Open Source Technology Center

KVM for IA64

Anthony Xu

intel Software

Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright © 2007 Intel Corporation.

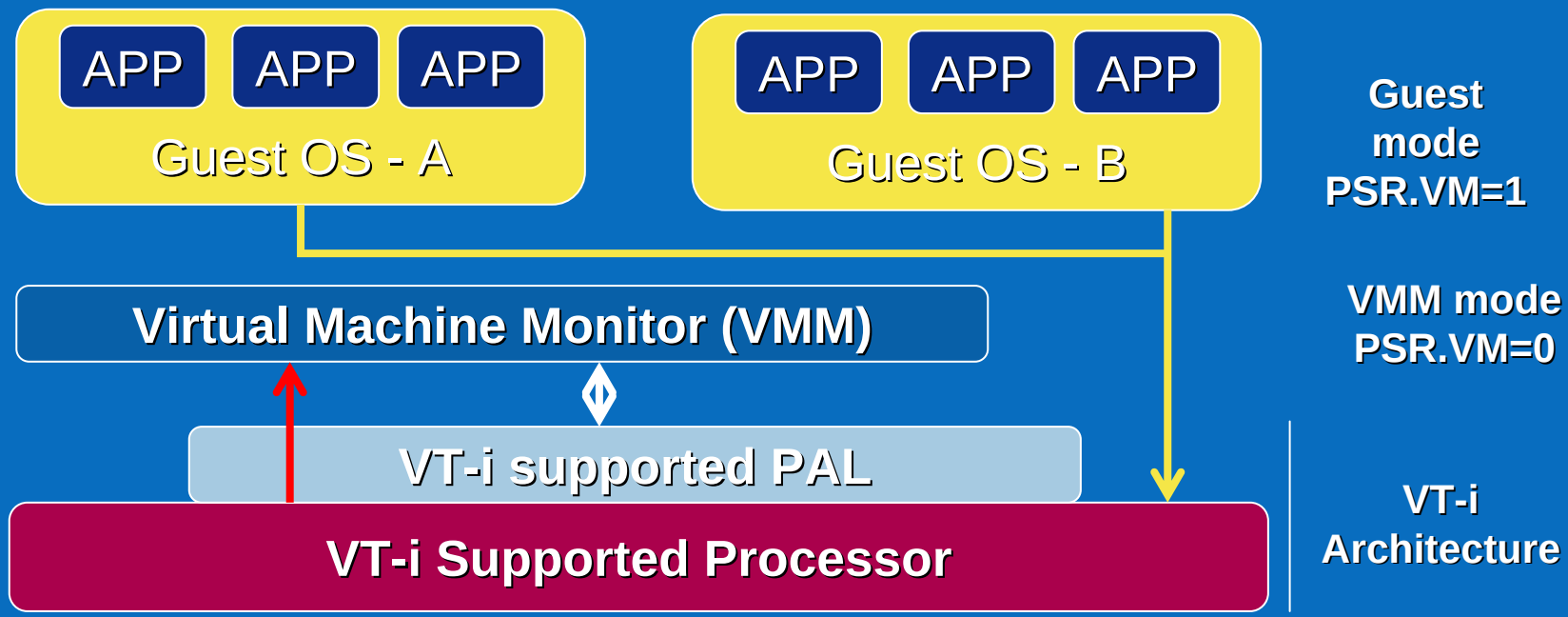
Throughout this presentation:
VT-x *refers to Intel® VT for IA-32 and Intel® 64*
VT-i *refers to the Intel® VT for IA-64, and*
VT-d *refers to Intel® VT for Directed I/O*

Agenda

- IA64 VT-i Architecture
- KVM/IA64 Architecture
- KVM/IA64 Status
- Support needed from host Linux kernel
- Call to action
- Q&A
- Demo

IA64 VT-i architecture

- Overview

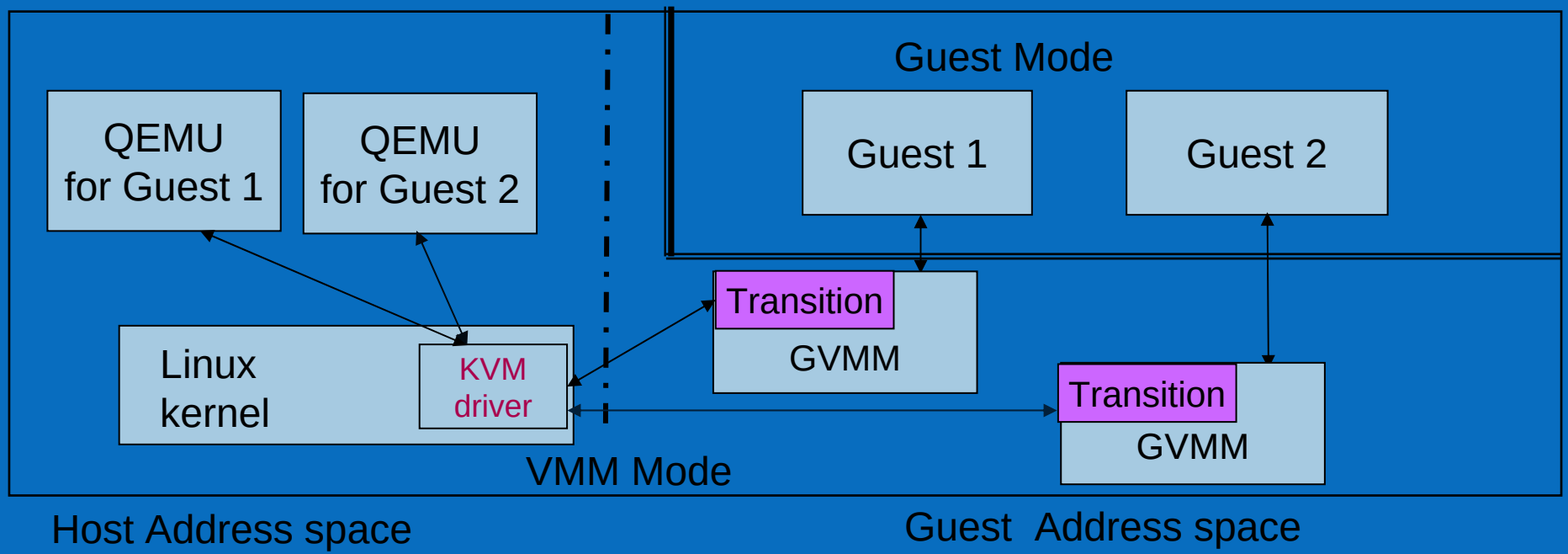


- New VM bit in PSR to indicate operation mode
- New virtualization fault vector to capture Guests executing privileged instruction
- Provides instruction type and opcode upon virtualization fault

IA64 VT-i architecture
- VMM/Guest Virtual Address Separation

- **Virtual Region Number (bits 61-63) always accessible**
- **Guests run with PSR.vm = 1**
  - One less virtual address bit available
- **VMM runs with PSR.vm = 0**
  - All supported virtual address bits are available

KVM / IA64 Architecture



- Host and Guest have separated address space and processor context
- Key components:
  - QEMU: Emulate guest IO devices
  - Guest VMM (GVMM): Handle CPU & memory virtualization
  - KVM driver: Bridges between QEMU and GVMM
- Transition code is responsible for address space and processor context switch

QEMU

- **Re-used existing QEMU device modules for IA64**
  - Added IA64 build target
  - Added IA64 conditional build into existing common code
- **Modified qemu_kvm.c**
  - removed IA32 specific codes
- **Enhanced apic.c to emulate SAPIC**
  - Legacy devices deliver interrupts through SAPIC
  - No PIC on IA64 platform
- **Explicit I/D cache sync upon copying data to guest**
  - Addresses IA64 split I/D L1 cache
- **Doesn't track dirty page in Qemu**
  - VGA refresh all frame buffer

KVM Driver

- **Modified kvm_main.c**
  - removed IA32 specific codes
- **Removed mmu.c**
  - MMU is also arch specific
  - Memory virtualization is implemented inside GVMM
- **Replaced vmx.c with vti.c**
  - Responsible for creating virtualization environment
  - Responsible of preparation tasks for transition
  - Remove CPU virtualization and Memory virtualization logics
  - Handle PAL call and SAL call emulation

GVMM

- **Runs in VMM mode while code is in Guest address space**
- **Implemented as a host module, loaded by host Linux kernel**
  - Mapped to Guest address space by KVM
  - Position Independent Code (PIC)
  - Self-contain and doesn't reference Linux kernel/module symbol
- **Each guest has its own private GVMM data (such as, VCPU, VPD)**
  - This data can be accessed by KVM with different mapping.
- **Optimization: GVMM code can be shared**

GVMM

- Memory Virtualization

- **Software managed TLB**
  - No need for shadow page table
- **Guest TLB virtualization**
  - GVMM uses memory (STLB) to cache Guest TLB.
  - The bigger the STLB, the higher the guest TLB hit.
  - The bigger the STLB, the higher the cost to emulate guest TLB purge all.
- **TLB miss handling**
  - GVMM searches STLB
  - If found, translate from Guest address to machine address through P2M, then insert into TLB.
  - Otherwise inject TLB miss to Guest.

GVMM
- CPU virtualization

Virtual processor description (VPD)

- **Architected data structure**
  - CR, PSR
- **Shared by VMM and PAL**
- **Required per virtual processor**

- **TLB related instructions virtualization**
- **Timer virtualization**
- **Interrupt virtualization**
  - IRR, ISR, EOI
  - Hardware acceleration

GVMM
- Transition Code

- Switch processor context
- Switch virtual address space

KVM/IA64 status

- **Booted UP guest (Linux & Windows)**
  - Using open source GFW
- **Passed internal stress tests.**
- **Reasonable performance**
- **Based on KVM18 release**
  - to be rebased and merged into the main tree.

Proposed Host Linux Enhancement

- **Insert TR**
  - The code segment for inserting TR must be mapped by TR
- **Global TLB purge**
  - Only one processor can do global TLB purge at a time
- **PAL call and SAL call**
  - KVM needs to call host PAL calls or SAL call to emulate guest PAL calls or SAL calls.
- **RID partition**
  - Need purge all TLB entries on transition between host and guest
  - Propose: Add new kernel option rid_bit to reserve RIDs for Guest

Call to Action

- **Merge the code into main tree ASAP**
  - Need community help
- **Community helps to get**
  - PV driver
  - Guest SMP support
  - Save/Restore
  - Live migration
  - Guest NUMA support

Q&A

DEMO


intel®
Leap ahead™

GVMM
- Transition Code

- **Transition Code reside inside GVMM**
- **Host to Guest**
  - KVM switch RR6 to guest RR6
  - KVM insert a TR mapping for GVMM in region 6
  - KVM br.call to predefined transition entry point
  - Switch to guest context
  - br.ret b0 into GVMM
- **Guest to Host**
  - KVM call transitions code
  - Switch to host context(not change RR6)
  - br.ret b0 into KVM
  - KVM switch RR6 to host