Cross-Vendor Migration

AMD Operating Systems Research Center

Uwe Dannowski   Andre Przywara

{Firstname.Lastname}@amd.com

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 operating systems (OSes) and applications are not well-equipped to handle CPU changes while the OSes are running. A system administrator would prefer to view a homogenous data center in which 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 among CPUs of different vendors.

This white paper focuses on the instruction set architecture (ISA) issues involved in cross-vendor live migration between Intel® and AMD CPUs. 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 paper'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 possible optimizations, 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 OSes 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 such as 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 application 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 arrived first in AMD CPUs, and AMD has followed Intel in fully implementing SSE3 instructions[4]), so the architectures can be expected to converge over time.

If 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.

Creating a Baseline ISA

If the system administrator wishes to create a pool of machines to participate in live migration, and the datacenter contains machines with processors from multiple vendors or with multiple ISA generations, the system administrator (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 rise to include newer ISA feature sets.

CPUID

The values the hypervisor delivers to guests through CPUID control the feature set (well-behaved) guests may use. Furthermore, CPUID also conveys information about the processor or the platform that may guide software in performance optimization.

CPUID leaf 0h reports the vendor name string and the maximum CPUID leaf supported. CPUID leaves beyond leaf 0h are vendor-specific. As such, software must infer interpretation rules for the remaining leaves from the vendor name string. To date, no conflicting allocations of CPUID leaves exist.

Strings and Identifiers

Because the vendor name string in CPUID leaf 0h determines the meaning of the remaining leaves, it is recommended that the hypervisor present an existing vendor name. The hypervisor

---

[*] There are cases in which obeying CPUID is insufficient, most pertaining 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. MONITOR and MWAIT have sufficient virtualization control hooks that handling this special case is a simple matter for hypervisor software.

may present the CPU vendor string of the machine this guest was originally booted on, or that of the majority of machines in the migration pool, or one that best matches the guest's requirements.

The processor family, model, and stepping presented in CPUID leaf 01h are of less relevance, because most software appears to be well-behaved and checks CPUID feature bits to determine availability of ISA features instead of inferring them from processor family and model. However, the availability of some resources (e.g., certain model-specific registers (MSRs) such as performance counters) is not governed by CPUID but by vendor name and family, model, and stepping. Furthermore, software sometimes assumes the existence of certain CPUID leaves depending on those values. Therefore, the choice of vendor name and family, model, and stepping also influences which virtual resources the hypervisor needs to provide (emulate).

Presenting an artificial name such as "Xenwashere" may be tempting, but cannot be recommended. Linux applies the fewest vendor-specific quirks when confronted with an unknown vendor name, whereas some 64-bit variants of Windows® refuse to boot on an unknown processor. Software may also get confused as to how to interpret the remaining CPUID leaves for a processor with an unexpected vendor name.

Feature Bit Vectors

Feature bit vectors indicate the ISA subsets available to software. Although feature bit definitions are vendor-specific, bit vectors that reside in the same CPUID fields have no conflicting bit assignments to date. A feature bit set to 1 indicates availability of the feature; feature bits marked as "Reserved" (such as a bit assigned to an ISA extension exclusively implemented by another vendor) tend to read as 0. Therefore, the logical AND of corresponding feature bit vectors of all CPUs in the datacenter is generally the correct baseline value.

Cache Size Information

Various CPUID leaves contain information about data caches and TLBs. Because software typically uses this information only for tuning, it is difficult to find a situation in which 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; a third is to have a default minimum cache size that will give reasonable performance on any CPU in the datacenter.

Topology Information

CPUID leaves 04h and 0bh provide information on Intel CPUs about the APIC ID allocation and node layout of the system, which the OS may use for scheduling decisions. 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, if not counterproductive.

ISA Differences

This section discusses selected ISA differences between recent AMD and Intel processors.

Fast System Calls in Compatibility Mode

The one significant difference between Intel's and AMD's implementations of the x86-64 ISA is how 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 – the mode used to execute 32-bit applications on a 64-bit OS.

Instruction supported?, CPUID value	AMD			Intel		
	Long			Long		
	64	32 compat	32 legacy	64	32 compat	32 legacy
SYSCALL, SYSRET	Yes, 1	Yes, 1	Yes, 1	Yes, 1	No, 0	No, 0
SYSENTER, SYSEXIT	No, 1	No, 1	Yes, 1	Yes, 1	Yes, 1	Yes, 1

Table 1: System call instructions supported in each processor operating 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]).

This discrepancy is not an issue for 32-bit code in 64-bit Windows guests, which transition into 64-bit mode before performing a SYSCALL.[5]

In Linux, the construction code 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.[6,7] The hypervisor has several choices:

- Failure: The hypervisor could ignore the problem and hope there are no 32-bit user programs in a 64-bit guest. In a controlled environment, this may be acceptable because a system administrator could guarantee the absence of 32-bit programs.

- 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.

- Emulation: The hypervisor could provide the CPUID indication of either AMD or Intel, depending on which is more prevalent in the datacenter, and could intercept the undefined opcode (#UD) exception to emulate the unimplemented instructions on the less prevalent CPU.

We implemented emulation of the instructions SYSENTER, SYSEXIT, and SYSCALL in the Xen and KVM hypervisors. Table 2 shows the cost of emulation as measured with a microbenchmark running as a 32-bit application in a 64-bit Linux virtual machine (lower number equates to lower cost).

Native SYSENTER on Intel	312
Native SYSCALL on AMD	190
Emulated SYSENTER+SYSEXIT on AMD	9900
Emulated SYSCALL on Intel [t8]	10,728
int80 on Intel	709
int80 on AMD	360

Table 2: Execution time (in cycles) of the `getpid()` Linux system call in a KVM guest using different methods for kernel entry. "On AMD" refers to execution on a 2.2 GHz AMD Opteron™ 2354 processor; "on Intel" refers to execution on a 2.4 GHz Intel Core™2 Quad Q6600 processor. Numbers reported are observed minima.

Emulation on hardware that does not support the instructions in the respective mode adds about 10,000 cycles per system call invocation. Listing 1 shows pseudo-code for emulating the instructions SYSENTER and SYSCALL in compatibility mode and SYSEXIT in 64-bit mode. SYSCALL's companion instruction SYSRET does not need emulation because it is available in 64-bit mode and allows returning to compatibility mode.

---

[t] At the time of writing, the code path in KVM for handling #UD intercepts on Intel processors had yet to be optimized.

```
EMULATE_SYSENTER
{
  IF (LOCK_PREFIX)
    INJECT #UD;

  IF (CR0.PE = 0)
    INJECT #GP;

  IF ((MSR_SYSENTER_CS & 0xFFFC) = 0)
    INJECT #GP;

  SETUP_SEGMENTS();

  EFLAGS.VM = 0;
  EFLAGS.IF := 0;
  EFLAGS.RF := 0;

  CS.SEL := MSR_SYSENTER_CS;
  SS.SEL := CS.SEL + 8;

  IF (LONG_MODE)
    CS.DB := 0;
    CS.L := 1;

  CPL := 0;
  RIP := MSR_SYSENTER_EIP;
  RSP := MSR_SYSENTER_ESP;
}
```

```
EMULATE_SYSEXIT
{
  IF (LOCK_PREFIX)
    INJECT #UD;

  IF ((CR0.PE = 0) OR (CPL != 0))
    INJECT #GP;

  IF ((MSR_SYSENTER_CS & 0xFFFC) = 0)
    INJECT #GP;

  SETUP_SEGMENTS();

  IF (REX.W)
    CS.SEL := MSR_SYSENTER_CS + 32;
    CS.DB := 0;
    CS.L := 1;
  ELSE
    CS.SEL := MSR_SYSENTER_CS + 16;
  CS.DPL := 3;

  SS.SEL := CS.SEL + 8;
  SS.SEL.RPL := 3;

  CPL := 3;
  RIP := RDX;
  RSP := RCX;
}
```

```
SETUP_SEGMENTS
{
  CS.L := 0;
  CS.BASE := 0;
  CS.G := 1;
  CS.LIMIT := 0xffffffff;
  CS.TYPE := 0x0b;
  CS.S := 1;
  CS.DPL := 0;
  CS.PRESENT := 1;
  CS.DB := 1;

  SS.UNUSABLE := 0;
  SS.BASE := 0;
  SS.LIMIT := 0xffffffff;
  SS.G := 1;
  SS.S := 1;
  SS.TYPE := 0x03;
  SS.DB := 1;
  SS.DPL := 0;
  SS.PRESENT := 1;
}
```

```
EMULATE_SYSCALL
{
  IF ((MSR_EFER.SCE = 0) OR (LOCK_PREFIX))
    INJECT #UD;

  SETUP_SEGMENTS();

  CS.SEL := MSR_STAR.SYSCALL_CS & 0xFFFC;
  SS.SEL := MSR_STAR.SYSCALL_CS + 8;
  EFLAGS.RF := 0;

  IF (LONG_MODE)
    RCX := next_RIP;
    IF (64BIT_MODE)
      RIP := MSR_LSTAR;
    ELSE
      RIP := MSR_CSTAR;
    R11 := EFLAGS;
    EFLAGS := EFLAGS & ~MSR_SYSCALL_MASK;
    CS.DB := 0;
    CS.L := 1;
  ELSE
    EFLAGS.VM = 0;
    EFLAGS.IF := 0;
    ECX := next_EIP;
    EIP := MSR_STAR.EIP;
}
```

Listing 1: Pseudo code for the emulation of the instructions SYSENTER and SYSEXIT on AMD processors, and SYSCALL on Intel processors.

In addition, the MSRs holding the target EIP and ESP for SYSENTER must be emulated on AMD processors. These MSRs have been extended to 64 bits on Intel processors, whereas they are implemented as 32-bit registers on AMD processors. While the storage for these MSRs in the guest state region of the VMCB is 64 bits wide, only the lower 32 bits are actually saved and restored. The hypervisor must therefore maintain the 64-bit wide values in separate storage.

Prefetch Instructions

With the 3DNow!™ instruction set extension, AMD also added two instructions to prefetch data into the cache, PREFETCH and PREFETCHW. Initially, their availability was governed by the 3DNow CPUID feature bit. From AMD's eighth generation-RevG processors on, these prefetch instructions have their own 3DNowPrefetch CPUID feature bit. However, these instructions are

also available when Long Mode is available.[9] Software written for Long Mode may therefore assume general availability of the PREFETCH and PREFETCHW instructions. This assumption holds for all AMD processors.

Intel processors do not implement these instructions. However, with family 15/model 6/ stepping 1, Intel processors started treating the instructions as a NOP, whereas earlier processors raise a #UD.[10] If we again assume a baseline of all CPUs supporting SVM or VT-x and Long Mode, then only two Intel Pentium 4 models (662 and 672) and four Intel Xeon models (70xx) seem to be affected; these chips were introduced in late 2005 and early 2006 respectively.

PUSH seg

AMD and Intel processors differ in how they modify memory when pushing segment register onto the stack in legacy and compatibility mode. AMD processors zero-extend the 16-bit segment selector and write the whole word to the stack. Intel processors only update the least significant 16 bits of the word on the stack, leaving the other bytes in memory intact. The result is a potentially different stack memory image, notably only for the bytes that would be ignored by corresponding POP instructions and any algorithms that inspect the 16-bit value on the stack. Software that requires the resulting memory images to match (for example for deterministic replay) may fail. However, such software would not be portable across a mix of AMD and Intel processors in the first place and is therefore outside the scope of this paper.

Model-Specific Registers

As the name "model-specific register" suggests, software is allowed to make assumptions about availability of MSRs based on the CPU vendor, family, and model. Evaluation of those values may happen once at guest boot time, with the result stored for later reference, or dynamically at any later time. It is therefore important to report a consistent CPU vendor, family, and model on all machines in the virtualization pool.

Generally, for guests to execute correctly, a hypervisor must provide access to the MSRs the guest OS expects based on the CPU vendor, family, and model it sees. Access by the guest OS either can be allowed (via the MSR permission bitmap) to MSRs that are managed by virtualization hardware or the hypervisor, or it must be emulated by the hypervisor. While some MSRs need elaborate handling, such as changing virtual processor modes or returning sensible values on read, others only require minimal emulation, such as discarding writes and returning zero on read. Most MSRs that are considered architectural are managed by virtualization hardware (i.e., their guest state and host state are swapped during world switches).

Allowing a guest to access directly MSRs that are not managed by the hypervisor can result in lost state after migration or unavailability of the MSR on the destination host, causing guest or host failure.

APIC TPR Optimization

32-bit Windows XP, Windows 2000, and 32-bit Windows Server 2003 before SP2 frequently access the APIC's Task Priority Register (TPR). APIC registers are mapped into the memory address space of the (virtual) processor and therefore need rather expensive pagefault-style hypervisor intervention to be virtualized.

To help virtualization, AMD added the CR8 register as an alternative way of manipulating the TPR, in long mode as well as in 32-bit compatibility and legacy mode. However, there has been no adoption of the CR8 method by 32-bit OSes. Intel added the FlexPriority feature that provides a 4-kB region in the guest physical address space with special semantics for APIC registers such as the TPR. No compatible feature is available on AMD processors today.

The AMD-V™ Optimization Driver[11] paravirtualizes TPR accesses for the OSes in question by replacing memory-mapped TPR accesses with accesses to CR8 and does not require support from the hypervisor. Unfortunately, only AMD processors support the encoding LOCK MOV CR0 to access CR8 in legacy mode. Therefore, a guest with the AMD-V Optimization Driver loaded would fail when migrated to an Intel processor.

KVM has the ability to modify the running guest's code that accesses the TPR to redirect to a hypervisor-provided code fragment that, if necessary, uses a hypercall to involve the hypervisor. A guest modified in such a way can also run on an Intel processor. Using this mode on AMD processors and FlexPriority on Intel processors in the migration pool yields the most flexible solution.

Floating-Point Accuracy

Minor differences in FPU implementation details between vendors and between processor families of the same vendor may yield results differing in the unit-in-the-last-place (ULP) for the same computation performed on different machines. Listing 2 illustrates this by example of the FSIN instruction.

```
// 80-bit extended precision variables
arg:
        .extended 0.436814692820413274284
result:
        .extended 0.0

test:
        FLD    (arg)         // Load arg into ST0
        FSIN                 // Compute sin(arg) in ST0
        FST    (result)      // Store ST0 into result

// result now contains
//   AMD:    c5b63c408eb79ad8fd3f = 4.23055397142996827752e-01
//   Intel:  c4b63c408eb79ad8fd3f = 4.23055397142996827725e-01
```

Listing 2: For certain input values, the result produced by the FSIN instruction on an AMD Opteron™ processor differs from that on an Intel Core i7 processor in the last binary position of the significand (c5 vs. c4).

An application may produce different results in two otherwise identical runs when migration to a machine with a different FPU implementation happened to occur between the runs. Generally, however, software should already have appropriate measures in place to deal with accumulating errors of that sort. The instructions that are susceptible to this ULP variance are specified with an error tolerance that informs software of the limitations of those instructions; any software that properly accounts for this variance should be tolerant of a change in the effective error induced by migration. Any change in effective error due to migration is still within the defined error

bounds of the instruction. Note that only x87 floating point computations and SSE's rarely used square-root approximations are affected; the remainder of the SSE computations exhibits no differences on any implementation. Furthermore, these effects can be observed only in the FPU's native 80-bit extended precision format; conversion to 64-bit (double) or 32-bit (float) format will mask the differences through rounding. Thus, the possibility for accumulating errors depends on whether the compiler (or the programmer) decided to store intermediate results in lower-precision variables or keep them in the FPU.

Virtualization Instructions

The only other major ISA divergence is that of the virtualization instructions and their accompanying data structures as they are used by the hypervisor itself. Each CPU vendor implements only one set of virtualization instructions. While interesting, the subject of a guest executing virtualization instructions is out of this paper's scope.

The hypervisor's use of hardware-assisted virtualization features requires guest-state translation on migration between AMD's VMCB and Intel's VMCS. While the format of most fields holding guest state is identical and seems to translate 1:1, processors apply different levels of scrutiny in validating this guest state.

- Both AMD and Intel processors store limits in segment descriptor caches as an expanded value (in contrast to a 20-bit limit and the G bit in the segment descriptor), but Intel processors require a consistent G bit in the guest CS segment attribute field, whereas AMD processors ignore this bit.

- AMD processors ignore the Accessed bit, whereas Intel processors require it to be set.

- Intel processors maintain an Unusable bit for NULL segments that does not exist on AMD processors, and therefore – like the G bit – has to be generated on migration from AMD to Intel processors. [12]

Listing 3 illustrates regeneration of those bits.

```
// Regenerate G-bit for CS
IF (SEG = CS)
    IF (SEG.LIMIT > 0xFFFFF)
        SEG.G := 1
    ELSE
        SEG.G := 0

// Generate Unusable-bit
SEG.UNUSABLE := !SEG.PRESENT || (SEG.TYPE = 0);

// Set Accessed bit
IF (!SEG.UNUSABLE)
    SEG.ACCESSED := 1
```

Listing 3: Generate segment descriptor cache bits to pass guest-state consistency checks on migration from AMD to Intel processors.

9

Guest Physical Address Size

Processors may differ in the size of physical addresses they support (e.g., 48 bits vs. 40 bits). A processor may also support a size of guest physical addresses that is different from that of its (host) physical addresses. AMD processors supporting nested paging may indicate the maximum supported size of guest physical addresses via CPUID leaf 080000008h, EAX[23:16]. If the field is zero, the physical address size in CPUID leaf 080000008h, EAX[7:0] also indicates the guest physical address size.

Migration of a guest using a large physical address size to a processor supporting only a smaller guest physical address size might fail due to guest physical addresses becoming invalid. It is therefore important to publish the minimum supported guest physical address size of all processors in the migration pool to guests as their supported physical address size via the CPUID mechanism and consider this limit when laying out guest physical memory.

Handling Errata

It is possible that CPU errata require special behavior from the OS 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.

Cross-Vendor Migration in Action

After implementing support for dealing with segment descriptor cache incompatibilities and the emulation of fast system call instructions (as described earlier in this document), we successfully migrated a large number of configurations from Table 3. However, not all points in the space Table 3 spans were tested.

Host CPUs	AMD Opteron™ 2210, Intel Core2 Duo E6300
	AMD Opteron™ 2354, Intel Core2 Quad Q6600
Hypervisors	KVM-85, Linux 2.6.31-rc
	Xen 3.4, Linux 2.6.18.8 Dom0
Guest OS	Linux 32-bit/64-bit (SLES10, SLES11, Slackware, Ubuntu)
	Windows XP 32-bit/64-bit
	Windows Server 2008 32/64-bit
	Windows Vista 32-bit/64-bit
	Windows 7 RC 32-bit/64-bit
	memtest86
Workloads	Idle after logon
	Busy loop
	Passmark BurnInTest™
	AMD Windows System Stability Test (internal)
	32-bit system call test application
Miscellaneous	1 and more virtual processors
	Shadow paging on Intel
	Nested paging and shadow paging on AMD

Table 3: Configurations used for experiments in this study.

Guests were booted on either side, initially migrated manually, and then subjected to automated migration that moved the guest between the pair of machines every 30 seconds for at least 24 hours.

We believe the remaining problems we observed during migration are unrelated to the cross-vendor aspect of migration, and are more general stability problems with live migration in the hypervisors we tested.

Hypervisor Configuration

As mentioned earlier, it is important to provide a virtual CPU that is consistent across the virtualization pool. This section briefly illustrates how the KVM and Xen hypervisors can be configured accordingly.

By default, KVM presents the vendor string of the host processor while presenting the feature bits of an artificial QEMU-specific CPU. However, when invoked with the following command line, the virtual CPU identifies itself as an Intel processor, irrespective of the host processor, using the vendor override option. The `qemu64` configuration provides a processor with 64-bit extensions, NX, and SSE3. Family, model, and stepping indicate a Pentium 4 processor.

```
qemu -cpu qemu64,vendor=GenuineIntel,family=15,model=6,stepping=1,+cx16
```

The following fragment from a Xen example VM configuration file illustrates the configuration of a migration-friendly virtual CPU.

```
cpuid = [ '0:eax=0x3,ebx=0x0,ecx=0x0,edx=0x0',
          '1:eax=0x0f61,
            ecx=xxxxxxxx0xx00xxxxxxxxx0xxxxxxxxx,
            edx=xxx0xxxxxxxxxxxxxxxxxxxxxxxxxxx',
          '0x80000000:eax=0x80000004,ebx=0x0,ecx=0x0,edx=0x0',
          '0x80000001:eax=0x0f61,
            ecx=xxxxxxxxxxxxxxxxxx0000000000000x,
            edx=00xx000xx0xxx0xxxxxxxxxxxxxxxxxx']
# up to leaf 0000_0003/8000_0004
# family 15 model 6 stepping 1 (Intel P4 Prescott, AMD K8)
# disable POPCNT, SSE4.[12], SSSE3
# disable HTT
# disable CMPLEGACY, SVM, EXTAPIC, ALTMOVCR8, ABM, SSE4a, MisAlignSSE,
#         3DNOWPrefetch, OSVW, IBS, SSE5, SKINIT, WDT
# disable 3DNOW, 3DNOWEXT, RDTSCP, Page1GB, FFXSR, MMXExt, MP
```

Conclusion

Cross-vendor migration gives customers the freedom to equip their virtualization pool with machines that best match their requirements, such as speed, power consumption, and price. Software that strictly respects CPUID before using a feature can be expected to function in a virtualization pool with machines of different vendors. It is the responsibility of the virtualization pool management software to find an ISA baseline.

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 unplugged ISA.

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

---

[1] http://www.linux-kvm.org/wiki/images/5/5a/KvmForum2007%24Kvm_Live_Migration_Forum_2007.pdf
[2] AMD64 Architecture Programmer's Manual, Volume 1, Section 3.6.1.
[3] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.
[4] AMD64 Architecture Programmer's Manual, Volume 4.
[5] http://kerneltrap.org/mailarchive/linux-kvm/2008/11/13/4079334
[6] http://lwn.net/Articles/30258/
[7] http://lxr.linux.no/linux+v2.6.28.1/arch/x86/vdso/vdso32-setup.c#L283
[8] http://lxr.linux.no/linux+v2.6.30.4/arch/x86/kvm/x86.c#L2370
[9] AMD64 Architecture Programmer's Manual, Volume 3
[10] http://communities.vmware.com/message/826840
[11] http://sourceforge.net/projects/amdvopt/
[12] http://www.mail-archive.com/kvm@vger.kernel.org/msg09455.html