Challenges Revisited

Supporting Virtual CPU Hotplug On Architectures That Support CPU Hotplug (like ARM64)

Challenges & Workarounds In Qemu

A Follow-up Presentation of KVMForum 2020: https://sched.co/eE4m

Salil Mehta (Huawei, Cambridge, UK)
James Morse (ARM, Cambridge, UK)

Date: 15th June 2023, 15:45–16:15 (Europe/Prague/Brno), Room 1
Overview

1) Motivation

2) Brief Anatomy: How Do We Setup ARM VCPU During VM Init?

3) Brief Anatomy: How Do We Setup ARM GIC During VM Init

4) VCPU Hot-plug Impediments: KVM Init (Recap)

5) VCPU Hot-plug Impediments: Guest Kernel Init (Recap)

6) Workaround: to KVM Init Impediments

7) Workaround: to Guest Kernel Init Impediments

8) Qemu Challenges: to Realize VCPU Hotplug

9) Qemu Workarounds: Creative Hack – Which Works! Really?

10) Prototype Code: Repositories & Upstreaming

11) Q&A
Motivation: of Supporting VCPU Hot-plug on ARM like Platforms

- Required by Cloud Vendors as part of the orchestration framework which could adjust resource requests (CPU and Mem requests) for the containers in a pod, dynamically based on usage or SLA (Pay-as-you-grow).

- Should have look-and-feel of x86_64 platform VCPU hot-plug feature

- For example, to support Kata-containers (Light Weight VMs) with Qemu/ARM64
  - Kata Containers VM starts with a minimum amount of resources, (hot-plug everything)
    - allowing for faster boot time and
    - reduction in memory footprint.
  - As the container launch progresses, devices are hot-plugged to the VM.
Terminology: From the ACPI Perspective

- **Possible** VCPUs = Maximum possible VCPUs in a VM
- **Present** VCPUs = which are present but further need to be ‘enabled’. Cannot be used,
- **Enabled** VCPUs = which are ‘present’ and can be ‘onlined’ (PSCI) for use
**Brief Anatomy:** How Do We **Setup ARM VCPU** During VM Init? (1)

- **QOM ARM CPU** Object & **KVM VCPU** object
- Allocation & Initialization
  - Various per-cpu features (PMU, SVE etc.)
- Realization
  - Creation KVM VCPU (**KVM_CREATE_VCPU**)
    - Alloc of 'struct kvm_vcpu'
    - Initialization of various per-cpu data-structures
    - Setup Timer, PMU, debug etc
    - Config of SGIs, PPIs
    - Associated **VGIC CPU interface** initialization
  - Initialization of KVM VCPU (**KVM_ARM_VCPU_INIT**)
  - Qemu CPU Thread Exec (**KVM_RUN**)

---

**KVM Forum 2023**

**HUAWEI**
Brief Anatomy: How Do We **Setup ARM GIC** During VM Init (2)

- **GIC Initialization (Qemu & KVM)**
  - Initialize GICV3 data structures (distributor, redistributor, cpu) in QEMU
  - KVM VGIC CPU interface initialization (**KVM_CREATE_VCPU**)
  - Initialize the GIC Distributor in KVM
    - Various SPIs, vITS alloc etc. (**KVM_DEV_ARM_VGIC_CTRL_INIT**)
    - IO Devs and memory regions – only once! (**KVM_RUN**)
      - Requires all KVM vCPU redistributors have been setup. (?)
      - vgic_v3_map_resources() and declare VGIC ready.
    - Base address (**KVM_VGIC_V3_ADDR_TYPE_DIST**)
  - Initialize the GIC redistributor in KVM
    - IO Devs, memory regions, Base address (**KVM_VGIC_V3_ADDR_TYPE_REDIST_REGION**)
  - Initialize GIC CPU Interface Register Info (like for ICC_CTLR_EL1) in Qemu Model
    - Required to get actual supported reg config from KVM
VCPU Hotplug Impediments: \textbf{KVM Init (Recap)}

1) All ARM VCPUs MUST be created in the Host KVM at VM Init
   - \textbf{Number of VCPUs} in the System cannot change after Init \textbf{(ARM CPU Arch Req)}
     - VCPU related features Init: Timers, PMU, Debug, private IRQs (= SGIs, PPIs...) etc. \textbf{(KVM\_CREATE\_VCPU)}
   - \textbf{Sizing of VGIC} in Host KVM can only happen once (Number of VCPUs) \textbf{(ARM CPU Arch Req?)}
     - Like VGIC Redistributor KVM iodevs, Memory Regions etc. \textbf{(KVM\_VGIC\_V3\_ADDR\_TYPE\_REDIST\_REGION)}
     - \textbf{Association}: VGIC Redistributor $\leftrightarrow$ VGIC CPU

2) VCPUs once created in KVM cannot be destroyed. \textbf{(Limitation of KVM)}
VCPU Hotplug Impediments: Guest Init (Recap)

1) Guest Kernel MUST see all ARM VCPUs specified by ACPI MADT GICC at Boot time (ARM CPU Arch Req)
   - Number of VCPUs in the Guest Kernel cannot change after Boot
   - Number of Redistributors exposed to Guest Kernel cannot change after boot
   - Association: VGIC Redistributor <=> VGIC CPU cannot change

2) VCPUs specified at Guest Kernel Boot by ACPI cannot be ‘un-plugged’ (ARM CPU Arch Req)
Workaround: to KVM Init Impediments (Recap)

1) **No workarounds!** Host KVM Initialization Should remain unchanged *(Requirement from ARM Maintainers)*

2) Fake VCPU Hot-{un}plug at VMM/Qemu level

3) Hence, **No other option than to Create all KVM VCPUs at VM Init**
   - Only Change **power state** of VCPUs When 'faking' hot-unplug in Qemu

4) **Size GICV3 once**
   - GIC Redistributor KVM iodevs, Memory Regions etc. – Yah! All of that!
   - Initialize **GICR and GIC CPU Interface association** in KVM once at VM Init

5) No need to destroy 'struct kvm_vcpu'. **Keep them parked** in powered down state
Workaround: to Guest Kernel Init Impediments (Recap)

What we Can't Do?

1) Number of Booted VCPUs & GIC redistributors Should remain unchanged (ARM Arch Req)

What we Can do?

2. Support to defer 'Online'ing the VCPUs to a later time when Guest has booted (Details in James Morse ppt)
   - Introduce new 'online-capable' Bit in ACPI MADT Table GIC CPU Interface 'flag' field [1]
     - GICC = 'Enabled' (must be 'online'd' during boot) or
     - GICC = 'online-capable' (can be deferred 'online'ing' after Guest Kernel boots)

3. Enabled or Disabled CPU in Guest Kernel? evaluate(_STA) for CPU
   - Enabled CPU (In Guest) = Plugged VCPU (In Qemu) = _STA.PRESENT=1 _STA.ENABLED=1 (ACPI Status)
   - Disabled CPU (In Guest) = Un-Plugged VCPU (In Qemu) = _STA.PRESENT=1 _STA.ENABLED=0 (ACPI Status)

[1] ACPI specification 6.5 "Table 5.37: GICC CPU Interface Flags"
Long Story Short: Feels Like Hotplug. But It Isn’t!

1. **Guest Kernel** (MADT/GICC, online-capable = 1)
   - **Plugged** (cold, hot)? Present = 1, Enabled = 1 (ACPI Status)
   - **Unplugged** (cold, hot)? Present = 1, Enabled = 0
   - ACPI Enumerate only **Plugged** i.e. `register_cpu()` \(\rightarrow\) sysfs
   - Attempt **Online** (via PSCI) (Subject to VMM Policy)

2. **Qemu** (Workarounds)
   - **Realize** only **Plugged**: (cold, hot) vCPUs (Qemu Threads)
   - **Park** KVM vCPUs: Unplugged QOM CPUs (KVM Power Off)
   - **Lie** to the **Guest**: about the ‘Presence’ of vCPUs
   - **Fake Hotplug**: at QOM
   - **Deny Online’ing**: Unplugged vCPUs (SMCC/HYP Handle)

3. **Host KVM** (No Workarounds: Arch Constraints)
   - **Possible** vCPUs: Fully Initialize
   - **Unplugged** vCPUs: Powered Down
   - **VGIC**: Fully Initialize
   - **Exit HVC/SMC**: CPU\(_{\text{ON,OFF}}\) to VMM (Policy Check)

**Covered In Previous Next Slides**

**Accompanying James Morse’s Slides**

**Coming-up In Next Slides**
Qemu Challenges: to Realize VCPU Hotplug (1)

1. Customer requirement: **x86_64 look-and-feel of VCPU Hotplug** on ARM (Different Architectures)
   - How to **fake VCPU Hot-plug at Qemu** when KVM & Guest Kernel don’t support it? **(Problem-1) ▶️**
   - QMP Command 'device_{add, del}' – How to use **same interface**?
     
     ```
     $ device_add host-arm-cpu, id=core4, core-id=4
     $ device_del core-id=4
     ```
   - VM **boot time** should be comparable to x86_64 (hot-plug should not degrade)

2. At VM init, Qemu **CPU Objects** MUST be created for each Host KVM VCPUs. **(KVM Constraint)**
   - What should we do to CPU Objects for 'yet-to-be-plugged' VCPUs (after Init)? **(Problem-2) ▶️**
     
     - Keep 'not-plugged' VCPUs realized? (CPU Threads will always exist – **Boot time**)?
     - Unrealized 'not-plugged' VCPU CPU objects can't be migrated (VMSD registration?)
     - Delete or keep them disabled after KVM initialization has finished?
Qemu Challenges: to Realize VCPU Hotplug (2)

3. Qemu MUST fully realize GIC V3 to correctly initialize Host KVM VGIC  (KVM Constraint → CPU Arch)
   - GIC V3 State MUST be pre-sized with all the 'possible' VCPUs (no escape?)
   - All redistributors state must pre-exists (even for not-plugged VCPUs)
   - GIC init, access operations or updates can only be performed on 'Enabled' VCPUs (Problem-3) >>

4. Guest MUST see all VCPUs as 'present' in MADT GICC CPU interface at boot (ARM CPU Arch Req)
   - Present ACPI _STA.PRESENT=1 _STA.ENABLED={0,1} to Guest, Always! (Problem-4) >>
   - 'not-plugged' Qemu VCPUs: show as 'disabled' i.e. ACPI _STA.ENABLED=0 (Conflicts with QOM State)
Qemu Challenges: to Realize VCPU Hotplug (3)

5. What if in future ARM CPU Arch supports Physical CPU Hotplug? (ARM CPU Arch says 'assume nothing')
   - Can Qemu implementation be generic enough in the following cases (Problem-5)  
     ❖ Virtual CPU Hotplug Only
     ❖ Physical CPU Hotplug Only
     ❖ Both Virtual & Physical Hotplug co-exists
   - How should Qemu distinguish these cases and tell Guest about compatibility?
     ❖ Should Guest evaluate(_OSC)?
Qemu Workarounds: Creative Hack – Which Works! Really? (1)

- Oh Yes! That thin Red line between Bravery/Madness, Bug/Feature, Creativity/Hack
  - Allow me to call it a Creative Hack? But is it Acceptable to the community? You decide!

Approach-1: To mitigate the 5 Qemu problems discussed earlier

1. Use existing `device_{add, del}` hot-plug device interface (Mitigation-1)

   - a new CPU object in Qemu shall be created for a hot-plugged VCPU
   - On VCPU hot-unplug, CPU object shall be destroyed
   - Update the GICV3 about this event (re-patching GICC <-> VCPU) (Deviation from ARM Arch. Okay?)

---

[1] Rough Qemu Boot Time Figures Presented At Linaro Open Discussion Meeting
Qemu Workarounds: Creative Hack – Which Works! Really? (2)

2. Creation, initialization and realization for {cold, hot-)plug VCPUs **(Mitigation-2)***

- During VM Init, create and initialize all 'possible' CPU objects

- Only 'realize' VCPUs which are cold plugged `(-smp cpus 4 maxcpus 6)` (CPU Threads)
  - Including cold-plugged VCPUs added using '-device' option
  - Will end up creating, initializing and running KVM VCPUs `{KVM_{CREATE_VCPU, ARM_CPU_INIT, RUN}}`

- **Do not** 'realize' un-plugged (will be 'hot plugged') VCPUs CPU Objects (i.e. No CPU Threads)
  - Only create & initialize KVM VCPU at Host KVM `{KVM_CREATE_VCPU, KVM_ARM_VCPU_INIT}`
  - Above strategy also drastically saves Boot time [1] (a requirement for Kata Containers)

---

[1] Rough Qemu Boot Time Figures Presented At Linaro Open Discussion Meeting

KVM Forum 2023
Qemu Workarounds: Creative Hack – Which Works! Really? (3)

3. Qemu GICV3 full Realization (Mitigation-3)

- No Workarounds for below (ARM CPU Architecture Constraint)
  - Pre-size GICv3State with 'possible' vCPUs but use only 'enabled' (no escape here)
  - Create all the GICV3 Redistributors for all ‘possible’ vCPUs both in Qemu & KVM
    - All related Memory Regions will be created in Qemu and initialized in KVM

- Some subtle changes are required for realization, initialization, access and updates for GICV3
  - GICV3 realization: only assign ‘enabled’ vCPUs [GICv3State->GICv3CPUState->CPUState]
  - Initialize the GICV3 CPU interface registers to KVM values (on vCPU reset)
    - GICv3 CPU interface Reginfo (ICC_CNTL_EL1) for newly plugged vCPUs need to be init again.
    - .resetfn(): fetch value of ICC_CNTL_EL1 from KVM.
Qemu Workarounds: Creative Hack – Which Works! Really? (4)

- Some **subtle changes** are required for realization, initialization, access and updates for GICV3 (Contd.)
  - Restrict VCPU updates for re-patching GICv3CPUState to 'enabled' VCPUs (plug/hotplug event)
  - Restrict KVM device access (get/put) of GICC registers to 'enabled' VCPUs (kvm_gic{c,d,r}_access)
    - Why? State sync up for example for pre_save, post_load, reset_hold (putting back to kvm) etc.

4. What **update** does on VCPU {Un-}Plug events?
  - Match GIC CPU Interface: ‘mp-affinity’ ⇔ GiCR_TYPER (All the GIC CPU Interface)
    - Match Found? VCPU Hot Unplug Event : VCPU Hotplug Event
  - Adjust: GIC CPU Interface VCPU = Hot plugged VCPU (Deviation from ARM CPU Arch. Okay?)
  - Initialize GIC CPU Interface Registers info again in Qemu (VCPU Hotplug)
Qemu Workarounds: Creative Hack – Which Works! Really? (5)

5. Guest can defer online'ing/enabling the VCPU after boot. [ARM Arch Complaint] (Mitigation-4)
   - Use ACPI MADT GICC CPU Interface flag 'online-capable' Bit [1] in Qemu.
   - VCPUs in Guest can be brought '{on, off}line' when Qemu hot-{un}plugs them
     - Will involve ACPI hot-plug handshake between Qemu and Guest Kernel
     - Handshake events: Device-Check/Eject-Request/Eject Ej0/_STA

6. Qemu MUST distinguish between ACPI _STA.PRESENT and _STA.ENABLED. Update Guest Kernel as,
   - _STA.PRESENT=1 and _STA.ENABLED=1 for 'plugged' VCPUs in Qemu
   - _STA.PRESENT=1 and _STA.ENABLED=0 for 'not-plugged' VCPUs in Qemu (Faking Present – Okay?)

7. Implement _OSC AML method to return to the guest about (Mitigation-5)

   - Hotplug 'Enabled' or 'Present' Support

---

[1] ACPI specification 6.5 "Table 5.37: GICC CPU Interface Flags"
VM Init in A Nutshell: Qemu, KVM, Kernel (AARCH64)

Sequence Depicting VM & Guest Kernel Initialization (AARCH64) (no Hotplug Support)

AARCH64 Does Not Support Physical CPU Hotplug

VM Init Sequence: Qemu, KVM, ACPI, Kernel (AARCH64)
Comparison: Virtual CPU Hotplug ACPI Event Seq (x86_64 vs AARCH64)

Guest OS

ACPI

QEMU

CPU Control Device

GEM Device

QEMU

KVM Interface

Host OS/KVM

Hardware

Supports Physical CPU Hotplug

Sequence Depicting ACPI Driven vCPU Hotplug Action (x86_64)

Sequence Depicting ACPI Driven vCPU Hotplug Action (AARCH64)

Hotplug Sequence: Qemu, KVM, ACPI, Kernel (AARCH64)

Comparison: Virtual CPU Hot-Unplug ACPI Event Seq (x86_64 vs AARCH64)

Sequence Depicting ACPI Driven vCPU Hot-Unplug Action (x86_64)

QEMU

KVM KVM Interface (Unplugged QEMU KVM VCPUs Are Parked Here)

QEMU

KVM ACPI AML Code (Arch Agnostic)

QEMU

QEMU


cpu_hotplug_wri()
switch (addr) {
case ACPI_CPU_SELECTOR_OFFSET_RW:
case ACPI_CPU_FLAGS_OFFSET_RW:
case ACPI_CPU_CMD_OFFSET_RW:

cpu_hotplug_wri()
switch (addr) {
case ACPI_CPU_SELECTOR_OFFSET_RW:
case ACPI_CPU_FLAGS_OFFSET_RW:
case ACPI_CPU_CMD_OFFSET_RW:

qdev_device_del

qdev_device_del

Sequence Depicting ACPI Driven vCPU Hot-Unplug Action (AARCH64)

KVM VCPUs And Associated LAPIC Are Never Destroyed

All VCPUs And VGIC Are Fully Initialized Once At VM Init (For VM Life Time)

Supports Physical CPU Hotplug

Does Not Support Physical CPU Hotplug

Hidden Backup Slide For Discussion Only: (Not To Be Presented)
Hotplug Sequence: Qemu, KVM, ACPI, Kernel

(AARCH64)

- Qemu
- KVM
- ACPI
- Kernel

Hotplug Sequence:

1. Qemu
2. KVM
3. ACPI
4. Kernel

Hidden Backup Slide For Discussion Only: (Not To Be Presented)
Qemu VCPU Realization involves spawning of vCPU threads and KVM_RUN IOCTL calls which can be avoided for unplugged VCPUs. This improves VM Init time and mem footprint. Can we can defer spawning of Qemu threads even for plugged till they are online’d using SMCC Exit calls to Qemu? (scope of optimization? – an experiment-in-progress!)

Below figures were taken to measure the impact of the VCPU realization i.e. spawning of Qemu threads and KVM_RUN IOCTL on the VM init time. Total possible vCPUS = 128. ‘N’ Realized VCPUs = ‘N’ vCPU Threads in Qemu.

<table>
<thead>
<tr>
<th>SN</th>
<th>Stage of VM Init</th>
<th>Realized = 1, Unrealized = 127</th>
<th>Realized = 128, Unrealized = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pre vCPU Init</td>
<td>~25 msec</td>
<td>~40 msec</td>
</tr>
<tr>
<td>2</td>
<td>Post vCPU Init (But Pre-VGIC Init)</td>
<td>~40 msec (+15)</td>
<td>~240 msec (+200)</td>
</tr>
<tr>
<td>3</td>
<td>Post GIC Init</td>
<td>~70 msec (+30)</td>
<td>~340 msec (+100)</td>
</tr>
<tr>
<td>4</td>
<td>Pre Linux Kernel Load</td>
<td>~90 msec (+20)</td>
<td>~360 msec (+20)</td>
</tr>
<tr>
<td>5</td>
<td>Post Kernel Load (But Pre SMBIOS setup)</td>
<td>~120 msec (+30)</td>
<td>~410 msec (+50)</td>
</tr>
</tbody>
</table>

[1] VM Init Time Discussion; Linaro Open Discussions Date: 22-03-2021
1. [PATCH RFC 0/4] Changes to Support *Virtual* CPU Hotplug for ARM64 (Salil Mehta/Xiongfeng Wang)
   o https://lore.kernel.org/all/20200625133757.22332-1-salil.mehta@huawei.com/

2. [PATCH RFC 00/22] Support of Virtual CPU Hotplug for ARMv8 Arch (Salil Mehta/Keqian Zhu)
   o https://lore.kernel.org/qemu-devel/20200613213629.21984-1-salil.mehta@huawei.com/

3. Latest Qemu Prototype (Pre RFC V2)
   o https://github.com/salil-mehta/qemu.git virt-cpuhp-armv8/rfc-v1-port11052023.dev-1 (Salil Mehta)

4. Latest Kernel Prototype (Pre RFC V2 = RFC V1 + Fixes)
   o https://git.gitlab.arm.com/linux-arm/linux-jm.git virtual_cpu_hotplug/rfc/v2 (James Morse)
Acknowledgements

1. Marc Zyngier (Google)
2. James Morse (ARM)
3. Jean-Philippe Brucker (Linaro)
4. Lorenzo Pieralisi (Linaro)
5. Keqian Zhu (Huawei)
6. Xiongfeng Wang (Huawei)
7. Jonathan Cameron (Huawei)
8. Shameerali Kolothum Thodi (Huawei)
9. Will Deacon (Google)
10. Catalin Marinas (ARM)
11. Russel King (Oracle)
12. Miguel Luis (Oracle)
13. Vishnu Pajjuri (Ampere)
14. Sudeep Holla (ARM)
15. Andrew Jones (Redhat)
16. Igor Mamedov (Redhat)
17. Gavin Shan (Redhat)
18. Darren Hart (Ampere)
19. Ilkka Koskinen (Ampere)
20. Karl Heubaum (Oracle) and all those I have missed out!
Comments: We Can Discuss?

Q & A
Thanks

To the Wonderful Audience & All The Organizers of The KVMForum2023