Cross-vendor migration: What do you mean my ISA isn't compatible?

Ben Serebrin, CPU Virtualization Architect, AMD

Xen Summit, February 2009

Introduction

Virtualization enables *Live Migration*, the ability to move a running guest from one physical machine to another with little or no disruption to the guest or its users. Live migration allows various load-balancing and high-availability techniques to be implemented by the hypervisor and datacenter management software. Unfortunately for live migration, CPU features are added over time and existing guest OSes are not well-equipped to handle CPU changes while the OSes are running. A sysadmin would prefer to view a homogenous data center where every machine can run every guest, but most real datacenters have several generations of machines that have been added over time. The first efforts of live migration can handle CPU changes within a CPU vendor's product line, but further efforts are required to allow guests to migrate between Intel and AMD CPUs.

This presentation focuses on the Instruction Set Architecture (ISA) issues involved in cross-vendor live migration. Other interesting topics, such as handling the migration of active network connections, the migration of emulated devices, and the migration of directly-assigned devices, are addressed elsewhere and are out of this presentation's scope.

Basics of Live Migration

We first present a very brief overview of live migration. Initially, the guest is running on Host 1, and the management software has decided that the guest should be moved to Host 2. Notionally, the guest should be frozen, and the hypervisor should copy the guest's memory, CPU state, device state, and network connections to Host 2, and then start executing the guest on Host 2.

There are many optimizations possible, including copying the memory while the guest is running and sending incremental memory diffs for dirty pages, to minimize the time the guest is frozen.[1]

Because common existing operating systems do not have an interface for being frozen or for having their hardware changed abruptly (except for plug-and-play devices, which do not include CPUs or fundamental hardware like chipsets), the migration job is significantly complicated if Host 1 is not identical to Host 2. Device emulation or migration-aware device drivers can allow the two hosts to have different models of devices, but there is no performant analogue for handling CPU ISA changes.

CPU ISA Progress and Detection

As x86 CPU generations are developed, CPU vendors add both architectural features (such as SSE4.1) and micro-architectural features (such as cache structure changes). Older ISA features are typically retained in newer CPU generations to support existing software. All of the modern ISA features and some of the microarchitectural features are represented in the CPUID instruction's results. Software (OS, library, and user code) is expected to check the CPUID bit for a given feature set before using that feature set.[2] However, this expectation does not directly comprehend live migration: it is implied that the software need only check the CPUID values one time per program initialization or OS boot. As we will describe below, the hypervisor must accommodate this behavior when implementing cross-vendor migration.

AMD and Intel implement different subsets of the x86 architecture, and at different points in their product lines. History has shown that the vendors typically take up the most useful sets of instructions (for example, Intel has recently added RDTSCP[3], which first arrived in AMD CPUs, and AMD has followed Intel in implementing SSE3 instructions[4]), so the architectures can be expected to converge over time.

In cases where software may take different code paths depending on the set of available instructions, correct behavior will result if the software obeys the results of the CPUID instruction.[*] Software should not test for the presence of new features by any method other than CPUID.

---

[*] There are cases where obeying CPUID is insufficient, most of which pertain to features that were added before the CPUID instruction was created. If we assume a baseline of all CPUs that support SVM or VT-x, then the only current exception is detecting MONITOR/MWAIT support in user mode.[5]

Creating a Baseline ISA
If the sysadmin wishes to create a pool of machines to participate in live migration, and the datacenter contains machines with multiple ISA generations, the sysadmin (or datacenter management software) must set a baseline ISA feature set for all guests that are destined for migration. The management software would then tell the hypervisor what CPUID values to deliver to the guests. For example, if the datacenter contains some machines with SSE2 and some machines with SSE2 and SSE3, the selected baseline should be SSE2 only.

Once a baseline is determined, well-behaved (i.e. CPUID-obeying) guest software will use only the ISA features that are present on all CPUs in the datacenter. As the datacenter adds new machines and retires old machines, the baseline may be raised to include newer ISA feature sets.

CPUID Contents
There are four general types of CPUID fields, and each must be handled in its own way:
Feature bit vectors indicate the ISA subsets. Generally, the logical AND of the bit vectors of all CPUs in the datacenter is the correct baseline value. The XSAVE vectors require slightly more care.
Strings and identifiers contain the vendor name string and processor family, model, and stepping. The hypervisor can present either an artificial name and version ("Xenwashere") or the CPU vendor and identifier of the machine this guest was originally booted on.
Cache sizes contain information about data caches and TLBs. Because software typically uses this information only for tuning, it is difficult to find a situation where software will function incorrectly due to stale cache size information. It would be an interesting investigation to determine what an optimal return value would be. One possibility is to pass through the current CPU's cache data unaltered, to allow user programs that are newly-launched to obtain the most recent cache sizes; another is to provide the cache data from the most common machine in the datacenter; and a third is to have a default minimum cache size that will give reasonable performance on any CPU in the datacenter.
Topology in CPUID leaf 4 provides information on Intel CPUs about the APIC ID allocation and node layout of the system, which the OS may use for NUMA-awareness. If the hypervisor is willing to guarantee similar relative layout of guest nodes, it may be reasonable to provide an artificial view of topology. However, any hypervisor that does not gang-schedule or migrates to different machines may well cause this topology information to be useless.

ISA Differences: SYSCALL vs. SYSENTER

The one significant difference between Intel's and AMD's implementation of the x86-64 ISA is where the vendors implement the SYSCALL/SYSRET and SYSENTER/SYSEXIT instructions. Table 1 shows modes in which each instruction is supported. 32-bit legacy OSes can consistently use the SYSENTER/SYSEXIT pair on both Intel and AMD, and 64-bit OSes can consistently use the SYSCALL/SYSRET pair for 64-bit user applications. However, there is no single instruction pair that is correct to use in 32-bit compatibility mode.

This discrepancy is surprisingly not an issue for 32-bit code in 64-bit Windows guests, which transition into 64-bit mode before performing a SYSCALL.[6]

In Linux, the construction of the vsyscall or vDSO page must choose whether to install SYSCALL, SYSENTER, or INT80 instructions, based on the CPUID indication at the time the guest was booted.[7] [8] The hypervisor has several choices:
Failure: the hypervisor could ignore the problem and hope that there are no 32-bit user programs in a 64-bit guest. In a controlled environment, this may be acceptable because a sysadmin could guarantee the absence of 32-bit programs.

Instruction Supported?, CPUID value	**AMD**			**Intel**		
	Long			Long		
	64	32 compat	32 legacy	64	32 compat	32 legacy
SYSCALL, SYSRET	Y, 1	Y, 1	Y, 1	Y, 1	N, 0	N, 0
SYSENTER, SYSEXIT	N, 1	N, 1	Y, 1	Y, 1	Y, 1	Y, 1

Table 1: System call instructions supported in each mode. Notably, Intel returns 0 in the SYSCALL CPUID bit (CPUID(0x8000_0001).EAX[11]) in 32-bit modes and returns 1 in 64-bit mode. AMD always returns 1 in the SYSENTER CPUID bit (CPUID(0x1).EAX[11]).

Emulation: the hypervisor could provide the CPUID indication of either AMD or Intel, depending on which is more prevalent in the datacenter, and could trap the #UD exception to emulate the unimplemented instructions on the less-prevalent CPU.
In-place edit: simply changing the SYSCALL instruction to SYSENTER or vice versa is insufficient because the instructions have slightly different behaviors. The guest could generate the two flavors of vDSO pages and inform the hypervisor of where those pages are, allowing the hypervisor to change virtual or guest physical address mappings during a migration.
Force to INT80: the hypervisor could hide both SYSCALL and SYSENTER. This could have unnecessary performance impacts on 64-bit code, however. Ideally, the hypervisor could force only 32-bit vsyscall pages to INT80, but the existing Linux detection code does not accommodate this. It would be a straightforward paravirtualization to make the 32-bit compatibility page use either INT80 or in-place editing.[9]
Wait for hardware: Depending on performance data and software complexity, AMD or Intel could consider implementing the missing SYSENTER or SYSCALL modes. Hardware changes are considerably slower than software changes, so a software solution is useful even if hardware changes do come later.

Virtualization Instructions

The only other major ISA divergence is that of the virtualization instructions themselves. Each CPU vendor implements only one set of virtualization instructions. While interesting, the subject of a guest executing virtualization instructions is out of scope here.

The hypervisor's use of virtualization instruction requires a few code changes to translate between AMD's VMCB and Intel's VMCS.[10]

Handling Errata

It is possible that CPU errata require special behavior from the operating system or hypervisor. In some cases, MSR bits must be set: the hypervisor can handle such workarounds. In more intrusive workarounds, special OS behavior is required and may require the OS to detect processor versions via CPUID. In these hopefully rare cases, the workaround detection must be part of the baseline if possible, or the CPU must be excluded from the migration pool if there are no other fallback configurations. There are currently no such situations known.

Does the future hold ISA hotplug?

It is unfortunate that ISA features must be sacrificed for migration compatibility. Future work could include enabling OSes, libraries, and user applications to understand addition and removal of ISA subsets, to extract maximum performance while retaining live migration capability.

We provide a sketch of a potential architecture for ISA hotplug: The guest OS could register a callback with the hypervisor for notification of ISA addition or removal, and the user programs could similarly register callbacks with their OSes. Various methods could be used to avoid notification storms, such as intercepting undefined opcode exceptions and providing notifications lazily. It might be necessary for hypervisors to emulate instructions to allow a code sequence to finish before the code path can be reconfigured to support the newly plugged ISA.

The testing matrix of software that can support such on-the-fly changes is significant; if any software supports ISA hotplug, it may reasonable to limit the scope to core OS and user math libraries.

Acknowledgements

Many thanks go to Travis Betak, Andre Przywara, Uwe Dannowski, Christoph Egger, Alex Fit-Florea, and several of our partners for their technical work.

References

[1] http://kvm.qumranet.com/kvmwiki/KvmForum2007?action=AttachFile&do=get&target=Kvm_Live_Migration_Forum_2007.pdf
[2] AMD Programmer's Manual, Volume 1, Section 3.6.1.
[3] Intel Software Developer's Manual, Volume 2B.
[4] AMD Programmer's Manual, Volume 4.
[5] Intel Software Developer's Manual, Volume 3A, Section 7.11.3.
[6] https://kerneltrap.org/mailarchive/linux-kvm/2008/11/13/4079334
[7] http://lwn.net/Articles/30258/
[8] http://lxr.linux.no/linux+v2.6.28.1/arch/x86/vdso/vdso32-setup.c#L283
[9] Alexander Graf, personal communication
[10] http://www.mail-archive.com/kvm@vger.kernel.org/msg09410.html