Towards multi-threaded TCG

Alex Bennée
alex.bennee@linaro.org

Linaro Connect SFO15
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

Processes

- vCPU_n
- ...
- vCPU3
- vCPU2
- vCPU1

- monitor
- VNC
- I/O
- poll()

CPUs

- CPU1
- CPU2
- CPU3
- ...
- CPU8
How it looks

```
1 [    ] 3.3%
2 [    ] 100.0%
3 [    ] 3.4%
4 [    ] 4.6%
5 [    ] 0.7%
6 [    ] 10.7%
7 [    ] 1.3%
8 [    ] 0.7%
Mem: 31861MB used: 20360MB buffers: 5480M cache: 5058M
Load average: 0.85 0.85 0.71

PID USER   PRI NI VIRT  RES  SHR S  %CPU %MEM   TIME+  COMMAND
22953 alex  20  0 742M 29112 10988 S 100. 0.1  0:09.20 /home/alex/lsrc/qemu/qemu.git/arm-softmmu/qemu-system-arm -machi
8949 alex   20  0 2715M 1723M 86292 S 14.5  5.4  5h57:08 /usr/lib/chromium-browser/chromium-browser --type=renderer --ena
18607 alex  20  0 1661M 97M 7588 S  6.6  0.3  3h10:31 /usr/lib/chromium-browser/chromium-browser --type=ppapi --channe
29724 root  20  0 27764 3923 1368 S  2.0  0.0  2h20:33 htop
430 alex   20  0 27644 3872 1372 S  2.0  0.0  1h38:21 htop
15859 alex  20  0 27052 3386 1452 R  1.3  0.0  0:07:51 htop
21465 alex  20  0 1354M 358M 7068 S  0.1  0.0  28:17:42 /opt/google/chrome/chrome --type=renderer --enable-deferred-imag
23665 alex  20  0 1610M 161M 23104 S  0.1  0.7  8:28:34 /opt/google/chrome/chrome --type=renderer --enable-deferred-imag
23681 alex  20  0 999M 151M 31936 S  0.1  0.5  8:20:12 /opt/google/chrome/chrome --type=renderer --enable-deferred-imag
8975 alex  20  0 2386M 1160M 76724 S  0.7  3.6  1h03:53 /usr/lib/chromium-browser/chromium-browser --type=renderer --ena
19612 alex  20  0 50824 4316 1444 S  0.1  0.0  0:44:13 mosh-server new -s -c 256 -l LANG=en_GB.UTF-8 -l LANGUAGE=en_GB.
22381 alex  20  0 877M 138M 39428 S  0.0  0.0  7:28:09 /opt/google/chrome/chrome --type=renderer --enable-deferred-imag
23188 alex  20  0 805M 291M 235M S  0.0  0.9  17:09:42 /opt/google/chrome/chrome --type=renderer --channel=32125.0.3
5595 alex  20  0 808M 127M 10832 S  0.0  0.4  21:13:18 emacs --daemon
```

13:03 alex@zen/x86_64 [kvm-unit-tests.git/mtticg/current-tests-v2] ➜ /arm/run ./arm/barrier-test.flat -smp 4 --append "excl"
Running with TCG
/home/alex/lsrc/qemu/qemu.git/arm-softmmu/qemu-system-arm -machine virt,accel=tcg -cpu cortex-a15 -device virtio-serial-device -device virtconsole,charger=ctd -chargerdev testdev,ivd=ctd -display none -serial stdio -kernel ./arm/barrier-test.flat -smp 4 --append excl
CPU1 online
CPU2 online
CPU3 online
CPU4 online
CPU5 online
CPU6: Done, 100000000 incs
CPU7 online
CPU8: Done, 100000000 incs
```
Multi-threaded TCG

Threads

CPU1

CPU2

CPU3

CPUn

vCPU1

vCPU2

vCPU3

monitor

VNC

I/O

poll()
Reality?

Multithreaded programming

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

```
version:

CPU: 1

Hardware: Generic DT based system
Revision:
Serial:
root@debian:~

Linux Debian 4.1.0-rc6-ajb #7 SMP Fri Jun 12 17:58:11 BST 2015 armv7l GNU/Linux

processor: 0
model name: ARMv7 Processor rev 1 (v7l)
BogoMIPS: 125.00
Features: half thumb fastmult fp vfp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpea eavt
CPU implementer: 0x41
CPU architecture: 7
CPU variant:
CPU version: 1

processor: 1
model name: ARMv7 Processor rev 1 (v7l)
BogoMIPS: 125.00
Features: half thumb fastmult fp vfp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpea eavt
CPU implementer: 0x41
CPU architecture: 7
CPU variant:
CPU version: 1

processor: 2
model name: ARMv7 Processor rev 1 (v7l)
BogoMIPS: 125.00
Features: half thumb fastmult fp vfp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpea eavt
CPU implementer: 0x41
CPU architecture: 7
CPU variant:
CPU version: 1

processor: 3
model name: ARMv7 Processor rev 1 (v7l)
BogoMIPS: 125.00
Features: half thumb fastmult fp vfp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpea eavt
CPU implementer: 0x41
CPU architecture: 7
CPU variant:
CPU version: 1

Hardware: Generic DT based system
Revision:
Serial:
root@debian:~
```

Thu Aug 13 09:32 (Lavg 4.14, 2.72, 1.74)
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
read

vCPU 2

read

TCG Variables

cpu_V0

write
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
qemu_ld_i32 tmp6, tmp5, leu1, 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
qemu_st_i32 tmp6, tmp5, leu1, 3

exit_tb $0x7ff368a0baab
```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>(%rsi),%ebp</td>
</tr>
<tr>
<td>inc</td>
<td>%ebp</td>
</tr>
<tr>
<td>mov</td>
<td>%ebp,(%rsi)</td>
</tr>
</tbody>
</table>
Basic Block
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

<table>
<thead>
<tr>
<th>block</th>
<th>prologue</th>
<th>code</th>
<th>exit 1</th>
<th>exit 2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>block</th>
<th>prologue</th>
<th>code</th>
<th>exit 1</th>
<th>exit 2</th>
</tr>
</thead>
</table>
Simplified Run Loop

Run Loop

Find Next Block

Generate Block

Run Block

Translated Code Buffer

Add new code to buffer
Helper Functions

- `cpu_tb_exec`
  - `block`
    - `prologue`
    - `code`
      - `exit 1`
      - `exit 2`

- QEMU C Code
  - Complex Op
  - System Op

- vCPU State
  - Registers
  - Global State
  - Jump Cache

- Return to runloop
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
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
Emilo’s Patches

- Recent patch series posted to list
- Alternate solutions
  - AIE helpers for LL/SC
  - Example implementation of barrier semantics
Memory Barriers

- Some example code (Emilo's patches)
- Use a number of 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

[Diagram of TLB walk process with steps and conditions like MMU Mode, Guest Address, MMU Index, Access Type, TLB Page, CPU TLB, and TLB Entry with fields such as addr_read, addr_write, addr_code, and addend.]
Annotated TLB Walk Code (In)

0x40000000: e3a00000      mov   r0, #0   ; 0x0
0x40000004: e59f1004      ldr   r1, [pc, #4]   ; 0x40000010
Annotated TLB Walk Code (Ops)

---- prologue
ld_i32 tmp5,env,$0xfffffffffffffff4
movi_i32 tmp6,$0x0
brcond_i32 tmp5,tmp6,ne,$L0

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

---- 0x40000004
movi_i32 tmp5,$0x4000000c
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:

```assembly
---- 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,leul,1 (val, addr, index, opc)
mov_i32 r1,tmp6
```
Annotated TLB Walk Code (Out Asm)

--- prologue

0x7ffe1ba1000:  mov    -0xc(%r14),%ebp
0x7ffe1ba1004:  test   %ebp,%ebp
0x7ffe1ba1006:  jne    0x7ffe1ba10c9

---- 0x40000000

0x7ffe1ba100c:  xor     %ebp,%ebp
0x7ffe1ba100e:  mov     %ebp,(%r14)

---- 0x40000004

- movi_i32
0x7ffe1ba1011:  mov     $0x40000010,%ebp
- qemu_ld_i32
0x7ffe1ba1016:  mov     %rbp,%rdi - r0
0x7ffe1ba1019:  mov     %ebp,%esi - r1

0x7ffe1ba101f:  and    $0xfffffc03,%esi

- index into tlb_table[mem_index][0]+target_page
0x7ffe1ba101b:  shr     $0x5,%rdi
0x7ffe1ba1025:  and     $0x1fe0,%edi

0x7ffe1ba102b:  lea     0x2c18(%r14,%rdi,1),%rdi
0x7ffe1ba1033:  cmp     (%rdi),%esi
0x7ffe1ba1035:  mov     %ebp,%esi
0x7ffe1ba1037:  jne     0x7ffe1ba111b

--- offset to "host address"
0x7ffe1ba103d:  add     0x10(%rdi),%rsi
--- actual load
0x7ffe1ba1041:  mov     (%rsi),%ebp
--- mov_i32 r1, tmp6

----- slow path function call

0x7ffe1ba1041:  mov  (%rsi),%ebp
0x7ffe1ba1043:  mov  %ebp,0x4(%r14)

0x7ffe1ba111b:  mov  %r14,%rdi
0x7ffe1ba111e:  mov  $0x21,%edx
0x7ffe1ba1123:  lea  -0xe7(%rip),%rcx  # 0x7ffe1ba1043
0x7ffe1ba112a:  mov  $0x5555555653980,%r10  # helper_le_1dul_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

for(;;)
Handle IRQ/EXP

tb_find_slow

cpu_tb_exec

Deal with Exit

jmp return

qemu_tcb_wait_io_event
Wait on CPU(halt_cond)?

BQL → RCU Memory Layout

tb_lock → Translation Lock

BQL  rcu_read  tb_lock

Generate new code

JIT CODE
SoftMMU Slowpath Reasons

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