Hypervisor Verification and Theory of Multicore Systems

W. Paul
Saarland University
Orlando, December 3, 2012
joint work with E. Cohen, S. Schmaltz....
What is a kernel?

- The Classic: Turing machine kernel
- Simulating k one tape Turing machines by 1 one tape Turing machine
  - Tracks: address translation
  - Head position and state: process control block
  - Round robin: scheduling
What is an M-kernel?

- **process virtualization:**
  - simulating k machines of type M by 1 one tape machine of type M
- **+ system calls**
  - for inter process communication...
- **M:**
  - MIPS, ARM, Power, x64...
What is a hypervisor?

- guests can be operating systems, i.e. in system mode
- 2 levels of translation
  - hypervisor page tables
  - guest page tables
  - 'subdivide tracks'
- hardware support
  - nested page tables
- no hardware support:
  - composition of translations is translation
  - maintain 'shadow page tables' (SPT) for combined translation
  - redirect memory management unit (mmu) to SPTs
Background

• 2007-2010: effort to formally verify MS HyperV
  – part of German Verisoft-XT project (Paul, Broy, Podelski, Rybalchenko...), 13 Mio €
  – MS Windows + Research (Cohen, Moskal, Leino,...)

• We failed 2010
  – tool development (VCC) successful
  – crucial portions of code verified
  – tool documentation and soundness argument less than perfect
  – paper and pencil theory incomplete in 2010

• We did not know (exactly enough) what to prove
Hypervisor Correctness is either

- **One** theorem in 1 theory
  - then we are in weapons business of cyber war
- or bug hunting
  - then we (formal verification engineers) are competing with the software community
This talk
(only) 2 years after end of project

• outlines
  – model stack for multicore hypervisor verification
    • I think complete
  – simulation theorems between layers
  – soundness of VCC and its use

• size of remaining gaps:
  • <= PhD thesis
  • I supervised 58 so far
Three kinds of arguments

• abstraction
  – classical commutative diagrams

• order construction
  – in nondeterministic model of concurrent implementation
  – from details of deterministic implementation

• order reduction
  – exclude w.l.o.g. interleavings in concurrent model
7 main theories (1)

• multicore ISA-sp
  – system programmers manual
  – hardware correctness

• serial ISA abstraction
  – to ISA-u (for users)

• serial language stack
  – C + macro assembly + ISA-sp
  – compilers + macroassemblers

• C+ ISA + devices
  – drivers
  – exception handlers
  – boot

• ownership in concurrent computation
  – push through stack
  – serial compiler translates parallel C
7 main theories (2)

• Soundness of VCC and its use
  – C + ghost + assertions
  – VCC proofs imply ownership discipline
  – use of C-verifier for C +ISA + devices

• Hypervisor correctness
  – virtual tread simulation (kernel layer)
  – nested address transation (shadow page tables)
  – ISA-sp virtualization
ISA-sp (1)

- X64
  - Intel: 3000 pages
  - AMD 1500 pages
  - Diss. Degenbaev 300 pages
    - math
- MIPS-86
  - MIPS-ISA+ X86 memory model
  - 15 pages
  - [http://www-wjp.cs.uni-saarland.de/publikationen/SchmaltzMIPS.pdf](http://www-wjp.cs.uni-saarland.de/publikationen/SchmaltzMIPS.pdf)
ISA-sp(2):
X64

• X64 ISA model
  – E. Cohen: nondeterministic communicating sequential components
  – sb: store buffer
  – mmu: memory management unit
  – APIC: device, interrupts
  – disk: for booting

• details subtle
  – better reverse engineer MIPS-86 and prove
ISA-sp (3): MIPS-86  
hardware correctness (formal/paper)

- Processor correctness  
  - pipelined  
  - one memory mode: WB  
  - software conditions:  
    alignment; no self modifying code  
  - digital gate level + gate delays  
  - sequentially consistent shared memory (MOESI)

- April 4, 2012  
  - 283 pages  
  - [link](http://www-wjp.cs.uni-saarland.de/lehre/vorlesung/rechnerarchitektur2/ws1112/layouts/multicorebook.pdf)

TODO  
- fetch and add (easy)  
- fences and sync (easy)  
- consistent memory modes (easy)  
- interrupts + devices (subtle)  
- MMU (subtle)  
- store buffers (easy)  
- Tomasulo scheduler (hard)
ISA-sp to ISA-u (1)

• **Caches** invisible
  – Use cacheable memory modes only
  – compatibility of coherency protocols (MOESI +...)
  – side remark in Smith & Plezkum

(sb: store buffer)
ISA-sp to ISA-u (2)

- **caches** invisible
- **sb** invisible in single core
  - easy folklore theorem
  - proof: Degenbaev et al: Pervasive theory of memory 2009
ISA-sp to ISA-u (3)

- **caches** invisible
- **sb** invisible
- **mmu** invisible
  - set up page table tree
  - linear/translated memory
  - easy folklore theorem
  - proof: Degenbaev et al: Pervasive theory of memory 2009

\[ \text{mem} \]
\[ \text{mmu} \]
\[ \text{core} \]

\( \text{sb: store buffer} \)
ISA-sp to ISA-u (4)

- caches invisible
- sb invisible
- mmu invisible
- ISA-u
language stack (1)
C+macro assembly + assembly+ISA-sp

- C small steps semantics (interleave in parallel C)
- C+ macro assembly realistic and close to VCC
- uses stack abstraction
- process save and restore handles stack pointers
- invisible in C + macroassembly
language stack (2)
combined language semantics

² two languages C + A where A implements C:
² two computations (c<sup>i</sup>) and (a<sup>i</sup>)
² maintain consis(c<sup>i</sup>; a<sup>s(i)</sup>)

² change of C to A: use (a<sup>i</sup>) but track e<sup>®</sup>ect on (c<sup>i</sup>)
² change from A to C: have a<sup>i</sup>:
  1. 9c : consis(c; a): continue with (unique) c
  2. error otherwise
language stack (3) 
compilation

• Optimizing C compiler:

• Optimizing C Compiler + macro assembler + assembler
  – C calls m-asm and vice versa
  – function pointers

– Schmaltz and Shadrin: VSTTE 2012
– Paul et al: SEFM 2012
MIPS ISA-u +devices (1)
formal hardware correctness

- Hardware truely parallel, processor pipelined
- ISA nondeterministic concurrent, 1 step at a time
- construct order of steps
- hardware complex due to a detail in ISA for external interrupts that we used
- 'continue' instead of 'repeat' as in X86
MIPS ISA-u + devices (2)
formal (C+assembly)- driver correctness

- disable and don’t poll interrupts of devices >1
- reorder their device steps out of driver run of dev 1
- pre and post conditions for drivers...

• Diss. Alkassar

• Alkassar et al: TACAS 08
MIPS ISA-u + devices (3)
startup

– Hypervisor:
  • disk: boot loader
  • APIC: wake up other cores
  • Diss Pentchev 2013?

– secure boot:
  • digital signatures
  • Verisoft (2003-2007)
Ownership (1)

concept

• Classify addresses
  1. local (e.g. C stack)
  2. shared and read only (e.g. program)
  3. shared owned (temporarily local/locked)
  4. shared writeable not owned (locks)

• invariants:
  – at most 1 owner ....
  – disjointness...

• safe programs: act like names of address classes suggest

• accesses to class 4 atomic at the language level
Ownership (2)

**Def:** structured parallel C (folklore)

- Classify addresses
  1. local (e.g. C stack)
  2. shared and read only (e.g. program)
  3. shared owned (temporarily local/locked)
  4. shared writeable not owned (locks)

- multiple C threads
- sequentially consistently memory
- shared: heap + global variables
- local: stacks
- safe w.r.t. ownership
  - class 4 access: volatile
Ownership (3)
structured parallel C to parallel assembly

• IF
  – translate threads with sequential compiler
  – translate volatile C access to interlocked ISA-u access

• THEN
  – ISA program safe
  – multicore ISA-u simulates parallel C

• A. Appel, X. Leroy et al: formal work in progress
  – no store buffers

• Dissertation C. Baumann 2012: pushing this through entire language hierarchy on paper
Ownership (4)
parallel store buffer reduction in ISA-sp

- maintain local dirty bits
  - class 4 write since last local sb-flush
- class 4 read only if dirty =0
- Cohen Schirmer ITP 2010: store buffers invisible
  - formal
  - no mmu
- to be pushed through hierarchy
  - implement sb-flush as compiler intrinsic in C
Ownership (5)

semantics from hell

- **Def:** VCC-C:
  - structured parallel C
  - with Cohen Schirmer dirty bits
- VCC-C + m-asm + asm +ISA-sp
Ownership (5) semantics from hell

• VCC-C:
  – structured parallel C
  – with Cohen Schirmer dirty bits

• VCC-C + m-asm + asm +ISA-sp
  – shared shadow page tables
  – MMU (ISA-sp) walks SPTs (volatile C data structure)
  – order reduction: interleave MMU steps at volatile C accesses to SPTs
Model stack

- VCC-C +...+ISA.sp
  - compilation (2-5)
  - ISA-sp
    - hardware correctness (1)
    - digital hardware (1)
    - timing analysis
    - gates+ regs.+drivers + delay
model and theory stack

TODO

• Soundness of VCC and ist use
  – VCC is parallel C verifier

• Theorem: hyperV virtualizes multiple ISA-sp (+ system calls)
VCC (1)
soundness: arguing about ownership

• C + ghost: Dissertation Schmaltz 2012
  – semantics
  – simulation of C by C+ghost
  – ghost code must terminate
  – VCC-C + ghost

• TODO for VCC soundness
  – Semantics of assertion language of C + ghost (logics)
  – show that assertions generated by VCC imply ownership + Cohen Schirmer dirty bit discipline
  – soundness of verification condition generator used for serial and parallel language constructs
VCC (2)
use for C + m-assembly +ISA-sp

• Dissertation Maus (Podelski)
  – hybrid C variables, located in memory outside of regular C variables
  – code non C portions of ISA-sp in hybrid variables
  – write obvious C simulator
  – translate m-assembly macros into C function calls in the naive way

• wildly productive
  – 14K LOC verified

• Maus et al: AMAST 2008

• soundness:
  – Dissertation Shadrin
  – Paul et al: SEFM 12
HyperV correctness (1)
kernel layer: many threads

- Simulation of K C+masm + ISA-sp threads by k physical ISA-sp threads
  - compile C part
  - thread control blocks
  - saving and restoring stack and heap pointers
  - C + masm + asm
  - APICs hard to simulate

- similar to kernel correctness from Verisoft-1 Project (14 Mio €)
  - paper: Gargano et al: TPHOLs 2005
  - formal: Alkassar et al, VSTTE 2010

- Dissertation Alekhin 2013?
HyperV correctness (2)
shadow page tables

• 2 translations
  – guest-OS to user
  – host to guest - OS

• with hardware support
  – nested page tables
  – no formal model and hardware construction yet

• without hardware support
  – composition of translations is translation
  – SPT for composition
  – Redirect MMU to SPTs

• SPT-algorithm without sharing between processors, formal
  – Dissertation Kovalev 2012
  – Alkassar et al FMCAD 2010

• in MS product SPTs with sharing
HyperV correctness(3)
ISA-sp virtualization and system calls

• Virtualization
  – with kernel layer and SPTs similar to Verisoft-1
  – new: state of ISA-sp components of sleeping virtual processors
    – sb empty
    – caches from hardware
    – tlb empty or tagged as in hardware

• Simple Hypervisor
  – formal in VCC
  – without save/restore:
    Alkassar et al: VSTTE 10
    – with: Paul et al: SEFM 12

• system calls and C data structures of kernel as in formal work
  – seL4 (only C portion but can extend with Verisoft-1 technology)
  – or Diss Dörrenbächer 2010 http://www-wjp.cs.uni-saarland.de/publikationen/JD10.pdf
Final remark

• Paul VSTTE 2005
  – a formal proof is an engineering object
  – a paper proof is a building plan

• IFIP working group on verified software 2012
  – lack of such building plans recognized as major obstacle for development of formally verified systems

• almost impossible to publish so far 😁