The Turtles Project: Design and Implementation of Nested Virtualization

"You're very clever, young man, very clever," said the old lady. "But it's turtles all the way down!"

Presenter: Naser AlDuaij

The Turtles Project: Nested Virtualization

- Running multiple unmodified hypervisors/VMs
- Single level (non-nested) hypervisor is modified
- Battle against performance degradation

The Turtles Project: Motivation

- Popular OS have hypervisors built in
- Platforms with hypervisors embedded in f/w
- Live migration of hypervisors with their Vms
- Increased security
- Testing/Debugging/Benchmarking hypervisors

The Turtles Project: Making it work

- CPU virtualization: Forward traps to upper levels
- Memory virtualization: Multi-dimensional paging
- I/O virtualization: Multi-level device assignment

The Turtles Project: Architecture

- IBM Sys z: Multi-level arch. support for nested virtualization (each hypervisor deals with guest trap)
- VMX/SVM: Single-level arch. support for nested virtualization (all traps to level 0)
- Only level 0 can use VMX instructions (emulation)
- VMX transitions: VMEntry/VMExit for guest/root

The Turtles Project: Architecture

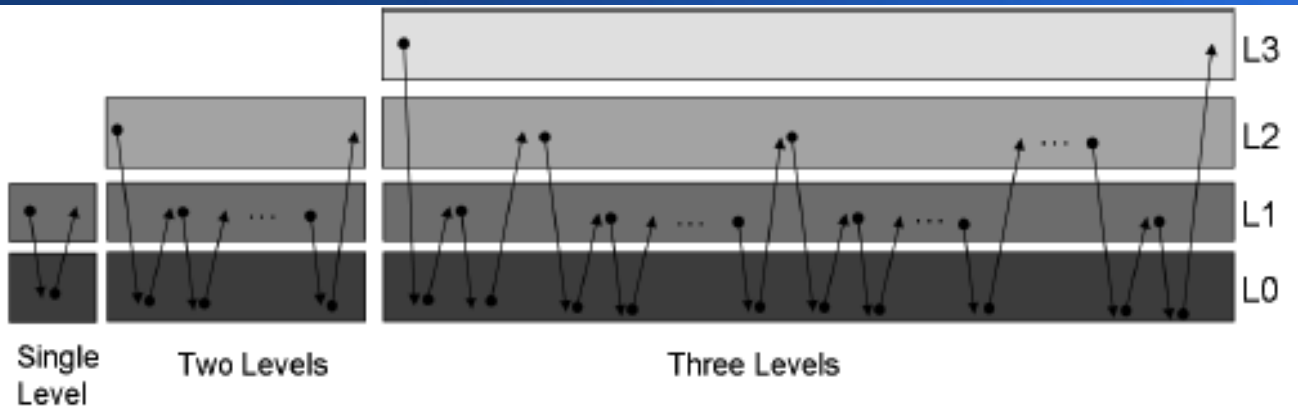

Figure 1: Nested traps with single-level architectural support for virtualization

The Turtles Project: Multiplexing

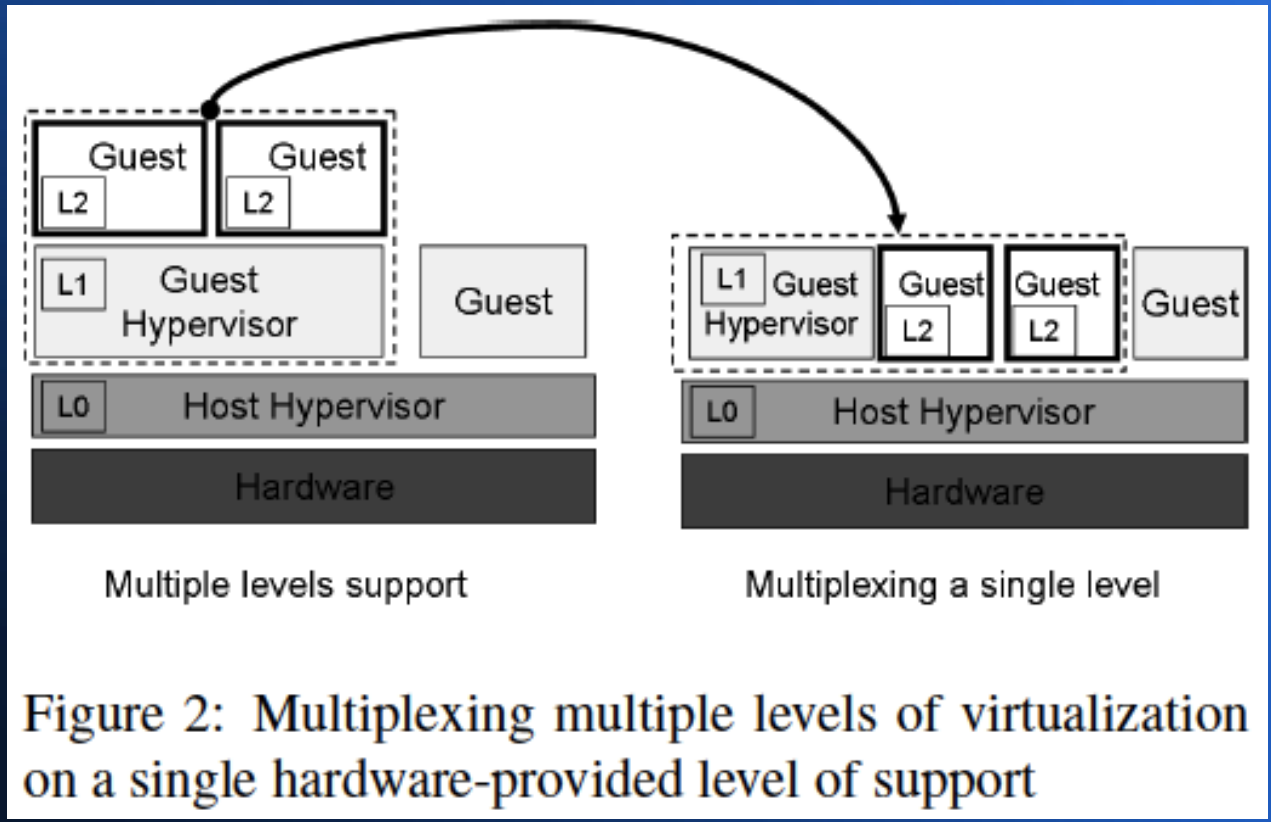

Figure 2: Multiplexing multiple levels of virtualization on a single hardware-provided level of support

The Turtles Project: CPU Nested Virtualization

- Only hypervisor (L0) runs in root mode
- VMCS/VMCB are control structures/blocks in memory
- VMCS divided into three groups:
  - Guest state (Virtual CPU registers)
  - Host state (Real CPU registers)
  - Control Data (To inject events into VMs)

The Turtles Project: CPU Nested Virtualization

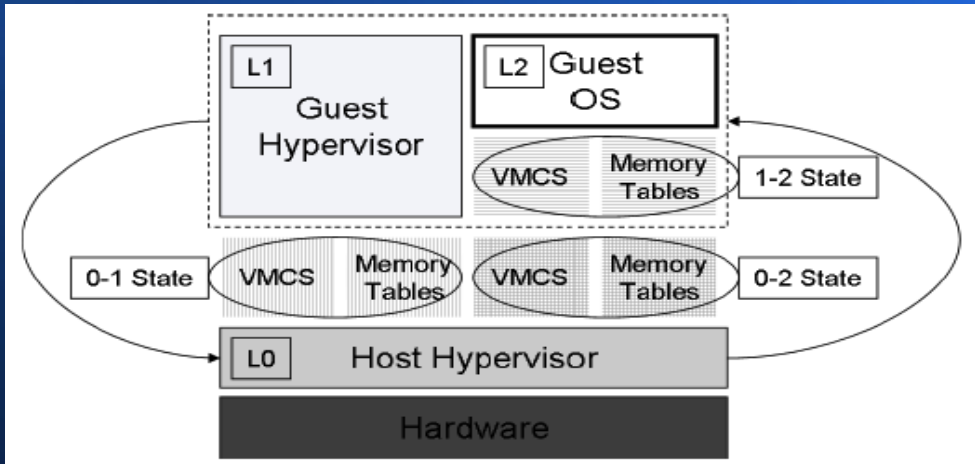

Figure 3: Extending VMX for nested virtualization

- VMCS 1-->2 is never loaded into the processor but is used by L0 to emulate a VMX for L1
- VMCS 0-->2 constructed using VMCS 1-->2

The Turtles Project: CPU Nested Virtualization

- L1 uses VMX instructions to load L2
- Causes VMExits
- L0 traps and emulates VMX instructions by L1
- vmlaunch/vmresume to launch a [new] VM
- VMExit from L1 to L0, VMEntry from L0 to L2

The Turtles Project: MMU virtualization

- Implementations for page table translation
  - Shadow-on-shadow: Slowest, useful if system does not support 2D page tables
  - Shadow on EPT: Straightforward. Support only for single-level EPT. Still considerable overhead
  - EPT on EPT: Multi-dimensional page tables pure HW

The Turtles Project: MMU Virtualization

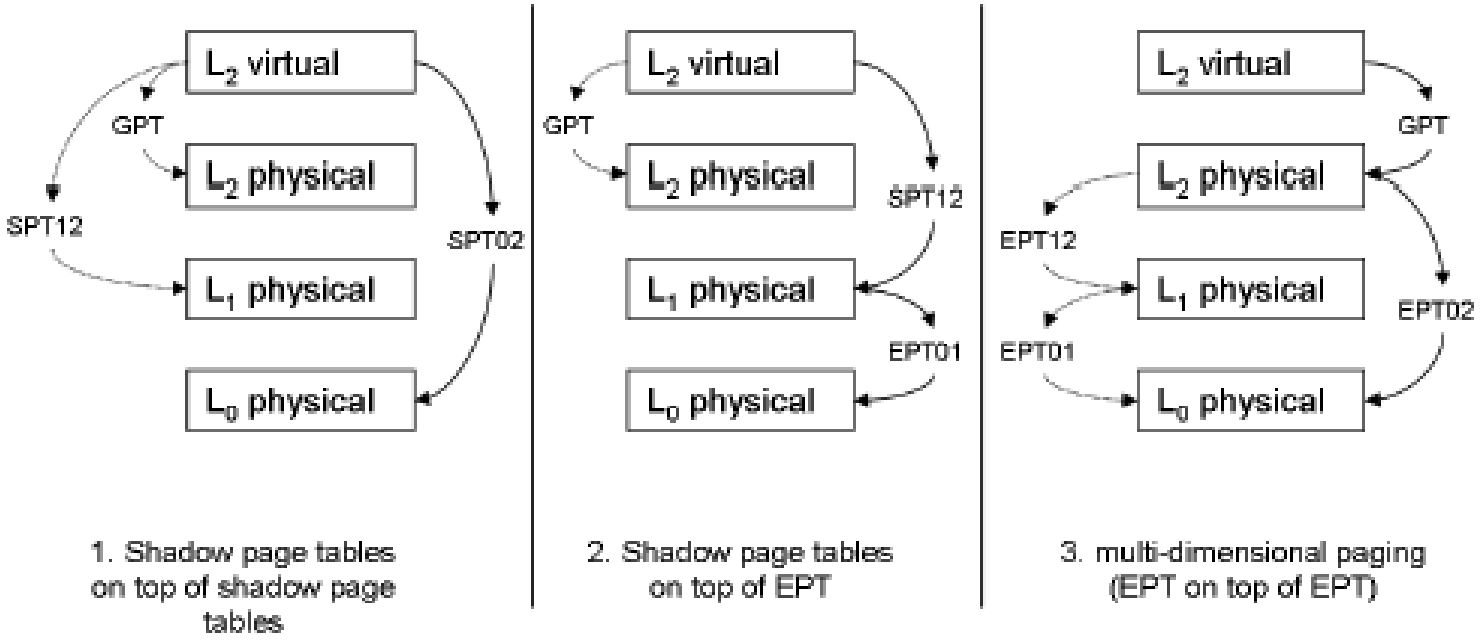

Figure 4: MMU alternatives for nested virtualization

The Turtles Project: I/O Vitualization

- Emulation: Emulates a known device- Guest uses an unmodified driver to interact with it
- Paravirtualization: Driver installed in guest
- Device Assignment: Direct access to the hardware

The Turtles Project: I/O Vitualization

- DMA is complicated with guest physical addresses
- IOMMU: HW component with a table of addresses from the hypervisor
- Allow multi-level virtualization by compressing multiple tables into one
- Better solution: Emulate IOMMU

The Turtles Project: Micro Optimizations

- Optimizations to VMCS merging code (not sig.)
  - Only perform a copy of VMCS if relevant values were modified
  - Copy multiple VMCS fields at once
- Exit-handling code slower due to additional exits
  - Reduce number of unnecessary traps
  - Direct Read/Write optimization (DRW)

The Turtles Project: Evaluation

- KVM
- kernbench: CPU, memory, and I/O intensive
- SPECjbb: Mainly CPU intensive
- Testing environments
  - Bare-metal (host), single-level, nested, nested DRW

The Turtles Project: Evaluation

Kernbench				
	Host	Guest	Nested	Nested$_{DRW}$
Run time	324.3	355	406.3	391.5
STD dev.	1.5	10	6.7	3.1
% overhead vs. host	-	9.5	25.3	20.7
% overhead vs. guest	-	-	14.5	10.3
%CPU	93	97	99	99

SPECjbb				
	Host	Guest	Nested	Nested$_{DRW}$
Score	90493	83599	77065	78347
STD dev.	1104	1230	1716	566
% degradation vs. host	-	7.6	14.8	13.4
% degradation vs. guest	-	-	7.8	6.3
%CPU	100	100	100	100

Table 2: `kernbench` and `SPECjbb` results

Benchmark	% overhead vs. single-level guest
`kernbench`	14.98
`SPECjbb`	8.85

Table 3: VMware Server as a guest hypervisor

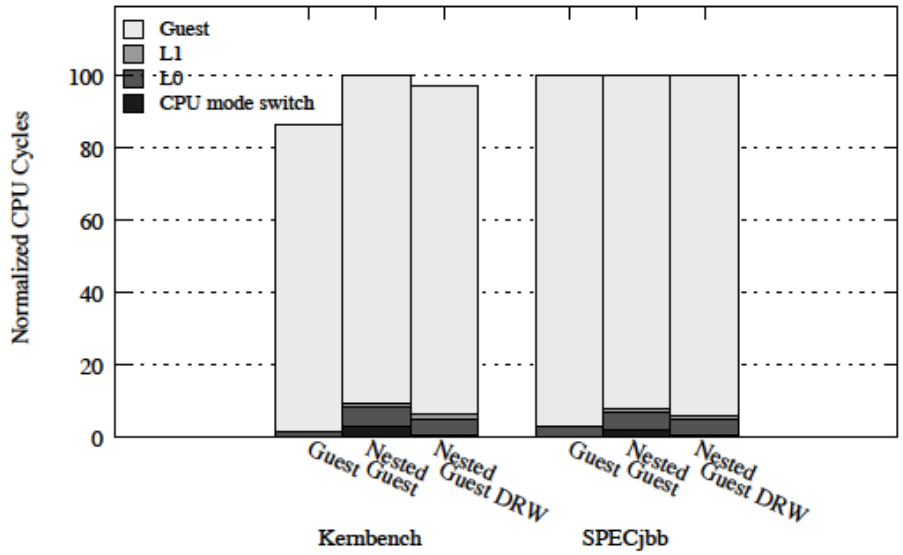

Figure 5: CPU cycle distribution

The Turtles Project: Evaluation

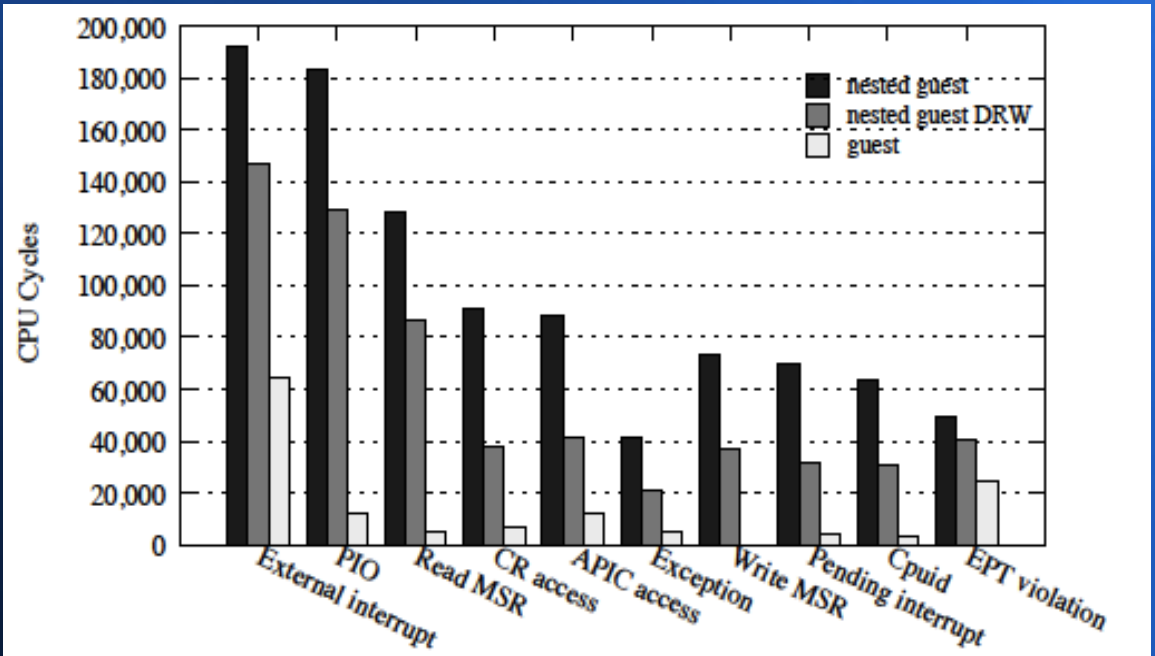

Figure 6: Cycle costs of handling different types of exits

The Turtles Project: I/O Evaluation

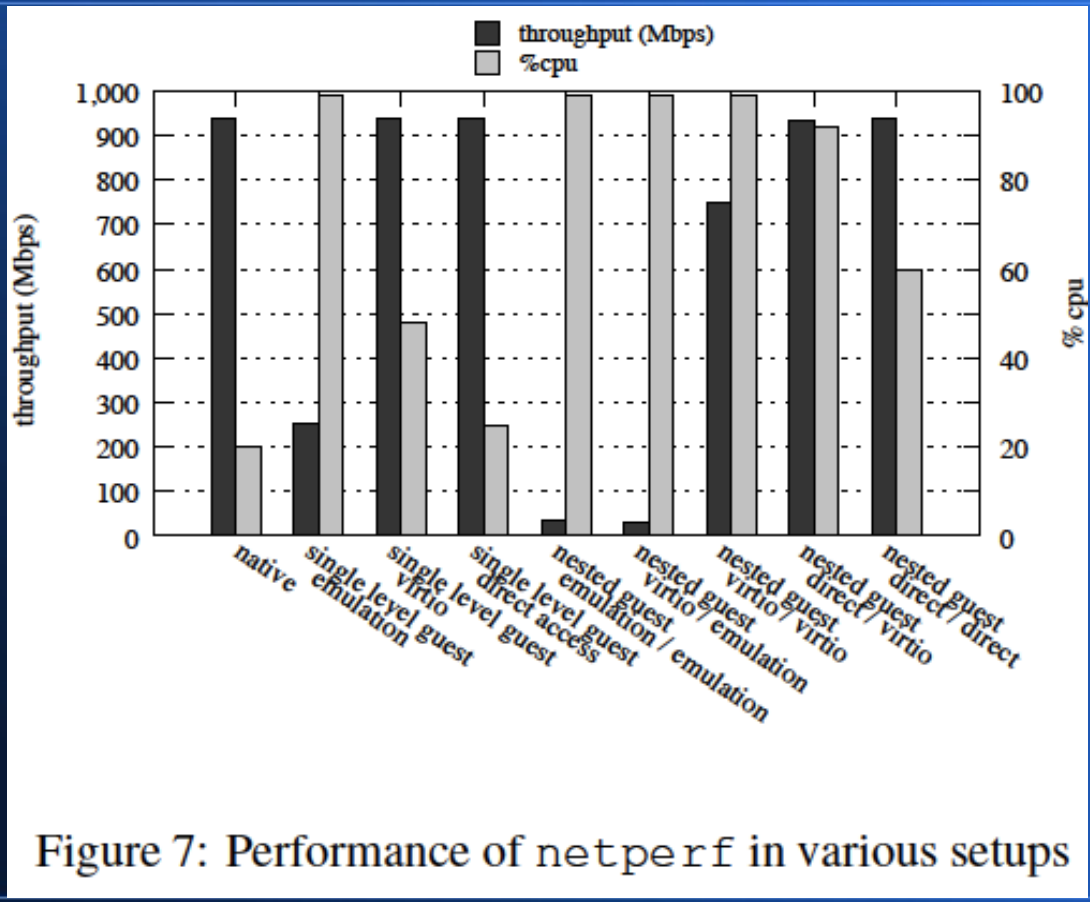

Figure 7: Performance of `netperf` in various setups

The Turtles Project: I/O Evaluation

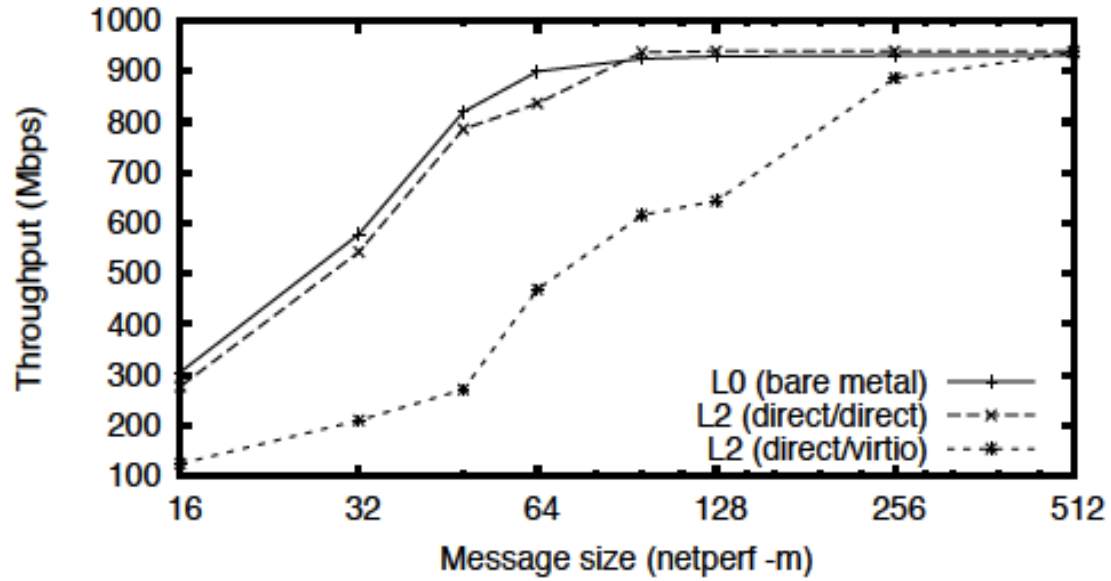

Figure 8: Performance of `netperf` with interrupt-less network driver

The Turtles Project: MMU Evaluation – Page faults cause the exits

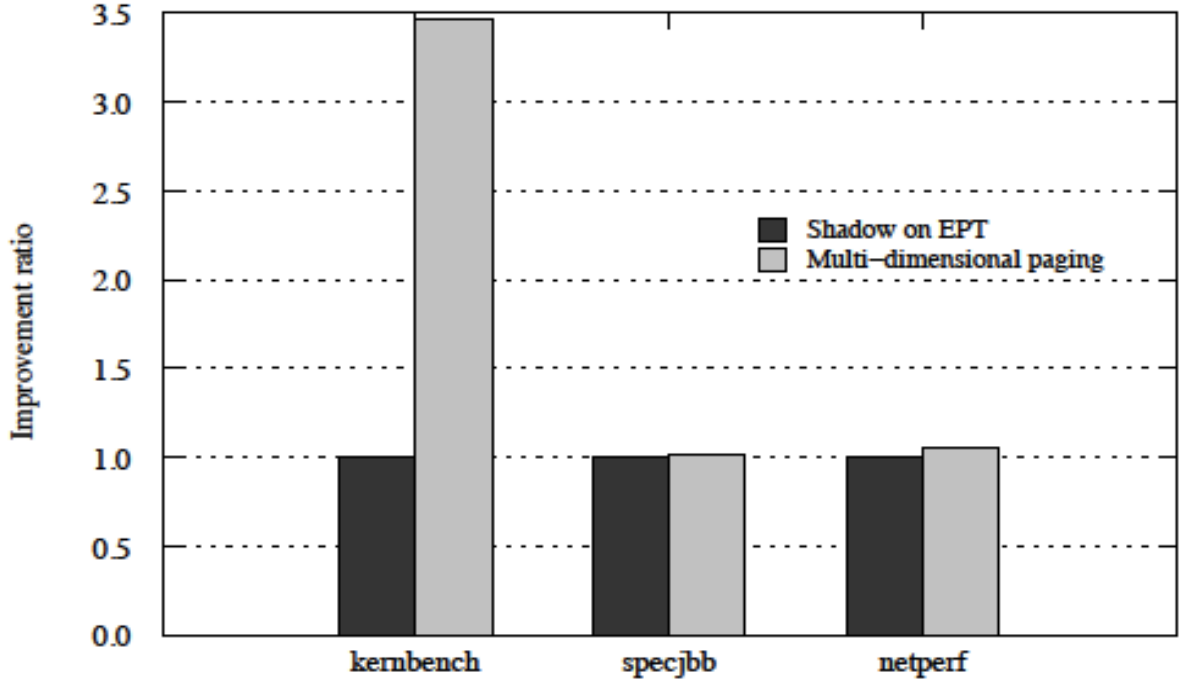

Figure 9: Impact of multi-dimensional paging

The Turtles Project: Evaluation (Running exits only)

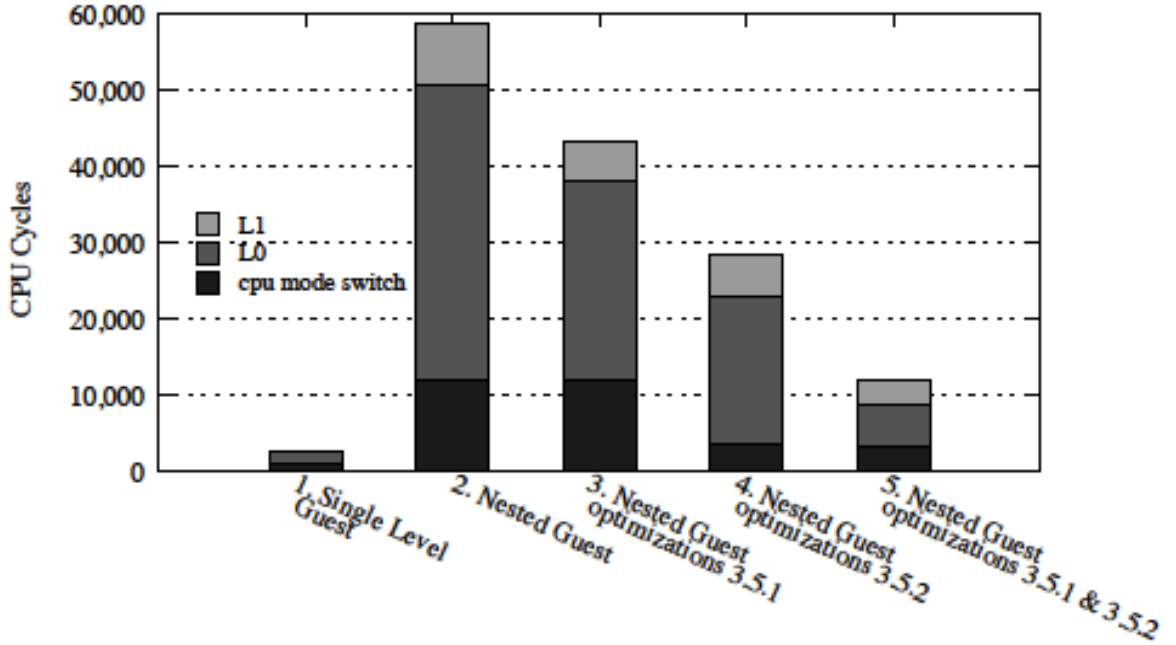

Figure 10: CPU cycle distribution for `cpuid`

The Turtles Project: Cost of a nested exit

1. L_2 executes a `cpuid` instruction
2. CPU traps and switches to root mode L_0
3. L_0 switches state from running L_2 to running L_1
4. CPU switches to guest mode L_1
5. L_1 modifies $\text{VMCS}_{1\to2}$
   repeat n times:
   (a) L_1 accesses $\text{VMCS}_{1\to2}$
   (b) CPU traps and switches to root mode L_0
   (c) L_0 emulates $\text{VMCS}_{1\to2}$ access and resumes L_1
   (d) CPU switches to guest mode L_1
6. L_1 emulates `cpuid` for L_2
7. L_1 executes a resume of L_2
8. CPU traps and switches to root mode L_0
9. L_0 switches state from running L_1 to running L_2
10. CPU switches to guest mode L_2

The Turtles Project: Other performance issues

- Virtualization API improvements
- Traps occurring on a core are handled by it
- Cache pollution when switching from guest-hypervisor

The Turtles Project

Questions?