Live Migration with AMD-V™ Extended Migration Technology

Publication #	**43781** Revision: **3.00**
Issue Date:	**April 2008**

Advanced Micro Devices 

Contents

List of Figures

Revision History

Date	Revision	Description
4/2008	3.00	Initial public release.

Chapter 1 Live Migration with AMD-V™ Extended Machine Migration

This document describes virtual machine migration.

1.1 Introduction

Virtual Machine (VM) migration is a capability increasingly used in today's enterprise environments. With live migration, a Virtual Machine Monitor (VMM) moves a running VM nearly instantaneously from one server to another and seamlessly to the end user while maintaining guest uptime guarantees. However, if the processors running in the computers in the pre- and post-migration environment are not identical, live migration can result in unexpected behavior of the guest software. Since the introduction of AMD64 processor technology in 2003, AMD has worked closely with virtualization software developers to define the functionality necessary to ensure that live migration is possible across a broad range of AMD64 processors. This document discusses the basics of live migration and describes the functionality currently available from AMD to enable virtualization software vendors to support live migration across a broad range of AMD64 processors. Figure 1 on page 8 illustrates live virtual machine migration on AMD processors.

1.2 System Virtualization

System virtualization is the ability to abstract and pool resources on a physical platform. This abstraction decouples software from hardware and enables multiple operating system images to run concurrently on a single physical platform without interfering with each other.

System virtualization has existed for decades on mainframes. In the past, industry-standard x86-based machines with their limited computing resources and challenges in correctly virtualizing the Instruction Set Architecture (ISA)[1,2] did not lend themselves well to the technique. However, with the increase in computing resources of x86-based servers in recent years and with new processor extensions such as those found in AMD-V™[3], x86-based virtualization is seeing widespread adoption and is rapidly becoming a pervasive and integral capability in enterprise environments[4].

Virtualization can increase utilization of computing resources by consolidating the workloads running on many physical systems into virtual machines running on a single physical system. Virtual machines can be provisioned on-demand, replicated, and migrated.

VM migration, which is the ability to move a virtual machine from one physical server to another under VMM control, is a capability increasingly utilized in today's enterprise environments.

This document discusses how AMD-V™ Extended Migration Technology enables virtual machine migration between AMD64 processors.

1.3 Virtual Machine Migration

A virtual machine can be migrated in the following ways:

- **Static migration.** The VM is shut down using OS-supported methods; its static VM image is moved to another VMM and restarted.
- **Cold migration.** The VM is suspended using OS-supported or VMM-supported methods. The suspended VM image is moved to a VMM on a different machine and resumed.
- **Live migration.** The VMM moves a running VM instance nearly instantaneously from one server to another. Live Migration allows for dynamic load balancing of virtualized resource pools, hardware maintenance without downtime, and dynamic failover support.

In the following sections, software running in the VM is called "guest software."

As long as the hardware in the pre- and post-migration environment is identical, guest software should behave in exactly the same way before and after the migration. Challenges can arise, however, when guest software runs in a different hardware environment after a migration.

Note that even though a VMM presents a virtual platform to guest software, there can be certain interfaces, depending on VMM design, which guest software can directly use to determine underlying hardware's capabilities, discussed further in Section 1.6 on page 10.

After the reboot/restart following a static migration, guest software should go through its platform discovery phase and be able to adjust to any differences in underlying (virtual) hardware.

Following a cold migration, guest software may continue to maintain an identical view of the pre- and post-migration hardware. When suspended using OS-supported methods, some operating systems re-scan the hardware upon resume. Depending on their policies and the hardware differences between current and previous hardware, the OSes may refuse to resume and require a reboot.

After a live migration, guest software continues to maintain an identical view of the pre- and post-migration hardware. This document discusses how AMD64 processors provide support for a VMM to hide differences in software-visible processor features during live and cold migration. For simplicity, "live migration" is used to refer to both migration methods.

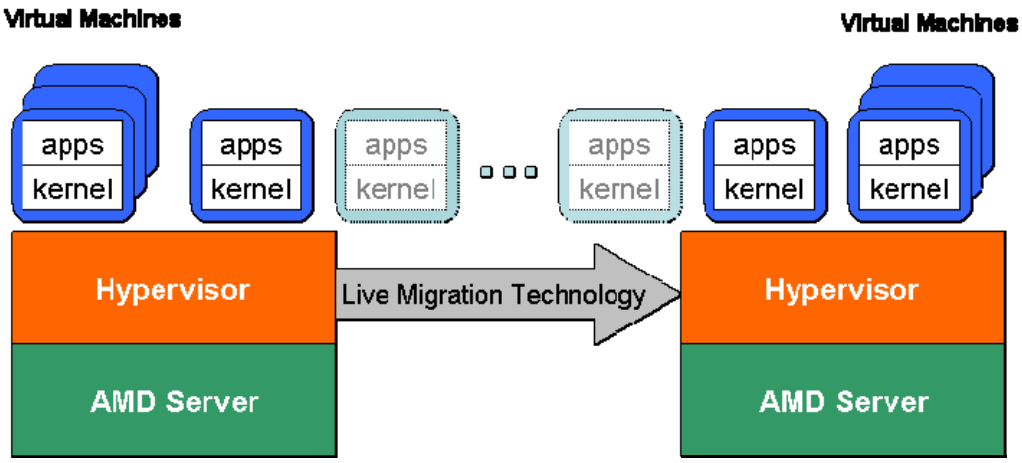

Figure 1. Live Virtual Machine Migration on AMD Processors

1.4 Determining Processor Features

Software should use CPUID to determine processor features and use this information to select appropriate code paths. CPUID is an x86 instruction that can be executed from kernel and applications. The following example shows CPUID called from an application.

```
//Example 1: Console application on Windows XP using
CPUID instruction to determine //processor revision and
feature information.

#include "stdafx.h"
#include <stdio.h>

int _tmain (int argc, _TCHAR* argv[])
{

     int version;
     int features_1;
     int features_2;

     __asm
     {
          mov eax, 1

          cpuid

          mov version, eax
          mov features_1, ecx
          mov features_2, edx
     }

     printf ("Processor version: 0x%x features_1: 0x%08x
     features_2: 0x%08x\n", version,
     features_1, features_2);

     return 0;
}
```

However, a small number of existing commercial software applications determines the presence of certain processor features using non-standard methods (such as using a try-catch code block to determine whether certain instructions are supported) or assumes the presence of certain features without testing for them. This software can cause indeterminate behavior under live migration scenarios. Note that such software is equally likely to fail in non-virtualized environments with different processors.

1.5 VMM Basics

To understand the implications of VMM design on VM migration, it is useful to briefly review various VMM design alternatives.

It is important to note that for performance reasons, a VMM may only interfere in the execution of guest software when the guest code performs privileged operations [1,2].

The first method is called **para-virtualization**, in which the guest's kernel is modified to cooperate with the VMM when performing privileged operations. The second method is called **binary translation**, in which the VMM modifies the guest's binary image at runtime to get processor control when guest software attempts to perform a privileged operation [2]. The VMM can then emulate the privileged operation and return control to guest software. The third method is **hardware-assisted virtualization**, in which the VMM uses processor extensions such as AMD-V™ to intercept and emulate privileged operations in the guest.

1.6 Problems with Live Migration

In the following sections, the processor in pre-migration environment is called the "current" processor. The processor in post-migration environment is called the "target" processor.

When performing a live migration, a VM is suspended; its current (guest and VMM) state is captured, moved to another VMM, and resumed on it. The VMM preserves the state and identity of virtual devices after the migration. This process happens quickly and seamlessly to the end user and maintains guest uptime guarantees.

As indicated previously, when hardware in the pre- and post-migration environment is different, live migration can result in unexpected guest behavior. After a live migration, there is no indication to guest software that it is now running on a different server with potentially different processor features. This fundamentally different programming environment may not interact well with existing software design. For example, a guest application may use CPUID instruction at startup to determine that the processor supports SSE3 instructions, then live migrate to another server and consequently fail when attempting to use those instructions on a target processor that does not support SSE3.

To help ensure that guest software does not malfunction after a live migration, the virtualization infrastructure should address the problems shown in the following subsections.

1.6.1 Backward-Compatibility Problem

The backward-compatibility problem happens when the target processor does not implement a feature that is implemented on the current processor. In the case of this problem, after live migration, use of such a feature may result in unexpected software behavior. For example, use of an instruction not implemented on the target processor results in a #UD fault.

1.6.2 Forward-Compatibility Problem

The forward-compatibility problem happens when the target processor implements features not supported on the current processor. This situation could be problematic in cases where the target processor uses an opcode (to implement an instruction) that is not defined on the current processor, and software on the current processor expects use of this undefined opcode to generate a #UD fault. In such cases, after a live migration, software observes a different processor behavior, that is, it does not observe #UD fault upon use of the (previously undefined) opcode.

Fortunately, a very small number of existing commercial software depends on such processor behavior. AMD actively works with its software partners to discourage use of such behaviors.

1.7 AMD-V™ Extended Migration Technology

Since the introduction of AMD64 processor technology in 2003, AMD has worked closely with virtualization software developers to define the functionality necessary so that live migration is possible across a broad range of products. AMD-V™ Extended Migration Technology provides various capabilities to address problems associated with Live Migration.

1.7.1 Backward-Compatibility Assurance

To assure backward compatibility, the VMM can intercept the #UD fault and emulate the faulting instruction. However, this method of emulating the faulting instruction is not generally acceptable because it adds to the complexity of the VMM and may also degrade the performance of the guest.

A simpler and more widely adopted solution by many VMM vendors is to expose to the guest software only those features that are supported on all processors in the resource pool. This solution essentially means establishing the lowest common denominator of processor features in the resource pool.

1.7.1.1 Controlling Information Returned through CPUID

When new features are introduced on AMD processors, their presence is indicated by unique bits in the information returned by the CPUID instruction. Properly written software relies only on this information to determine processor features.

To ensure proper behavior during VM migration, a VMM must control information returned to guest software as a result of the CPUID instruction.

A VMM using hardware-assisted virtualization (AMD-V™) (such as Xen HVM or Microsoft® Virtual Server) can use the CPUID intercept to return the appropriate bits whenever guest software executes that instruction.

A VMM using binary translation (such as VMware ESX) or para-virtualization (such as Xen) may need to handle this differently depending on whether CPUID was executed by guest's kernel or guest's application. In the case of CPUID instruction executed by the guest's kernel, the VMM can binary-translate or para-virtualize that instance of the instruction and return appropriate information to the guest.

For a CPUID instruction executed within the guest application's context (where binary translation or para-virtualization techniques are not feasible), the VMM requires a way to return the appropriate CPUID information to that guest application. Note that with binary translation and para-virtualization, it is the user-mode CPUID features that the VMM must control. Without this ability to control the user-mode CPUID features, all processors in the resource pool must support the same features, which significantly reduces flexibility in utilizing resources in environments supporting Live Migration.

The AMD CPUID Override Model Specific Registers (MSRs), as described in Section 1.7.1.2, provide the VMM the ability to control CPUID information returned to guest applications.

1.7.1.2 CPUID Override Model Specific Registers

Starting with AMD Family 0Fh revision C AMD Opteron™ processors, AMD provides CPUID Override MSRs to allow a VMM (or OS) to specify a subset of information that the CPUID instruction returns. This is accomplished by setting the appropriate MSRs as described below.

MSR Type	***CPUID Features Register Override***
MSR Description	Provides control over features reported in CPUID function 0000_0001 in ECX and EDX.
MSR Index	C001_1004h
Bits	**Description**
63:32	Features. R/W, function 1, ECX
31:0	Features. R/W, function 1, EDX

MSR Type	***Extended CPUID Features Register Override***
MSR Description	Provides control over features reported in CPUID function 8000_0001 in ECX and EDX.
MSR Index	C001_1005h
Bits	**Description**
63:32	Features. R/W, function 1, ECX
31:0	Features. R/W, function 1, EDX

MSR Type	***Logical Processor Count (K8 only)***
MSR Description	MSR is provided for backward-compatibility with K8 processors.
MSR Index	C001_100Dh
Bits	**Description**
63:32	Reserved
31:0	Logical Processor Count; R/W

MSR Type	***Processor Name String Registers***
MSR Description	These registers hold the CPUID name string in ASCII. The state of these registers is returned by CPUID instruction Functions 8000_0004, 8000_0003, 8000_0002. Each register contains a block of 8 ASCII characters; the least significant byte corresponds to the first ASCII character of the block; the most-significant byte corresponds to the last character of the block. MSR C001_0030h provides the first block of the name string; MSR C001_0035h contains the last block of the name string.
MSR Index	C001_0030h – C001_0035h
Bits	**Description**
63:0	CpuNameString. R/W.

When the VMM or OS does not override CPUID return information using this mechanism, the processor returns CPUID information based on the native feature set.

The following code sample shows how a VMM (or OS) can hide reporting of SSE4A instructions on a quad-core AMD Opteron™ processor.

```
/*
 *  Example 2: Use MSR C001_1005 to clear bit 6
(SSE4A) reported in ECX after CPUID
 *  Function 8000_0001
*/


     /*
     Read current value of the CPUID Override MSR
     C001_1005.
      After RDMSR completes, EDX:EAX contains the
     64bit MSR value. EDX is loaded with the high 32
     bits of the MSR and EAX is loaded with the low 32
     bits. The high 32 bits of this MSR are returned
     in ECX after CPUID Function 8000_0001
     */

          MOV CX, 0xC0011005h
          RDMSR

     /*
     Clear bit 6 (SSE4A) of EDX register
     */

          ANDL EDX, 0xFFFFFFBFh

     /*
     Write the new EDX:EAX value into CPUID override
     MSR
     */

          WRMSR
```

The following code sample shows how a VMM (or OS) can hide reporting of RDTSCP instruction on AMD Opteron™ processor, rev. F.

```
/*
 *  Example 3: Use MSR C001_1005 to clear bit 27
(RDTSCP) reported in EDX after
 * CPUID Function 8000_0001
*/

     /*
     Read current value of the CPUID Override MSR
     C001_1005.
      After RDMSR completes, EDX:EAX contains the
     64bit MSR value. EDX is loaded with the high 32
     bits of the MSR and EAX is loaded with the low 32
     bits. The low 32 bits of this MSR are returned in
     EDX after CPUID Function 8000_0001
     */
     /*
     Write the new EDX:EAX value into CPUID override
     MSR. AMD Opteron™ Rev-F Processors require a 32
     bit password in EDI. Contact AMD to get the
     password.
     */

     MOV EDI, <PASSWORD>

          MOV CX, 0xC0011005h
          RDMSR

     /*
     Clear bit 27 (RDTSCP) of EAX  register
     */

          ANDL EAX, 0xF7FFFFFh

          WRMSR
```

Note that overriding CPUID return information using these MSRs does not disable the respective features of the processor. On AMD Opteron™ processors, a password must be passed through the EDI register before executing RDMSR and WRMSR to set the CPUID overrides. This password can be obtained from AMD.

Figure 2. shows Live Migration support in current AMD processors:

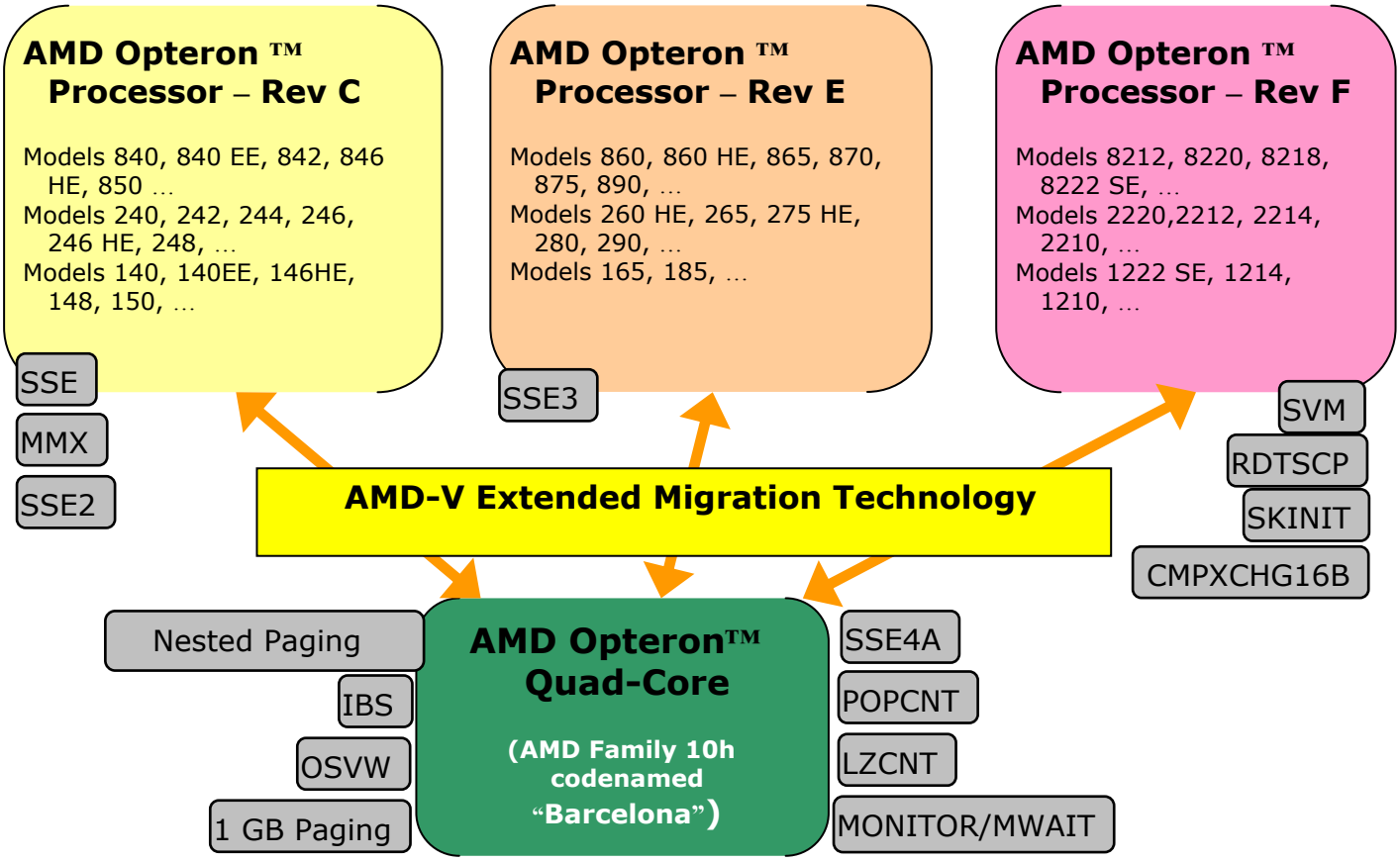

Figure 2. Live Migration Support in Current AMD Processors

If guest software does not use CPUID to determine processor features and, instead, relies on deprecated methods (such as using certain code sequences to detect implementation differences and using the result to identify processor features), then these overrides may be of little use. Fortunately, AMD field surveys indicate that such software is very limited in today's enterprise environments.

1.7.2 Forward-Compatibility Assurance

In Section 1.6.2 on page 11, we discussed forward-compatibility problems. To avoid these problems, the VMM needs a method to disable new instructions until all the processors in the resource pool implement them. Disabling new instructions is required to ensure that software continues to observe the same processor behavior (such as expected #UD faults) in forward-migration scenarios.

1.7.3 Feature-Disable Bits

On future processors, AMD may support feature-disable bits to disable certain instructions or instruction sets. Such feature-disable bits could be exposed through an MSR-based interface.

AMD believes that the long-term solution for forward-compatibility assurance is encouraging developers to write well-behaved, "live-migration friendly" code and adding appropriate support in VMMs to handle forward-compatibility problems.

1.8 Conclusions

In this document, we have described in detail the potential problems that can cause guest software to malfunction after a Live Migration. We also discussed how AMD64 processors are designed to provide the functionality needed to support the VMM in addressing these problems.

AMD-V Extended Migration Technology allows a VMM to safely migrate a VM between various AMD64 processor revisions. AMD-V provides IT administrators an unprecedented flexibility in deploying, maintaining, and upgrading servers in Live Migration environments based on AMD64 processors.

Readers are cautioned about instances of legacy software that rely on non-standard methods to determine processor features. However, such instances are limited and should diminish over time.

AMD encourages IT administrators to work with their VMM software vendors in determining the level of VMM's support for AMD-V™ Extended Migration Technology. The respective vendors can provide guidance when configuring Live Migration resource pools with AMD processors[5].

1.9 References

1. Popek, G. J.; Goldberg, R. P. "*Formal Requirements for Virtualizable Third Generation Architectures*," *Communications of the ACM;* 17(7): 412 –421.
2. Adams, K.; Agesen, O. "*Comparison of Software and Hardware Techniques for x86 Virtualization*." ASPLOS.
3. AMD. *AMD64 Architecture Programmer's Manual*, Vol. 2.
4. IDC. "Impact of Virtualization Software on Operating Environments." *IDC Technology Assessment.*
5. VMware. "VMotion CPU Compatibility Requirements for AMD Processors."