QEMU CPU Hotplug

Bharata B Rao, IBM India
<bharata@linux.vnet.ibm.com>
David Gibson, Red Hat Australia
<dgibson@redhat.com>
Igor Mammedov, Red Hat Czech Republic
<imammedo@redhat.com>

KVM Forum 2016

Guest CPU Hot-plug

- Add / remove virtual CPUs in a VM
    - Guest is running
    - No reboot
- Scale guest compute capacity on demand
- Useful for vertical scaling in Cloud
- Requires guest awareness
    - Protocol depends on platform
        - ACPI (x86 & ARM)
        - PAPR events (POWER)

What we had (v2.6 and earlier)

- **cpu-add** QMP command
  - Only implemented on x86
  - No unplug
- No generic CPU hot-plug model
  - **cpu-add** always added a single vCPU thread
  - Not compatible with hotplug protocol on some platforms
  - **cpu-add** “out of order” breaks migration
- Not based on standard **-device** / **device_add** interfaces
  - Doesn’t match hotplug model used for other devices
- No way to query for possible CPUs
  - Requires assumptions about how **-smp** is interpreted
  - Not valid for all platforms

What we wanted

- Consistent QOM model for CPUs
- CPU hotplug with standard **device_add**
- Support for many architectures / targets
- Support for many machine types
    - pc / q35
    - pseries
    - S390
    - ARM / aarch64?
- Possible CPUs introspection
    - Management needs to know what to **device_add**

Hotplug Granularity

Thread

- Matches **cpu-add**
  - Existing guest tools
  - Existing management
- Most flexible

- Impossible on ‘pseries’
  - Guest events have no way to express this

Core

- Matches PAPR model

- Little reason on other platforms

Socket

- Matches hardware
  - Probably...

- Inflexible
- “Socket” may be artificial
  - pseries
  - aarch64 virtual platform

Hotplug Granularity (2)

- Machine type defines hotplug granularity
  - Thread
    - pc / q35 (matches ACPI protocol)
    - s390
  - Core
    - pseries (matches PAPR protocol)
  - Socket
    - Nothing yet (but matches plausible real hardware)
  - Multi-chip module?
  - Daughterboard?

CPU QOM Model

- vCPU thread is a QOM object (already)
  - Couldn’t be user instantiated
- Hotpluggable CPU module is also QOM object
  - Added with **-device** or **device_add**

➢ Sometimes the same object..

- thread granularity

```
(qemu) info qom-tree
/machine (pc-i440fx-2.7-machine)
  /peripheral (container)
    /cpu1 (qemu64-x86_64-cpu)
```

➢ ..sometimes not

- other granularity

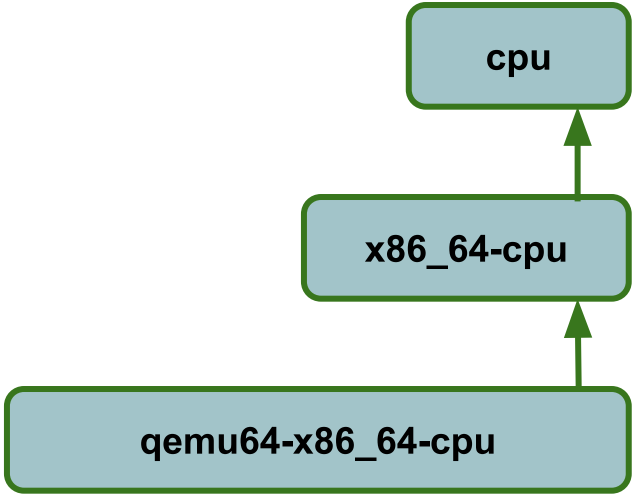
```
(qemu) info qom-tree
/machine (pseries-2.7-machine)
  /peripheral (container)
    /core1 (POWER8E_v2.1-spapr-cpu-core)
      /thread[0] (POWER8E_v2.1-powerpc64-cpu)
```

CPU QOM Model (2)

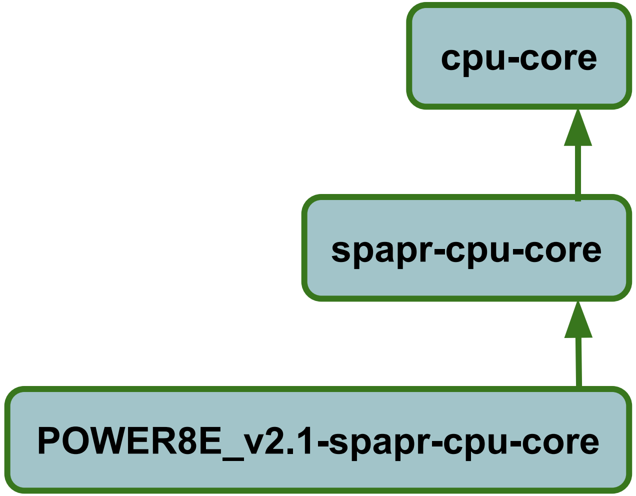
- Could be additional QOM objects
  - Sockets, modules etc.
  - Decided by machine type
  - No examples yet
- Machine type converts **-smp** and **-cpu** into initial QOM objects
  - But could be extended for heterogeneous boards
- Abstract **cpu-core** class introduced
  - sPAPR uses this as base class for sPAPR specific types
  - .. can be re-used by future platforms

CPU Type Hierarchy Examples

pc (x86) type hierarchy



pseries type hierarchy



The new CPU device semantics

- **-device CPU-device-type[,socket-id=][,core-id=][,thread-id=]**
  - CPU-device-type is machine-dependent
- sPAPR
  - **-device POWER8_v2.0-spapr-cpu-core,core-id=8**
    - Only core-id needs to be specified
- X86
  - **-device qemu64-x86_64-cpu,socket-id=2,core-id=0,thread-id=0**
    - Need to specify thread-id, core-id and socket-id

Discovery and introspection

How would we know what CPU objects to create ?

- **query-hotpluggable-cpus**
    - QMP interface
    - Lists information management needs to hot plug:
        - Device type for **device_add**
            - Depends on machine type and “-cpu cpu_model”
            - Might depend on other parameters
        - Device properties for each CPU
            - thread-id, core-id, socket-id, node-id
            - Future machine types might use more
    - Lists both initial and possible CPUs
- **info hotpluggable-cpus** (HMP wrapper)

Demonstration

- Example of info hotpluggable-cpus and device_add device_del
- Pseries with multiple SMT modes
- X86

sPAPR PowerPC semantics - single threaded guest

```
-smp 1,maxcpus=2

(qemu) info hotpluggable-cpus
Hotpluggable CPUs:
  type: "host-spapr-cpu-core"
  vcpus_count: "1"
  CPUInstance Properties:
    core-id: "1"
  type: "host-spapr-cpu-core"
  vcpus_count: "1"
  qom_path: "/machine/unattached/device[1]"
  CPUInstance Properties:
    core-id: "0"
(qemu) device_add host-spapr-cpu-core,id=core1,core-id=1
(qemu) device_del core1
```

sPAPR PowerPC semantics - SMT4 guest

```
-smp 4,cores=2,threads=4,maxcpus=8 -cpu POWER8E
```

```
(qemu) info hotpluggable-cpus
Hotpluggable CPUs:
  type: "POWER8E_v2.1-spapr-cpu-core"
  vcpus_count: "4"
  CPUInstance Properties:
    core-id: "4"
  type: "POWER8E_v2.1-spapr-cpu-core"
  vcpus_count: "4"
  qom_path: "/machine/unattached/device[1]"
  CPUInstance Properties:
    core-id: "0"
(qemu) device_add POWER8E_v2.1-spapr-cpu-core,id=core1,core-id=4
(qemu) device_del core1
```

sPAPR PowerPC semantics - SMT8 guest

```
-smp 8,cores=2,threads=8,maxcpus=16

(qemu) info hotpluggable-cpus
Hotpluggable CPUs:
  type: "host-spapr-cpu-core"
  vcpus_count: "8"
  CPUInstance Properties:
    core-id: "8"
  type: "host-spapr-cpu-core"
  vcpus_count: "8"
  qom_path: "/machine/unattached/device[1]"
  CPUInstance Properties:
    core-id: "0"
(qemu) device_add host-spapr-cpu-core,id=core1,core-id=8
(qemu) device_del core1
```

Problems: KVM and CPU removal

- KVM doesn't support destroying vCPU instances
  - … and allowing it to do so looks difficult
- Alternative approach
  - Destroy CPU object at QEMU side
  - Keep KVM vCPU instance in “parked” state
  - Re-use “parked” KVM vCPU instance when the same CPU is next plugged

Problems: Handling errors during hotplug

- CPU **realize()**
  - Can cleanly report errors and abort
  - .. but can’t easily check machine imposed constraints
- Machine **plug()** handler
  - CPU is already realized
    - Tricky or impossible to rollback
    - Too late to set additional CPU properties
- New: Machine **pre_plug()** handler
  - Called before **realize()**
  - Validates properties against machine model
    - Can also set extra properties determined by machine
  - Detects problems early, no rollback

Problems: CPU Options

- Many platforms have optional CPU properties
  - X86 available features
  - POWER compatibility mode
- Usually need to be the same for all CPUs
  - So adding to every **device_add** is tedious and redundant
- **-global** provides a natural way to set properties uniformly
  - Works for both initial and hot added CPUs
  - Allows flexibility if we allow non-uniform CPUs in future
- Need to convert **-cpu** options to **-global** properties
  - Where this is done depends on platform
  - Needs further cleanup

Problems: Migration nightmares

- cpu_index was allocated in cpu_exec_init()
  - Value depended on CPU instantiation order
  - Used as migration instance id
- Migration requires matching instance ids on source and destination
  - No reasonable way to ensure identical hotplug / unplug order on source and destination
  - Out of order hotplug or unplug would break migration afterwards
    - Already broken on x86 with **cpu-add**
- Devised a stable cpu_index scheme with minimal impact on archs
  - Machine type can generate cpu_index values before CPU realize()
  - To support CPU hotplug, machines should assign stable values manually
    - sPAPR uses core-id to generate thread cpu_index values
  - Machines that don't support CPU hotplug can still use old auto-assignment
    - Minimal changes until necessary

Future work: NUMA

- Management has to guess which NUMA nodes hotplugged CPUS will be in
  - Already a problem with **cpu-add**
- **-numa** command line option isn't enough
  - Management can't know CPU indexes to use until it has run **query-hotpluggable-cpus**
- Possible solution:
  - QMP command to assign a CPU object (socket / core / thread) to a NUMA node at run time
    - Start QEMU in stopped mode '-S'
    - Use query-hotpluggable-cpus to get list of possible cpus
    - Assign NUMA nodes to each CPU
    - Start guest with 'continue'

Future Work: More machine types

- S390
  - Recently implemented **cpu-add**, move to new model
- ARM / aarch64
  - Some machine types will support hotplug
- powernv
  - In-progress “bare metal” (not paravirtualized) POWER machine
  - May require interactions with other devices on the physical CPU chip
- Prerequisites:
  - cpu_exec_init() and cpu_exec_exit() need to be called at realize / unrealize
    - Already done for x86, s390 and ppc
    - Necessary for handling failures
    - Necessary for manual cpu_index allocation

Future work: POWER specific

- Clean up device tree creation:
  - Device tree represents cores, not threads
  - Currently constructed by 1st thread
  - Should construct from core device, now that it’s a real object
- DRC state migration
  - “Dynamic Reconfiguration Connector”
    - Paravirtual abstraction to communicate hotplug state with guest
  - Not all state currently migrated
    - Concurrent migration and hotplug events can break

Future work: Other

- libvirt support for new CPU hotplug interface (Peter Krempa)
  - First, existing libvirt API in terms of new QEMU API
    - Limited, but helps existing tools
  - Then, new libvirt API
    - More flexible
- -smp rework (Andrew Jones)
  - Convert -smp,sockets=S,cores=C,threads=T into machine properties
  - Removes reliance on global variables for topology
  - Allows machine types to define or override **-smp** parsing
- Support boot cpu removal
  - Assorted places in QEMU assume the existence of CPU 0

Legal

- This work represents the view of the authors, and does not necessarily represent the view of IBM or of Red Hat
- IBM is a trademark of International Business Machines in the United States and/or other countries
- Red Hat is a trademark of Red Hat Inc. in the United States and/or other countries
- Linux is the registered trademark of Linus Torvalds
- Other company, product and service names may be trademarks of others
- This document is provided “AS IS”, with no express or implied warranties. Use the information in this document at your own risk