xc(%r14),%ebp
0x7fffe1ba1004: test %ebp,%ebp
0x7fffe1ba1006: jne 0x7fffe1ba10c9

--- 0x40000000
0x7fffe1ba100c: xor %ebp,%ebp
0x7fffe1ba100e: mov %ebp,(%r14)

--- 0x40000004
- movi_i32
0x7fffe1ba1011: mov $0x40000010,%ebp
- qemu ld_i32
0x7fffe1ba1016: mov %rbp,%rdi - r0
0x7fffe1ba1019: mov %ebp,%esi - r1

0x7fffe1ba101f: and $0xffffffffc03,%esi

- index into tlb_table[mem_index][0]+target_page
0x7fffe1ba101b: shr $0x5,%rdi
0x7fffe1ba1025: and $0x1fe0,%edi

0x7fffe1ba102b: lea 0x2c18(%r14,%rdi,1),%rdi
0x7fffe1ba1033: cmp (%rdi),%esi
0x7fffe1ba1035: mov %ebp,%esi
0x7fffe1ba1037: jne 0x7fffe1ba111b

--- offset to "host address"
0x7fffe1ba103d: add 0x10(%rdi),%rsi
--- actual load
0x7ffe1ba1041: mov (%rsi),%ebp
--- mov_i32 r1, tmp6
0x7ffe1ba1043: mov %ebp,0x4(%r14)

----- slow path function call
0x7ffe1ba111b: mov %r14,%rdi
0x7ffe1ba111e: mov $0x21,%edx
0x7ffe1ba1123: lea -0xe7(%rip),%rcx # 0x7ffe1ba1043
0x7ffe1ba112a: mov $0x555555553980,%r10 # helper_le_ldul_mmu
0x7ffe1ba1134: callq *%r10
0x7ffe1ba1137: mov %eax,%ebp
0x7ffe1ba1139: jmpq 0x7ffe1ba1043
Locking in run loop

Main Run Loop

- tcg_exec_all
- cpu_exec
- setjmp

for(;;)
Handle IRQ/EXP

- tb_find_fast
  - tb_find_slow
- cpu_tb_exec

Deal with Exit

qemu_tcg_wait_io_event
Wait on CPU(halt_cond)?

BQL ➔ RCU Memory Layout
tb_lock ➔ Translation Lock

Generate new code

JIT CODE

BQL  rcu_read  tb_lock
<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait on CFS(mate_cond)</td>
</tr>
<tr>
<td>flush_queued_safe_work</td>
</tr>
<tr>
<td>flush_queued_work</td>
</tr>
</tbody>
</table>
SoftMMU Slowpath Reasons

- Missing mapping
  - first access (fill)
  - crossed target page (refill)
- Mapping invalidated
- Page not dirty
- Page is MMIO