Performance Evaluation of AMD RVI Hardware Assist

VMware ESX 3.5

Introduction

For the majority of common workloads, performance in a virtualized environment is close to that in a native environment. Virtualization does create some overheads, however. These come from the virtualization of the CPU, the MMU (Memory Management Unit), and the I/O devices. In some of their recent x86 processors AMD and Intel have begun to provide hardware extensions to help bridge this performance gap. In 2006, both vendors introduced their first-generation hardware support for x86 virtualization with AMD-Virtualization™ (AMD-V™) and Intel® VT-x technologies. Recently AMD introduced its second generation of hardware support that incorporates MMU virtualization, called Rapid Virtualization Indexing (RVI).

We evaluated RVI performance by comparing it to the performance of our software-only shadow page table technique on an RVI-enabled AMD system. From our studies we conclude that RVI-enabled systems can improve performance compared to using shadow paging for MMU virtualization. RVI provides performance gains of up to 42% for MMU-intensive benchmarks and up to 500% for MMU-intensive microbenchmarks. We have also observed that although RVI increases memory access latencies for a few workloads, this cost can be reduced by effectively using large pages in the guest and the hypervisor.

Background

Prior to the introduction of hardware support for virtualization, the VMware® virtual machine monitor (VMM) used software techniques for virtualizing x86 processors. We used binary translation (BT) for instruction set virtualization, shadow paging for MMU virtualization, and device emulation for device virtualization.

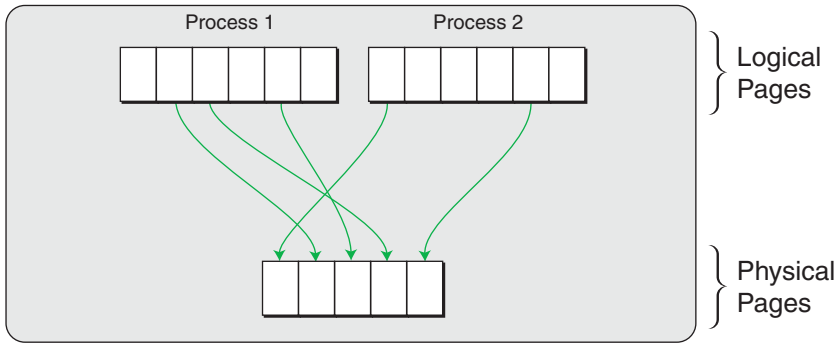
With the advent of AMD-V in 2006 the VMM could use BT or AMD-V for instruction set virtualization on AMD processors, depending on which technology had better performance for a specific guest/CPU configuration. Due to the lack of hardware support for MMU virtualization in older CPUs, the VMM still used shadow paging for MMU virtualization. The shadow page tables stored information about the physical location of guest memory. Under shadow paging, in order to provide transparent MMU virtualization the VMM intercepted guest page table updates to keep the shadow page tables coherent with the guest page tables. This caused some overhead in the virtual execution of those applications for which the guest had to frequently update its page table structures.

With the introduction of RVI, the VMM can now rely on hardware to eliminate the need for shadow page tables. This removes much of the overhead otherwise incurred to keep the shadow page tables up-to-date. We describe these various paging methods in more detail in the next section and describe our experimental methodologies, benchmarks, and results in subsequent sections. Finally, we conclude by providing a summary of our performance experience with RVI.

MMU Architecture and Performance

In a native system the operating system maintains a mapping of logical page numbers (LPNs) to physical page numbers (PPNs) in page table structures (see Figure 1). When a logical address is accessed, the hardware walks these page tables to determine the corresponding physical address. For faster memory access the x86 hardware caches the most recently used LPN->PPN mappings in its translation lookaside buffer (TLB).

Figure 1. Native System Memory Management Unit Diagram

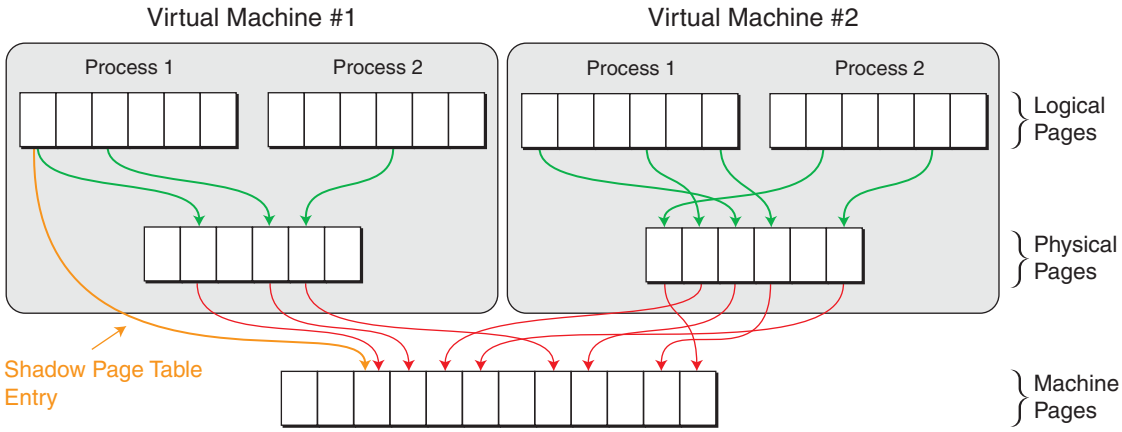


In a virtualized system the guest operating system maintains page tables just as the operating system in a native system does, but in addition the VMM maintains a mapping of PPNs to machine page numbers (MPNs), as described in the following two sections, "Software MMU" and "Hardware MMU."

Software MMU

In shadow paging the VMM maintains PPN->MPN mappings in its internal data structures and stores LPN->MPN mappings in shadow page tables that are exposed to the hardware (see Figure 2). The most recently used LPN->MPN translations are cached in the hardware TLB. The VMM keeps these shadow page tables synchronized to the guest page tables. This synchronization introduces virtualization overhead when the guest updates its page tables.

Figure 2. Shadow Page Tables Diagram

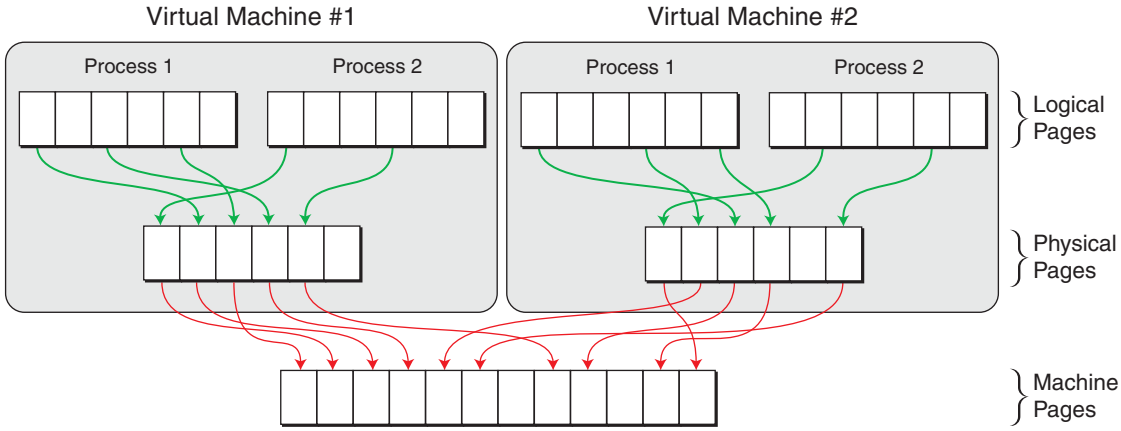


Hardware MMU

Using RVI, the guest operating system continues to maintain LPN->PPN mappings in the guest page tables, but the VMM maintains PPN->MPN mappings in an additional level of page tables, called nested page tables (see Figure 3). In this case both the guest page tables and the nested page tables are exposed to the hardware.

When a logical address is accessed, the hardware walks the guest page tables as in the case of native execution, but for every PPN accessed during the guest page table walk, the hardware also walks the nested page tables to determine the corresponding MPN. This composite translation eliminates the need to maintain shadow page tables and synchronize them with the guest page tables. However the extra operation also increases the cost of a page walk, thereby impacting the performance of applications that stress the TLB. This cost can be reduced by using large pages, thus reducing the stress on the TLB for applications with good spatial locality. For optimal performance the ESX VMM and VMkernel aggressively try to use large pages for their own memory when RVI is used.

Figure 3. Hardware Memory Management Unit Diagram



Experimental Methodology

This section describes the experimental configuration and the benchmarks used in this study.

Hardware

Dell PowerEdge 2970

CPUs: Two Quad-Core AMD Opteron™ 8384 Processors (“Shanghai”)

RAM: 32GB

Networking: Two Broadcom BCM5708C NetXtreme II GbE controllers

Storage controller: Dell PowerEdge PERC 5/i SAS RAID controller

Disks: Six 7200 RPM 160GB Seagate Barracuda SAS hard drives (7200.7 ST3160023AS)

Virtualization Software

These experiments were performed with VMware ESX 3.5, Update 2. The tests in this study show performance differences between the BT VMM and the RVI VMM as a way of comparing shadow paging with RVI. We compared the RVI VMM to the BT VMM instead of to an AMD-V VMM without RVI because ESX 3.5 supports AMD-V only with RVI.

Benchmarks

In this study we used various benchmarks that to varying degrees stress MMU-related components in both software and hardware. These include:

- Kernel Microbenchmarks: A benchmark suite for system software performance analysis.
- Apache Compile: compiling and building an Apache web server.
- SPECjbb®2005: An industry standard server-side Java benchmark.
- SQL Server Database Hammer: A database workload that uses Microsoft SQL Server on the backend.
- Citrix XenApp: A workload that exports client sessions along with configured applications.

We ran these benchmarks in 64-bit virtual machines with a combination of Windows and Linux guest operating systems. Table 1 details the guest operating system used for each of these benchmarks.

Table 1. Guest Operating Systems Used for Benchmarks

Benchmark	Operating System
Kernel Microbenchmarks	64-bit Red Hat Enterprise Linux 5, Update 1
Apache Compile	64-bit Red Hat Enterprise Linux 5, Update 1
SPECjbb2005	64-bit Windows Server 2008
SQL Server Database Hammer	64-bit Windows Server 2008
Citrix XenApp	64-bit Windows Server 2003

It is important to understand the performance implications of RVI as we scale up the number of virtual processors in a virtual machine. We therefore ran some of the benchmarks in multiprocessor virtual machines.

Experiments

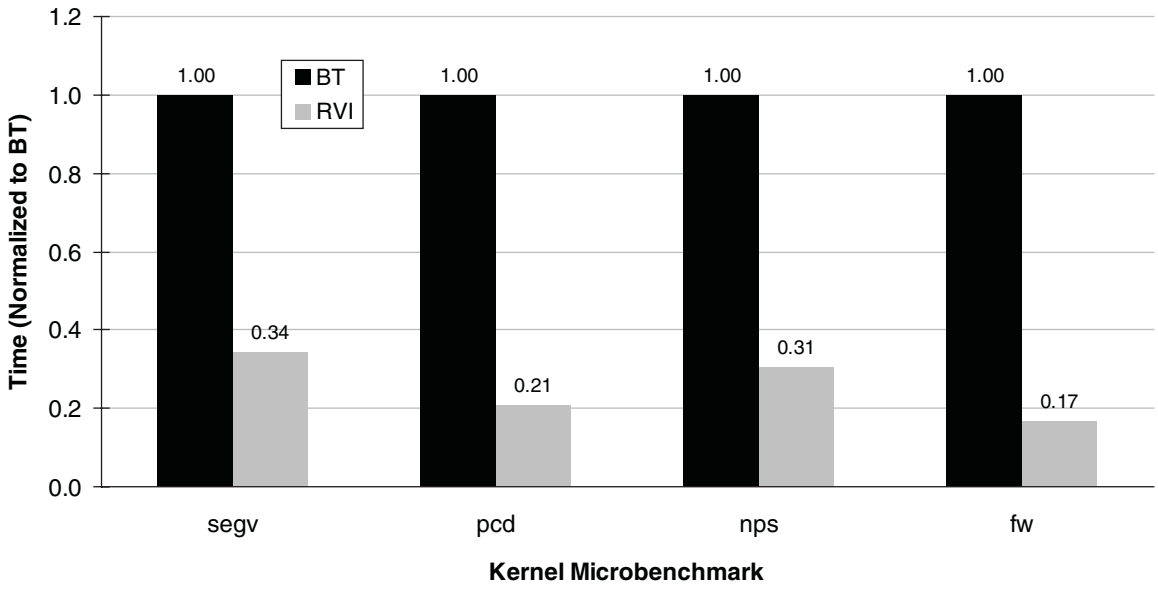
In this section, we describe the performance of RVI as compared with shadow paging for the workloads mentioned in the previous section.

MMU-Intensive Kernel Microbenchmarks

Kernel microbenchmarks comprise a suite of benchmarks that stress different subsystems of the operating system. These microbenchmarks are not representative of application performance; however they are useful for amplifying the performance impact of different subsystems so that they can be more easily studied.They can be broadly divided into system-call intensive benchmarks and MMU-intensive benchmarks. In our experiments we found that system-call intensive benchmarks performed equivalently with and without RVI enabled (for brevity these results are not included in this paper). However, with RVI enabled we observed gains of up to 500% on MMU-intensive microbenchmarks. Figure 4 shows the results for the following kernel microbenchmarks:

- segv: A C program that measures the processor fault-handling overhead in Linux by repeatedly generating and handling page faults.
- pcd: A C program that measures Linux process creation and destruction time under a pre-defined number of executing processes on the system.
- nps: A C program that measures the Linux process-switching overhead.
- fw: A C program that measures Linux process-creation overhead by forking processes and waiting for them to complete.

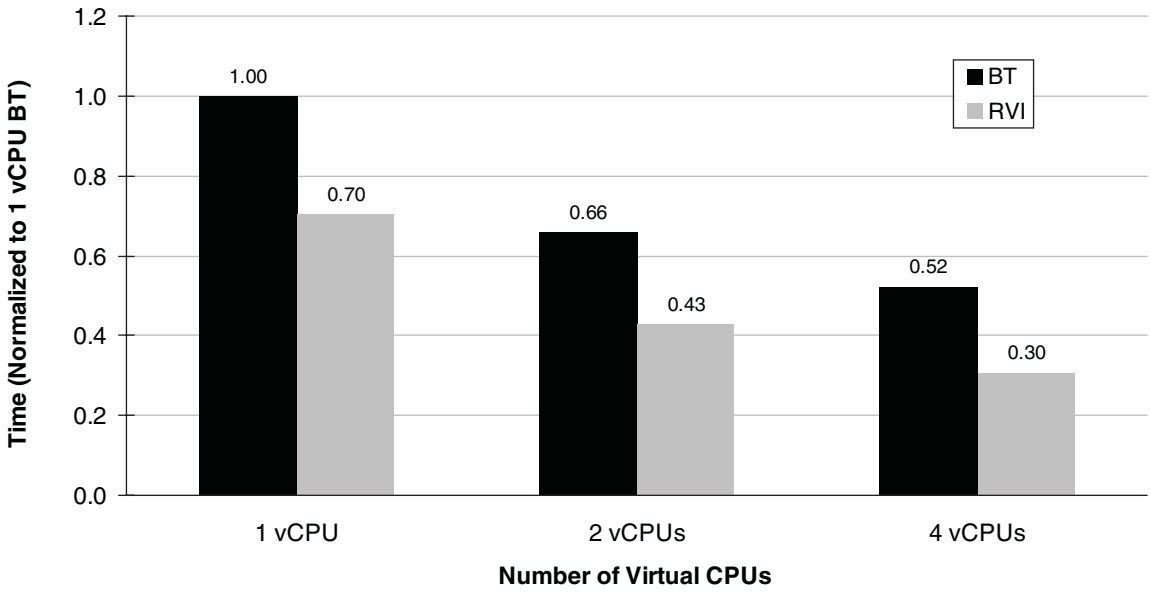
Figure 4. MMU-Intensive Kernel Microbenchmark Results (Lower is Better)



Apache Compile

The Apache compile workload compiles and builds the Apache web server. This particular application is at an extreme of compilation workloads in that it is comprised of many small files. As a result many short-lived processes are created as each file is compiled. This behavior causes intensive MMU activity, similar to the MMU-intensive kernel microbenchmarks, and thus benefits greatly from RVI, as shown in Figure 5. The improvement provided by RVI increases with larger numbers of vCPUs; in the four vCPU case RVI performed 42% better than BT.

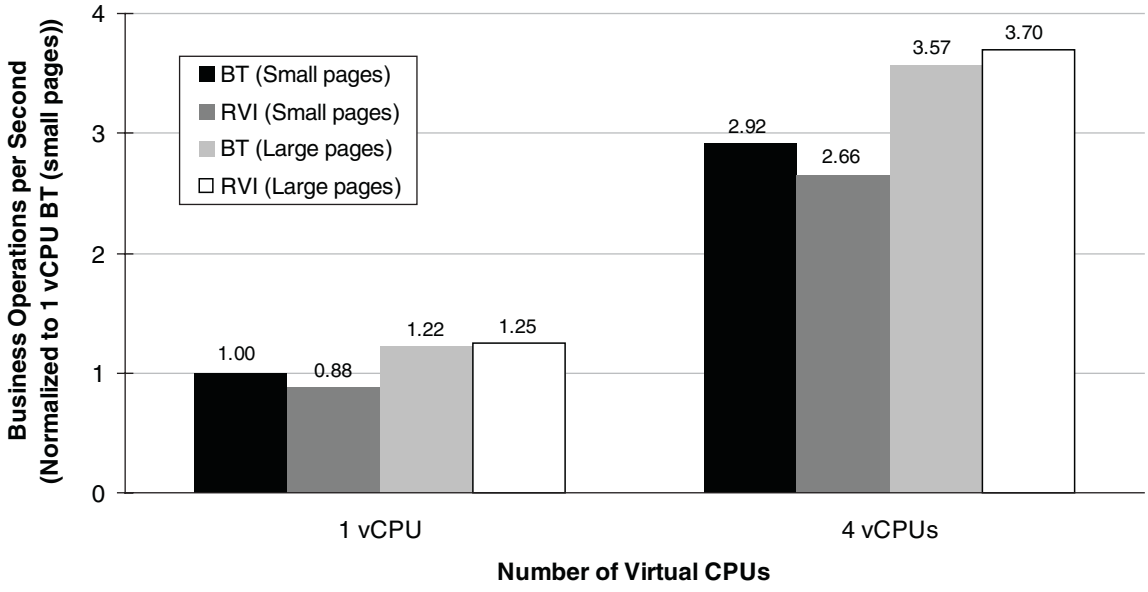
Figure 5. Apache Compile Time (Lower is Better)



SPECjbb2005

SPECjbb2005 is an industry-standard server-side Java benchmark. It has little MMU activity but exhibits high TLB miss activity due to Java's usage of the heap and associated garbage collection. Modern x86 operating system vendors provide large page support to enhance the performance of such TLB-intensive workloads. Because RVI further increases the TLB miss latency (due to additional paging levels), large page usage in the guest operating system is imperative for high performance of such applications in an RVI-enabled virtual machine, as shown in Figure 6.

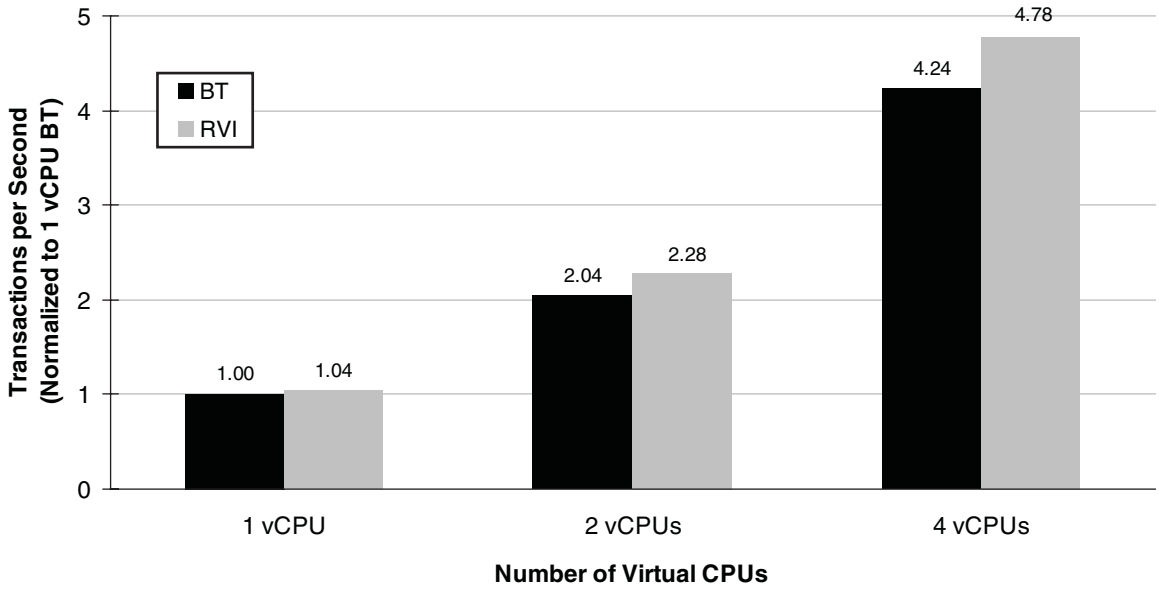
Figure 6. SPECjbb2005 Results (Higher is Better)



SQL Server Database Hammer

Database Hammer is a database workload for evaluating Microsoft SQL Server database performance. As shown in Figure 7, we observed that Database Hammer with lower vCPU counts is not MMU intensive, resulting in similar performance with and without RVI. However as we scale up the number of vCPUs we do see some MMU activity, thereby favoring RVI. We configured the guest to use large pages for all our Database Hammer runs.

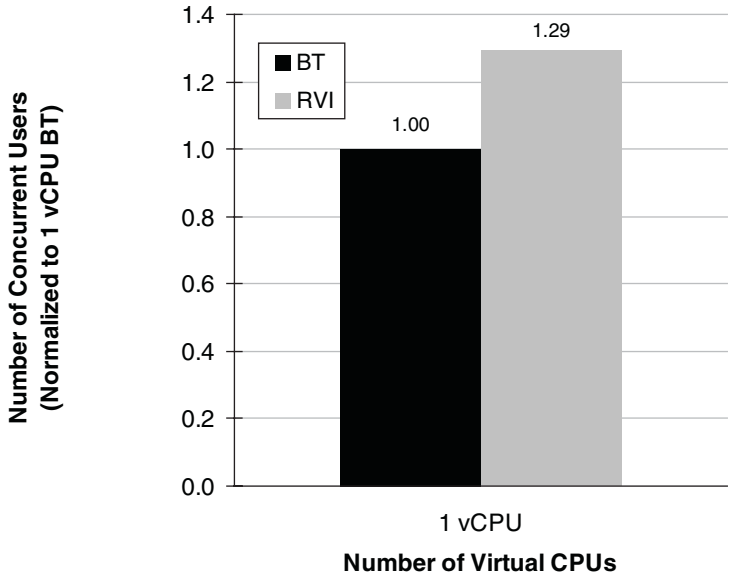
Figure 7. SQL Server Database Hammer Results (Higher is Better)



Citrix XenApp

Citrix XenApp is a presentation server or application session provider that enables its clients to connect and run their favorite personal desktop applications. To run Citrix, we used the Citrix Server Test Kit (CSTK) 2.1 workload generator for simulating users. Each user was configured as a normal user running the Microsoft Word workload from Microsoft Office 2000. This workload requires about 70MB of physical RAM per user. Due to heavy process creation and inter-process switching, Citrix is an MMU-intensive workload. As shown in Figure 8, we observed that RVI provided a boost of approximately 29%.

Figure 8. Citrix XenApp Results (Higher is Better)



Conclusion

AMD RVI-enabled CPUs offload a significant part of the VMM's MMU virtualization responsibilities to the hardware, resulting in higher performance. Results of experiments done on this platform indicate that the current VMware VMM leverages these features quite well, resulting in performance gains of up to 42% for MMU-intensive benchmarks and up to 500% for MMU-intensive microbenchmarks.

We recommend that TLB-intensive workloads make extensive use of large pages to mitigate the higher cost of a TLB miss.

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com

Author: Nikhil Bhatia

VMware, Inc. 3401 Hillview Ave., Palo Alto, CA 94304 www.vmware.com

Revision: 20090311; Item: EN-000147-01