



White Paper

Optimizing Applications for Multi-core, Virtualized Environments

By David Ott, Sr. Software Engineer
Intel Software Solutions Group

Introduction

Execution environments for today's software applications are increasingly complex, but offer a great deal more than those of the past. In the traditional model, a single set of hardware resources is managed by a single operating system designed to multiplex these resources seamlessly across applications. In this model, an application executes on a single CPU and uses a system call interface to invoke operating system assistance for device I/O and data communications.

Modern execution environments may add complexity to this traditional model in at least two ways. First, they add additional CPU resources. Today's server platforms are commonly multi-core, offering two or more execution cores within a single processor package. The additional core(s) enable parallel execution, something operating systems and applications can take advantage of to greatly improve system performance.

Execution environments may, furthermore, be virtualized. In a virtualized environment, system resources are managed by a virtual machine monitor (VMM) and multiplexed across virtual machines (VM), each consisting of an operating system and associated applications. The framework offers a number of advantages including the simultaneous hosting of heterogeneous operating systems, consolidation of server applications on the same set of hardware, and improved platform resource utilization.

For developers, the intersection of these two modifications to the traditional execution environment model is both exciting and vexing. The added features and performance offer exciting advantages over traditional environments, but developers may feel somewhat bewildered at how to proceed in optimizing their applications for both aspects of these complex environments. In this brief paper, we address the issue of how to approach performance optimization in this context.

Optimizing for Multi-core

A key observation to note is that optimizing an application for both multi-core and virtualization aspects of an execution environment may be addressed independently. To some degree, this is because virtualized environments are designed to handle program execution and resource management in a way that is transparent to operating systems and their associated applications. The basic recommendation, then, is to first optimize for multi-core, and then go on to consider performance optimizations (if any) for virtualization

To optimize an application for multi-core, a developer should use a native (i.e., non-virtualized) environment. This eliminates performance artifacts that originate from virtualization aspects of the execution environment in order to focus exclusively on multi-core aspects.

Techniques for optimizing an application for multi-core are thoroughly documented elsewhere, and hence a lengthy treatment here would merely be redundant. (For starters, take a look at the Intel Software Network (ISN) Web site.) However, here are a few ideas to illustrate what multi-core optimization might entail:

- Identify opportunities for parallelism using functional or data decomposition of the application's problem domain.
- Use threading libraries (e.g., Windows thread API, POSIX thread API) to explicitly parallelize your application.
- Consider a compiler-based threading alternative (e.g., OpenMP) to parallelize your application.
- Use Intel software tools (e.g., VTune, ThreadProfiler) to identify hotspots and apply optimizations to key code areas.

To avoid problems in the virtualized environment later, multi-core optimizations should avoid several things. First, it is best to avoid making assumptions about the underlying CPU topology. For example, it is possible to assign certain threads to specific CPUs. Doing so, however, may cause problems later when a VMM provides a different set of CPUs to the VM as part of its own resource management or configuration scheme.

It is also best to avoid microarchitecture-level optimizations. For example, it is possible to exploit specific facts about register format and organization to reduce partial and MOB stalls. Such techniques, however, may cause problems later when low-level manipulations result in VMM traps that significantly slow performance or limit the mobility of a virtual machine between virtualized servers. Similarly, low-level operating system calls dealing with system memory and interrupts can result in significant VMM overhead, as can the excessive use of CPUID, RDTSC, and other kernel-level instructions.

Finally, frequent or excessive memory allocations and de-allocations can cause significant performance overhead as page table modifications require VMM intercession. The same problem can also be induced by frequent or excessive process creations and deletions since these result in updates to the base page table pointer.

Optimizing for Virtualization

After optimizing your application for multi-core, you should benchmark its performance in the native environment. Then, run the application in your virtualized environment and compare results.

Note that some performance degradation is to be expected. This is due to VMM overhead as it manages key hardware resources on behalf of the VM. How much degradation to expect depends on a variety of factors, including the VMM vendor, VM and VMM configuration, presence of hardware virtualization features (Intel provides many), and, of course, the application itself. For example, it's not uncommon to see up to 20% performance degradation with application workloads including considerable I/O. In contrast, applications that are highly CPU intensive may experience less than 5% performance degradation since very little VMM intercession is required.

Optimizing for virtualization may largely involve adjusting VMM configuration parameters impacting the execution environment. It is important to know and follow the recommendations of your VMM vendor when making key provisioning decisions at installation/setup time. For example, adequate physical memory needs to be allocated for the VMM to insure its optimal performance when managing system resources. Many VMM vendors also provide paravirtualized drivers which perform better than standard operating system drivers used in native environments. Hardware features may also need to be enabled that assist the VMM in various ways.

Another key configuration issue is that of VM sizing. VMs that will run on the same physical host need to be provisioned with physical resources using the VMM vendor's configuration scheme. It is often very important to understand the physical requirements of your application so that the hosting VM can be adequately provisioned, but without wasting system resources. For example, a multi-threaded application may perform optimally with four vCPUs. Adding more vCPUs will show only marginal improvement and increase VMM scheduling overhead. Similarly, an application may perform far better with a certain minimum virtual memory allocation. By the same token, allocating more than necessary shows only marginal improvement, deprives other VMs of precious system resources, and increases VMM memory management overhead.

To analyze performance issues not solved by environment configuration, application developers can make use of performance tools to compare CPU utilization, disk I/O, network performance, etc. between native and virtualized environments. A significant mismatch may point to underlying issues in application code. For example, memory usage patterns may be identified that incur significant VMM overhead. Often times an algorithm can be modified to avoid VMM-sensitive issues.

Note that many performance tools offered by operating system vendors work well in a native environment but give faulty results in a virtualized environment. This is because such tools rely in various ways on system time values. In a native environment, the operating system receives clock ticks from the underlying hardware in a straight-forward and reliable manner. In a virtualized environment, however, clock ticks must be distributed to each VM by the VMM using a queuing mechanism. As such, a variety of artifacts can occur that make time values misleading. For example, clock ticks may not be processed immediately or overflow the queue making time appear to pass more slowly or erratically. In general, VMM vendor performance tools should be utilized for key performance characterizations within a virtualized environment.

Summary

As execution environments for today's software applications grow increasingly complex, application developers must become aware of new types of performance issues and optimizations. In this paper, we have discussed briefly the handling of application optimization for multi-core, virtualized environments. Developers are advised to handle each independently, to avoid certain low-level optimization practices, to understand VMM configuration and VM sizing considerations, and to compare native and virtualized performance components to identify issues and troubleshoot.

Author Bio



David E. Ott is a Senior Software Engineer with Intel's Software Solutions Group. He joined Intel in 2005 as a middleware systems engineer for the Technology and Manufacturing Group. Currently, David focuses on performance and virtualization aspects of enterprise platform technologies. David holds M.S. and Ph.D. degrees in Computer Science from the University of North Carolina at Chapel Hill.