SplitX: Split Guest/Hypervisor Execution on Multi-Core

Alex Landau[⋆] Muli Ben-Yehuda[†⋆] Abel Gordon[⋆]

[⋆]IBM Research—Haifa

[†]Technion—Israel Institute of Technology

WIOV '11 June 14, 2011 • Portland, OR
3rd Workshop on I/O Virtualization
usenix

IOLanes

Background: machine virtualization

- Running multiple different unmodified operating systems
- Each in an isolated virtual machine
- Simultaneously
- On the x86 architecture
- Live migration, record & replay, testing, security, . . .
- Foundation of IaaS cloud computing
- Used nearly everywhere

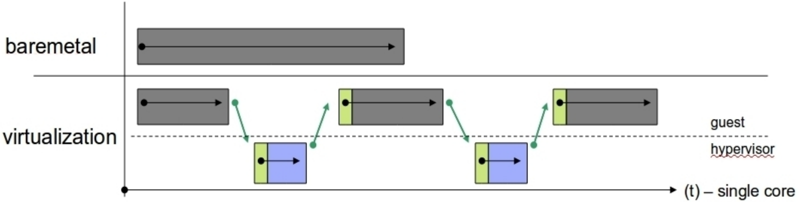
IOLanes

The problem is performance

- Machine virtualization can reduce performance by tens of percents to orders of magnitude [Adams06,Santos08,Ram09,Ben-Yehuda10,Amit11,...]
- Overhead limits use of virtualization in many scenarios
- We would like to make it possible to use virtualization everywhere
- Where does the overhead come from?

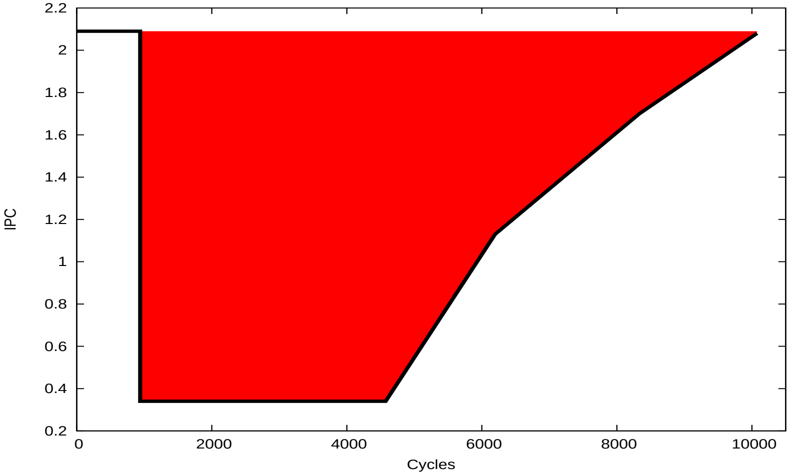
The origin of overhead

- Popek and Goldberg's virtualization model [Popek74]: Trap and emulate
- Privileged instructions trap to the hypervisor
- Hypervisor emulates their behavior
- Traps cause an exit. An exit has:
  - A direct cost for the world switch to the hypervisor and back
  - An indirect cost incurred by the hypervisor and the guest sharing the same core
  - A synchronous cost for handling the exit at the hypervisor
- How bad can it be?



Cost × frequency

Drop in application IPC (red) due to a single null exit at $t = 940$



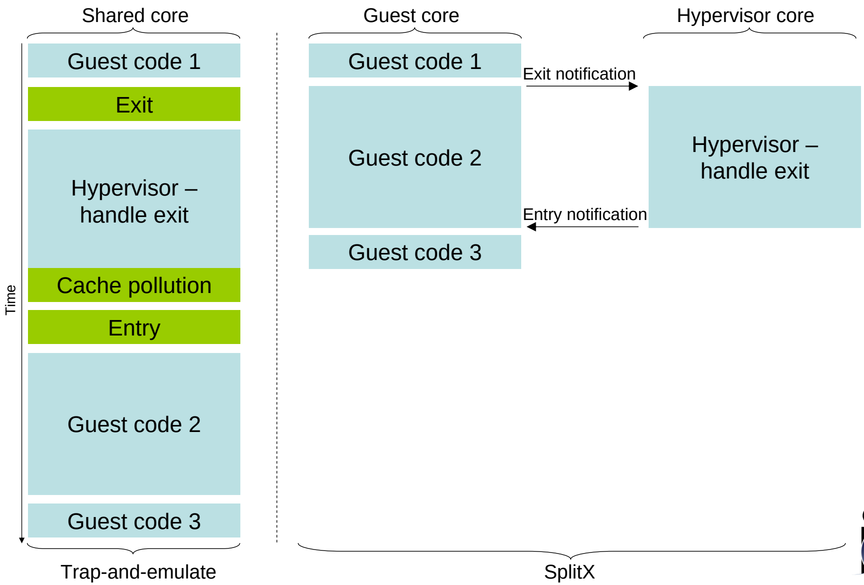
Cost × frequency

Overhead per exit for selected exit types

Exit Type	Number of Exits	Cycle Cost/Exit
External interrupt	8,961,000	363,000
I/O instruction	10,042,000	85,000
APIC access	691,249,000	18,000
EPT violation	645,000	12,000

- netperf client run on 1GbE with para-virtualized NIC
- Total run: ~7.1 $\times$ 10^{10} cycles vs. ~5.2 $\times$ 10^{10} cycles for bare-metal
- 35.73% slow-down due to the guest and hypervisor sharing the same core

SplitX: dedicated cores for guest and hypervisor

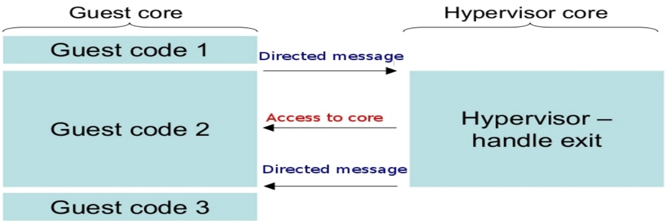


SplitX benefits

- The direct cost is replaced by an inter-core message (2,000 cycles vs. 550 cycles: 3.5x improvement)
- Indirect cost is eliminated completely
- Synchronous cost can be eliminated for some exit types
- Well suited for specialized cores and non-coherent architectures
- Our analysis shows that virtualization with SplitX should reach the holy grail: bare-metal performance with zero overhead!

Architectural support for SplitX

- Cheap directed inter-core signals
  - Extends existing inter-processor-interrupt (IPI) mechanism
  - Guest $\Rightarrow$ hypervisor: guest core sends message indicating an exit, hypervisor core calls software handler
  - Hypervisor $\Rightarrow$ guest: hypervisor sends completion message, guest core handles the message without interrupting the guest
- Manage resources of other cores
  - Hypervisor needs to change the internal state of guest core
  - For example, set `cr0` to a specific value



Architectural support: implementing exits

- Three categories of exits:
  - Non-exits (e.g., HLT) do not need to be handled at all
  - Synchronous exits (e.g., INVLPG): guest is paused until reply received
  - Asynchronous exits (e.g., PIO): guest continues running until a synchronization point is reached
- Interrupt injections and EOIs do not interrupt guest execution
  - Interrupts: hypervisor sends an IPI to the guest
  - EOIs become guest to hypervisor messages like other exits

Category	Exit reasons
Non-exits	HLT, MWAIT, PAUSE
Sync. exits	TASK SWITCH, INVD, INVLPG, CR-WRITE, DR-ACCESS, EPT VIOLATION, INVEPT
Asyncėxits	PIO, WBINVD, CPUID, RDTSC, RDPMC, CR-READ, RDMSR, VM* except VMLAUNCH/VMRESUME

Approximating SplitX on current hardware

- Hardware approximation via hardware exploitation where possible, minimal guest para-virtualization where not
- Guest ⇒ hypervisor: give guest direct access to APIC. Guest can now send IPIs to hypervisor and signal EOIs without exits.
- Hypervisor ⇒ guest: hypervisor sends the guest an NMI; NMI is an exception and does not cause an exit
- Managing guest core resources: hypervisor runs a minimal trusted stub in the guest context to approximate hardware operation

Potential savings

- netperf client run on 1GbE with para-virtualized NIC
- Total run: ~7.1×10^{10} cycles vs. ~5.2×10^{10} cycles for bare-metal: 35.73% slow-down for traditional guest
- Estimated the total cycles a SplitX guest core would consume
  - Sync exits: discounted direct and indirect costs
  - Async exits: also discounted synchronous cost
  - Added 250 cycles per exit: inter-core msgs and data movement
- SplitX guest: ~5.200187×10^{10} cycles vs. ~5.2×10^{10} cycles for bare-metal: difference of 0.0036%

Exit Type	Sync?	# Exits	Cost/Exit	Total	Direct?	Indirect?	Async?	Comm?
External intr.	A.	8961	363	3253726	17922	8961	3253727	2240.25
IO instruction	A	10042	85	848646	20084	10042	848647	2510.5
APIC access	A	691249	18	12469663	1382498	691249	12469663	172812.25
EPT violation	S	645	12	7782	1290	645	0.0	161.25

Table: Savings/overhead per exit type (selected exits) in 1K cycles

IOLanes

Related work

- Offload computation to a dedicated core or set of cores:
  - Sidecore [Kumar07,Gavrilovka09]
  - VPE [Liu09]
  - IsoStack [Shalev10]
  - System call offload [Nellans10,Soares10]
  - vIOMMU [Amit11]
- The Barrelfish [Baumann09a,Baumann09b] multikernel is operating system for non-cache-coherent architectures where each functional unit runs on its own core
- SplitX applies the same core idea of spatial division of cores to machine virtualization for unmodified operating systems

Conclusions

- Exits are the biggest cause of performance loss
- SplitX: a novel approach for eliminating exits by splitting the guest and the hypervisor into different cores
- Needs modest new hardware enhancements; can be approximated on current hardware
- What would happen if virtualization was free from overhead?

Questions?