Nested Virtualization
Terminology

Host Hypervisor

Guest Hypervisor

Nested VM

App

Kernel

VM

App

App

Nested VM

App

Kernel

L0

L1

L2

Hardware
Use Cases

1. IaaS hosting private clouds
2. Test your hypervisor in a VM
3. Debug your hypervisor in a VM
4. Develop hypervisors using a cloud
Theorem 2

“A conventional third generation computer is recursively virtualizable if it is: (a) virtualizable, and (b) a VMM without any timing dependencies can be constructed for it.”

Formal requirements for virtualizable third generation architectures [Popek and Goldberg ’74]
Recursively Virtualizable

- Only applies to virtualizable architectures
- ARM and x86 are not virtualizable
- Hardware support for virtualization
ARM Virtualization Extensions

EL0
User

EL1
Kernel

EL2
Hypervisor
ARM Virtualization Extensions

EL0
VM
User Space
Kernel

EL1
VM
User Space
Kernel

EL2
Hypervisor
ARM Nested Virtualization

EL0
- User Space

EL1
- Kernel

Virtual EL2
- Guest Hypervisor

EL2
- Host Hypervisor

EL0
- User Space

EL1
- Kernel

Virtual EL2
- Guest Hypervisor
ARM Nested Virtualization

EL0
User Space

EL1
Kernel

EL ??
Guest Hypervisor

EL2
Host Hypervisor
ARM Nested Virtualization

- EL0
  - User Space
  - Guest Hypervisor

- EL1
  - Kernel

- EL0
  - Host Hypervisor

- El2
  - Guest Hypervisor
  - User Space

Trap-and-emulate
ARM Nested Virtualization

- EL0
  - User Space
- EL1
  - Kernel
  - Guest Hypervisor
- EL1
  - Kernel
  - Guest Hypervisor
- EL2
  - Host Hypervisor

?? -and-emulate
ARMv8.3

- Supports running the guest hypervisor in EL1
- HCR_EL2.NV:
  - Traps EL2 operations executed in EL1 to EL2
  - Traps `eret` to EL2
KVM/ARM Nested Virtualization

- CPU Virtualization
- Memory Virtualization
- Timer Virtualization
- Interrupt Virtualization
Nested CPU Virtualization

```c
struct kvm_cpu_context {
    u64 sys_regs[NR_SYS_REGS];
    u64 el2_regs[NR_EL2_REGS];
}

struct kvm_vcpu_arch {
    ...
    struct kvm_cpu_context ctxt;
}
```
Hypervisor-VM Switch

Host

EL0

App

App

VM

EL1

App

App

EL2

Linux

KVM

Kernel

Save EL1 sys_regs

Restore EL1 sys_regs
Hypervisor-Hypervisor Switch

Host

EL0

App
App

EL1

Linux
KVM

VM

Kernel

Guest Hypervisor

Save/restore EL1 sys_regs

Save/restore el2_regs
Emulating EL2 in EL1

- Define mapping of EL2 registers to EL1 registers
- Example: TTBR0_EL2 to TTBR0_EL1
- Example: SCTLR_EL2 adapted to SCTLR_EL1
- Shadow EL1 registers
Shadow Registers

PSTATE.mode == EL0/1

PSTATE.mode == EL2

&sys_regs

&shadow_sys_regs

u64 *vcpu->ctxtx.hw_regs
Virtual Exceptions

- Trap to virtual EL2
- “Forward” exceptions
- Emulate virtual exceptions
Virtual Exceptions

- Returning from virtual EL2
- Trap `eret` to EL2 (ARMv8.3)
- Emulate virtual exception return
KVM/ARM Nested Virtualization

- CPU Virtualization
- Memory Virtualization
- Timer Virtualization
- Interrupt Virtualization
Memory Virtualization

Virtual Address (VA) --> Physical Address (PA)
Memory Virtualization

1. Virtual Address (VA) → Stage 1: VM kernel
2. Intermediate Physical Address (IPA) → Stage 2: Hypervisor
3. Physical Address (PA)
Nested Memory Virtualization

- Virtual Address (VA)
- Intermediate Physical Address (IPA)
- Nested Intermediate Physical Address
- Physical Address (PA)

Stage 1: Nested VM kernel
Stage ?: Guest hypervisor
Stage 2: Host hypervisor
Nested Memory Virtualization

- Virtual Address (VA)
- Intermediate Physical Address (IPA)
- Physical Address (PA)

Stage 1: Nested VM kernel
Stage 2: Host hypervisor

Shadow Stage 2 Page Table
Shadow Stage 2 Page Tables

• Translate IPA to PA

• Entries are created by host KVM by walking guest hypervisor stage 2 page tables in software
KVM/ARM Nested Virtualization

- CPU Virtualization
- Memory Virtualization
- Timer Virtualization
- Interrupt Virtualization
Nested Timer Virtualization

- ARM provides a virtual and physical timer in EL1
- EL2 provides a separate EL2 “hyp” timer
- KVM must emulate a VM with EL2 and the hyp timer
KVM/ARM Nested Virtualization

- CPU Virtualization
- Memory Virtualization
- Timer Virtualization
- **Interrupt Virtualization**
ARM Generic Interrupt Controller (GIC)
ARM Generic Interrupt Controller (GIC)
Nested Interrupt Virtualization

• Deliver both virtual and nested virtual interrupts using the GIC

• Multi-level virtualization using single-level virtualization hardware [Turtles - OSDI ‘10]
Nested Interrupt Virtualization

- Shadow LRs

- Guest hypervisor traps when attempting to program virtual LRs

- Host hypervisor handles traps by writing to shadow LRs

- Hardware uses shadow LRs when running the nested VM
Implementation Status

- RFC v1 on @kvmarm by Jintack Lim (Columbia University)
- CONFIG_KVM_ARM_NESTED_HYP
- vcpu->arch.features & KVM_ARM_VCPU_NESTED_VIRT
To Do

• Must expose EL2 registers to user space

• Mostly scattered out over existing files. Should we try to isolate more?

• Hard-coded addresses and interrupt numbers

• Reverse map for shadow stage 2 page tables

• More efficient emulation of TLBI instructions

• Get rid of config option and use command line parameter instead

• Hypercalls from the VM vs. virtual self-hypercalls and PSCI
Questions?

and please review the patches…
Backup Slides
KVM/ARM Nested Virtualization

CPU Virtualization - VHE

- VHE is fun with nested virtualization
- We don’t set the E2H bit
- The VM thinks it runs in EL2 using VHE so uses EL1 register accesses to access EL2 registers
- But really does run in EL1 and doesn’t need to trap except on a few registers with different bit configuration
KVM/ARM Nested Virtualization

Memory Virtualization - VMIDs and ASIDs

• EL2 is separate translation regime
  • No ASIDs - cannot alias with EL1 translations
  • VMID not used

• Emulating virtual EL2 in EL1
  • Separate VMID for virtual EL2
  • Always use ASID 0