Efficient Memory Virtualization for Cross-ISA System Mode Emulation

Chao-Jui Chang\textsuperscript{a}, Jan-Jan Wu\textsuperscript{a}, Wei-Chung Hsu\textsuperscript{b}, Pangfeng Liu\textsuperscript{b}, Pen-Chung Yew\textsuperscript{c}

Institute of Information Science, Academia Sinica (a)
Department of Computer Science and Information Engineering, National Taiwan University (b)
Department of Computer Science and Engineering, University of Minnesota (c)
Cross-ISA System Emulation

- **Dynamic Binary Translation (DBT)** enables cross-ISA emulation and allows running different ISA binaries on one machine.
  - Execute ARM binary on x64 machine
- Cross-ISA full system emulation enables developing and testing OS and application on different platforms.
  - Example: Android Emulator on a PC (ARM to x64)
- However, cross-ISA full system emulation is not as efficient as same-ISA system virtualization.
  - All hardware functions are emulated by software
Memory Overhead in Cross-ISA Emulation

- **Memory subsystem emulation** in QEMU system mode takes about 30~40% time during the execution.

![SoftMMU Overhead Chart]

- Chart showing the percentage overhead for various benchmarks:
  - 401.bzip: 42%
  - 403.gcc: 29%
  - 429.mcf: 43%
  - 445.gobmk: 27%
  - 456.hmmer: 31%
  - 458.sjeng: 25%
  - 462.libquantum: 35%
  - 471.omnetpp: 23%
  - 473.astar: 26%
  - 483.xalancbmk: 31%
  - Geo-mean: 31%
Memory Virtualization in Cross-ISA Emulation

**Host Machine**
- Execution flow
- Table reference
- Table pointer

**Host OS**
- DBT Emulator
  - *Run as a host Process*

**Host CPU**
- Host Page Table

**Host Memory**

**DBT Emulator**
- Virtual Machine
  - Guest OS
  - Guest Page Table
- Soft-MMU
  - GVA
- GPA
- gCR3
- Guest Memory
- DBT VM
  - Memory Manager

**Table指针**
- CR3
- HVA
- HPA
- GPA
- gCR3

**图示说明**
1. DBT Emulator
2. GVA
3. Soft-MMU
4. GPA
5. DBT VM
6. Memory Manager
7. Host Page Table
8. HPA
Goal

• Reduce the overhead of memory virtualization in cross-ISA full system emulation.
  – Eliminate multiple steps of memory address translation.
• Design a retargetable memory virtualization solution.
  – Support multiple guest ISAs, e.g. ARM and x86
Outline

• Motivation and Introduction

• Related Work
  – Software-based Shadow Page Table
  – Hardware-assisted Second Level Address Translation

• Efficient Memory Virtualization

• Evaluation

• Conclusion
Shadow Page Table

- **Shadow Page Table (SPT)**\(^1\) is a shadow version of guest page table that maps guest address to host address directly.
- Shadow page table needs to synchronize with guest page table.

1: James Smith, “Virtual Machines: Versatile Platforms for Systems and Processors”  
Figure from “KVM MMU Virtualization”
Page Table Switching Problem in Shadow Page Table for Cross-ISA

VMM

VMEnter

Guest Code Block
... Guest memory access ...
... Guest code ...
... Guest memory access ...
... Guest code

Same-ISA

One Page Table Switching

hPT

sPT
Page Table Switching Problem in Shadow Page Table for Cross-ISA

Shadow page table technology incurs page table switching overhead for cross-ISA emulation.
Second Level Address Translation

- **Second Level Address Translation (SLAT)** technique adds an additional page table level to the architecture and lets hardware page walker perform 2-level address translation.
- SLAT technology is limited to x86 platform.

1: Intel Extended Paging and AMD Nested Paging
Outline

• Motivation and Introduction
• Related Work

• **Efficient Memory Virtualization**
  – Embedded Shadow Page Table
  – Performance Optimization

• Evaluation
• Conclusion
Proposed Key Ideas for Efficient Memory Virtualization

• Leverage host hardware
  – Most CPUs have special hardware for address translation.
  – Use “memory management unit (MMU)” hardware to accelerate memory address translation.

• Access guest memory directly
  – Let MMU hardware be aware of the guest addresses, so that guest code accesses guest memory without memory emulation.
Embedded Shadow Page Table

- Embedded shadow page table avoids page table switching for cross-ISA emulation.

Cross-ISA with SPT

Cross-ISA with *embedded* SPT
Design Issues for Cross-ISA

• Where to embed the shadow page tables without conflicting with existing host address?

• How to reduce overhead of creating embedded shadow page table entry?
  – Not all guest page table entries are populated at begin.

• How to reduce synchronization overhead between shadow and guest page tables?
  – In cross-ISA, checking guest page table modification is by software.
  – Monitoring all guest memory operations results in significant performance penalty.
Issue 1: Virtual Memory Layout

- Modern CPUs support 64-bit address space, e.g. x86_64.
  - Most existing ARM programs are 32-bit.
- Use wider address space to isolate guest VM from emulator.
  - Move DBT emulator above 4GB and leave first 0~4GB for guest.
Steps to Create Embedded Shadow Page Table

(A) Host Page Table

DBT Emulator

Guest OS
GVA: 0x12345000

HVA->HPA
0x7fceaac1d000 -> 0x0effa000

(GVA->HPA) Above 4GB

0x7fceaace1000

0x7fceaac11000

0xFFFFF000

0x12345000

0x00000000

(B) Guest Page Table

GVA->GPA
0x12345000 -> 0xbcde5000

GVA->HVA
0x7fceaac1d000 -> 0x0effa000

(C) DBT Internal Table

GVA->GPA
0x12345000 -> 0xbcde5000

GPA->HVA
0xbcde5000 -> 0x7fceaac1d000

Above 4GB

Below 4GB

Embedded Shadow Page
GVA->HPA
0x12345000 -> 0x0effa000

Page pointer
Translation step

Found

0x7fceaace1000

0x7fceaac11000

0xFFFFF000

0x12345000

0x00000000

Host Physical Memory

HPA: 0x0effa000

(GPA->HVA) 0x0effa000

0x7fceaac1d000

0xbcd5000

0x7fceaace1000
We propose a signal notification mechanism for dynamic creation of embedded shadow page table entries.
We propose an interception-based mechanism for monitoring guest page table modifications.

• **“TLB flush” instruction**
  – A special instruction for flushing a TLB entry.
  – To keep TLB consistency, the OS issues the “TLB flush” instruction after it updates the page table.

• **Intercept guest “TLB flush” instruction**
  – Avoid monitoring all guest memory operations.
Outline

• Motivation and Introduction
• Related Work
• Efficient Memory Virtualization
• Evaluation
• Conclusion
Experiment Setting

• Experiment platform
  – Host Machine: Intel X5550 @ 2.67GHz, 8GB RAM
  – Host OS: Gentoo Linux, 64 bit kernel
  – Guest OS: Busybox image, 32 bit, ARM

• Benchmark
  – SPEC CINT2006

• Comparison
  – Software MMU
  – Embedded Shadow Page Table
SPEC CINT2006: 1.51X Speedup
Number of Memory Instructions Reduced

- Embedded shadow page table avoids multiple steps of address translation in software.
Outline

• Motivation and Introduction
• Related Work
• Efficient Memory Virtualization
• Evaluation

• Conclusion
Conclusion

• We propose an **embedded shadow page table** approach that enables efficient memory virtualization for cross-ISA full system emulation.
  – Use 64-bit address space
  – Leverage existing hardware on modern CPUs
• We devise signal-based and interception-based methodologies to reduce the overhead for creating and maintaining embedded shadow page table.
• Our implementation on QEMU demonstrates 1.51 times speedup on ARM to x64 emulation.
Thank You