

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

With Windows NT firmly entrenched in the low- to midrange server market, Microsoft has set Windows 2000's (Win2K's) sights on large servers with enterprise-class workloads. The current industry characterization of large servers is four or more processors and multiple gigabytes of physical memory. Variants of the UNIX OS dominate such systems because UNIX has matured throughout the 1990s in performance, reliability, manageability, and availability—aspects crucial to large-server computing. To be an attractive alternative to UNIX on large systems, Win2K must excel in all these areas.

The laundry list of features and enhancements that make Win2K an improvement over NT 4.0 addresses the shortcomings that have prevented NT 4.0 from penetrating the large-server market. In this two-column series, I highlight the new features and fine-tuning that Microsoft introduces in Win2K to make it a scalable OS. This month, I describe new features and optimizations that let Win2K better exploit large amounts of physical memory. Next month, I'll continue with a look at optimizations that improve Win2K's processor use in multiprocessor systems.

## Memory Scalability

Large servers' workloads are typically memory-intensive. The workloads' data set comprises multiple gigabytes and challenges the physical memory present on large servers. For example, you might use a large server to run a database server that manages a multigigabyte corporatwide database or several department databases with a cumulative size of several gigabytes. Other examples of large-memory workloads are enterprise resource planning (ERP), scientific, or financial analysis applications with multiple gigabytes of input data. Mass storage devices have higher latency than main memory by several orders of magnitude, as well as lower throughput, so building a server that can store all or most of a workload's data sets in main memory is important.

Consider a database query that a user runs against a 6GB database. If the server has only 1GB of physical memory, the database application must read the entire contents of the database from disk into memory during the query's processing. With a disk throughput of about 10MBps, the query would take approximately 10 minutes. But if the same server has 8GB of memory, the database application can cache the entire database in memory and will require no disk access for a query. If memory throughput is on the order of 1GBps, the query will take only seconds. The difference in query response times is the difference between a disk-bound server and a server that is suited for the workload.

Most of the time, server applications don't require access to a workload's entire data set. Instead, the applications cache the most frequently accessed portions of the data set in memory, leaving the infrequently accessed portions on disk. An example of a caching application is a Web server, which loads frequently accessed files into a memory cache for fast delivery of the files. In general, the more file data the Web server caches in memory, the less frequently the Web server has to fetch files from disk.

It seems obvious that the more memory a server has, the larger the workload the server can efficiently run. However, simply adding more memory to a server doesn't necessarily result in a server application scaling to take advantage of the memory. Efficient memory scaling has two requirements. First, an OS must be able to use the memory that might exist. Second, the OS must let server applications directly access the memory.

Most 64-bit OSs have no problem meeting either requirement. Such OSs are typically able to match 64-bit hardware in the amount of physical memory they can address. Similarly, 64-bit applications have almost 264 bits of virtual memory at their disposal, so the amount of memory that they can

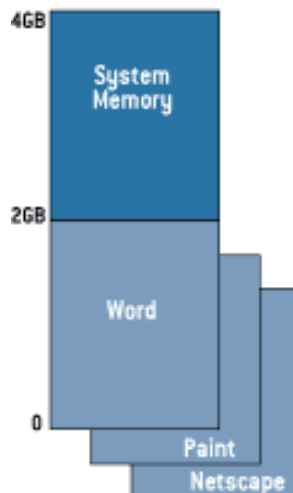
# Inside W2K Scalability Enhancements

Mark Russinovich  
(Reprinted from WindowsItPro Magazine)

directly access exceeds the amount that computer systems of the foreseeable future will support. In contrast, 32-bit OSs have several shortcomings that necessitate special features before the OSs can support large amounts of memory.

The first shortcoming in 32-bit systems is that 32-bit computer hardware design, specifically the Intel x86 line, has historically supported at most 232 bytes—4GB—of physical memory. The second shortcoming in 32-bit architectures exists because a 32-bit reference size affects applications' virtual memory address limits. A 32-bit memory address implies a virtual address space of at most 4GB.

Most OSs, including Win2K, NT, and UNIX, divide applications' virtual address space into two regions: a region that is private to each application, and a region that maps the memory that the OS, device drivers, and file system cache occupy. Figure 1 illustrates these regions. This division lets the OS and device drivers directly access application memory for efficient data transfers between applications and the OS. If the system were to give each application an entire 4GB address space and the OS a separate address space, then system calls, including file I/O, would require a relatively costly transfer of data from application address spaces to the system's address space, and vice versa.



In NT 4.0, the application-to-OS division is in the middle of the 4GB address space, such that applications have 2GB of private memory and the system assigns itself the remaining 2GB. On the x86 version of NT Server, Enterprise Edition (NTS/E), Win2K Advanced Server (Win2K AS), and Win2K Datacenter Server (Datacenter), an administrator can enable the /3GB boot switch, which moves the division so that applications have 3GB of private memory and the system has 1GB.

An application's private address space size places an upper limit on the amount of in-memory data that an application can directly manipulate. For example, on a 32-bit computer with 4GB of physical memory and a 3GB/1GB application-to-OS virtual-address split, a database server can manage at most 3GB of database data without having to read from the disk. The performance picture is complicated if the OS serves any disk reads that the application must perform from a file-system cache; in this scenario, the application might avoid actual disk I/O. Therefore, even with 3GB of private virtual memory, applications on NT 4.0 can, under some circumstances, directly and indirectly access almost 4GB of physical memory (assuming that the system has that much memory). However, applications are at the OS's mercy as to what data the system caches in memory beyond the 3GB to which it has direct access. In addition, if a server application is sharing the computer with other active applications, the OS must also divide the physical memory among those applications.

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

## Physical Memory on the Alpha

Microsoft offers Win2K for both the x86 and Alpha processors. The Alpha is a 64-bit processor that Digital Equipment (now Compaq) originally developed. Recent implementations of the Alpha (i.e., the EV5 and EV6 generations) support at least 8GB of physical memory. Different members of the Alpha line represent physical addresses with varying numbers of bits, which determines how much physical memory each processor can support. For example, the 21164PC processor, which is part of the EV5 generation, implements 33-bit physical addresses, but the newer 21264 processor uses 43-bit physical addresses. Thus, the 21164PC can use as much as 233 bytes (8GB) of physical memory, whereas the 21264 can use 243 bytes (8TB).

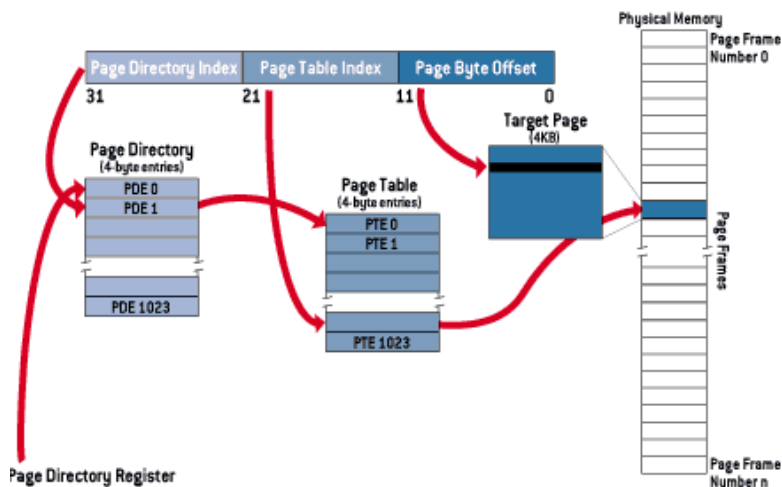
All 32-bit Alpha versions of NT, including Win2K, NT 4.0, and NT 3.1, store 35-bit physical addresses in their internal-memory-management data structures. A 35-bit representation limits physical memory support to 32GB. Therefore, although certain versions of the Alpha processor can support more memory, the 32-bit version of Win2K can use a maximum of only 32GB.

## Breaking the 4GB Physical Barrier on the x86

Running Win2K or NT forces an Alpha processor to support memory sizes smaller than its native capabilities allow. In contrast, the x86's original design supports a maximum of only 4GB of internal and external memory. For the x86 processor design to support more than 4GB of memory, changes to its design were necessary. Because Intel realized that a 4GB limit would hamper the x86 processor's growth in enterprise computing, the company added new operating modes to the x86. Intel released the Pentium Pro with a new mode called Physical Address Extension (PAE), and the company introduced 36-bit Page Size Extension (PSE36) in the Pentium II processor.

In its traditional operating mode, the x86 implements a two-level paging architecture to translate virtual addresses (which an OS and its applications use) into physical addresses (which memory hardware uses). The x86 memory management unit (MMU) divides virtual addresses into three fields, as Figure 2, page 54, shows. The CR3 special processor register anchors the page directory data structure, and the first field of a virtual address serves as an index to the directory. The MMU extracts a 4-byte address, or page directory entry (PDE), from the page directory at the appropriate index to locate a page table. The second field of the virtual address identifies the target entry in the page table.

Page table entries (PTEs) are 4 bytes (32 bits) in size. A PTE contains a 20-bit address of a physical page, and because a page is 4096 (2<sup>12</sup>) bytes on the x86, the x86 has a maximum of 220+12 bytes, or 4GB, of physical memory. The last field of a virtual address denotes the offset into the page that the PTE specifies.

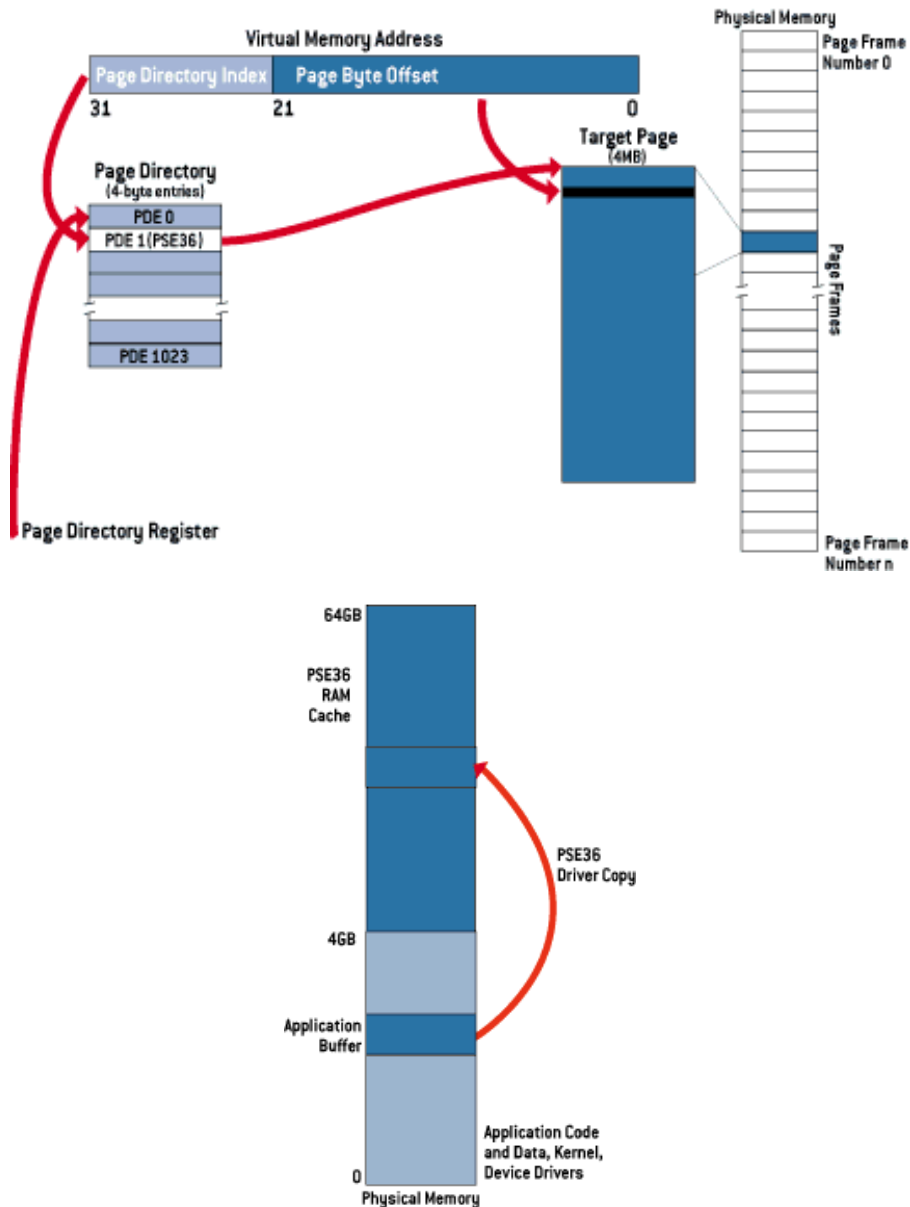


# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

PSE36 lets an OS direct the MMU to perform one-level address translation on select PDEs, a process that Figure 3 illustrates. An OS enables the translation by marking a PDE as page-size extended, so that the MMU uses the physical address in the PDE as the final page address, rather than as a page table's address. In addition, the MMU uses 14 bits for the page address, instead of a PDE's standard 20 bits, but the system interprets the pages as 4MB (2<sup>22</sup> bytes) in size. This alteration results in 36-bit physical addresses (14 bits of PDE address plus 22 bits of page size), which is large enough to reference 64GB of data. PSE36 memory's drawback is that the large page size (4MB vs. the standard 4KB) makes the page inefficient for general-purpose use. Early in 1998, Intel developed a special device driver, the Intel PSE36 Driver, to give applications using PSE36 an interface to memory above 4GB. The driver runs only under NTS/E and lets a maximum of one application use memory above the 4GB boundary as a type of RAM disk. All application and OS memory is below the 4GB boundary, so when an application wants to write to PSE36 memory, the application notifies the PSE36 Driver, which must copy the application's buffer to the specified location above 4GB. Figure 4 illustrates this memory-write process.



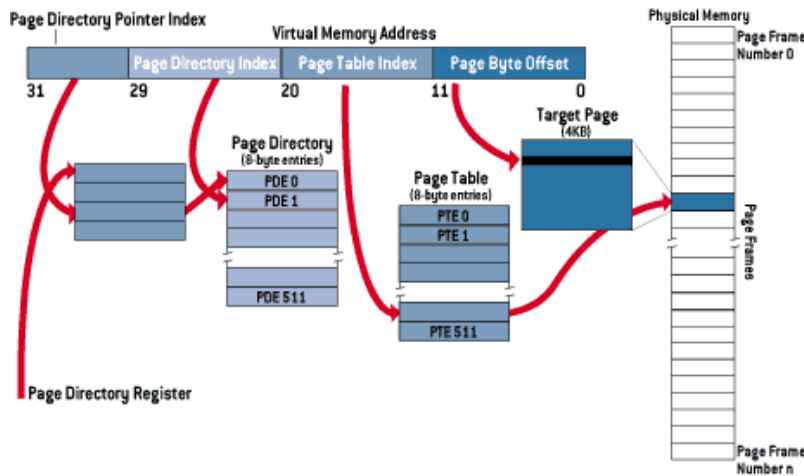
# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

The additional physical memory that PSE36 makes available to a server application typically enhances the performance of the application, which would otherwise perform disk I/O. Unfortunately, the copy operations that result when the PSE36 Driver transfers data to and from memory above 4GB can hurt overall performance. Therefore, Microsoft hasn't promoted PSE36 but chose instead to use the x86's PAE mode to implement large-memory support.

When the x86 executes in PAE mode, the MMU divides virtual addresses into four fields, as Figure 5 shows. The MMU still implements page directories and page tables, but a third level, the page directory pointer table, exists above them. PAE mode can address more memory than the standard translation mode not only because of the extra level of translation but also because PDEs and PTEs are 8 bytes, rather than 4. The system represents physical addresses internally with 24 bits, which gives the x86 the ability to support a maximum of 224+12 bytes, or 64GB, of memory. PAE's advantage over PSE36 is dramatic: An OS can use all the physical memory as general-purpose memory, so copy operations to access memory above 4GB aren't necessary.



Because PAE mode is either on or off and the mode has a different virtual-to-physical translation model than standard x86 mode has, vendors must modify x86 OSs to use PAE mode. Microsoft developed a Win2K kernel version that implements PAE memory translation on the x86; if a system is PAE-capable and has more than 4GB of memory, the boot loader NT Loader (NTLDR) loads the PAE kernel. Thus, rather than load the ntoskrnl.exe image as the kernel, NTLDR loads ntkrnlpa.exe. (Uniprocessor and multiprocessor versions of PAE and non-PAE kernels exist.) After the kernel loads, Win2K Professional (Win2K Pro) and Win2K Server restrict memory usage to 4GB. Win2K AS and Datacenter can use the additional memory above 4GB: In Win2K AS, the PAE kernel will use at most 8GB of physical memory; in Datacenter, the kernel will use the maximum 64GB, if that much memory is present. Contrast this memory usage with the Alpha versions of Win2K, all of which use up to 32GB of memory. Table 1, page 57, summarizes Win2K's physical memory support on both x86 and Alpha hardware.

| Table 1: Maximum Physical Memory Support in Win2K |           |              |          |            |
|---|-----------|--------------|----------|------------|
|   | Win2K Pro | Win2K Server | Win2K AS | Datacenter |
| x86   | 4GB       | 4GB          | 8GB      | 64GB       |
| Alpha   | 32GB      | 32GB         | 32GB     | 32GB       |

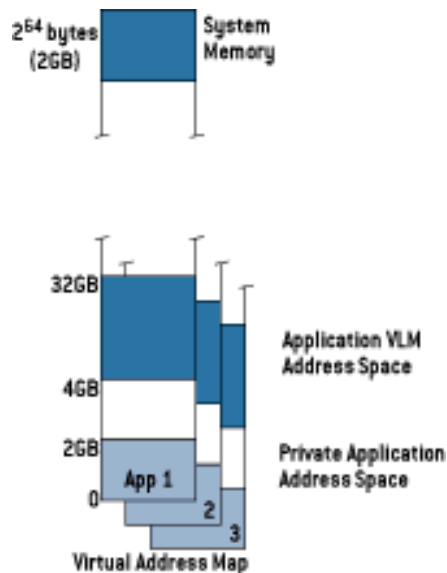
# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

## Very Large Memory

Although an Alpha 21264 processor running in 32-bit mode lets Win2K manage up to 32GB of physical memory, applications are still stuck with a 4GB virtual address space by default. Internally, the Alpha represents all virtual addresses as 64-bit values; however, the Alpha uses the sign-extension technique to translate 32-bit addresses into 64-bit addresses. When Win2K or NT creates a 4GB address space for an application, the application uses the 2GB at the bottom of the 64-bit address range and the 2GB at the top of the 64-bit address range, as Figure 6, page 57 shows. (MMUs interpret Alpha virtual addresses as 43-bit or 48-bit sign-extended values—depending on processor execution mode—so the Alpha isn't a true 64-bit processor with respect to virtual addresses.)



Early in Win2K's development, Microsoft saw an opportunity to extend Alpha applications' addressing capabilities. The company introduced the very large memory (VLM) API, whereby the Win2K kernel lets an application create up to 28GB more virtual memory in its private address space, for a total of 30GB.

Some restrictions exist regarding the VLM an application allocates. First, virtual VLM translates directly to physical memory. Thus, if an application allocates 2GB of VLM, the application is allocating 2GB of physical memory for its exclusive use. The data that an application stores in VLM resides in physical memory that the system never pages out to a paging file on disk in the way that the data and code in standard virtual memory can be paged out. A system must have a minimum of 128MB of memory for the system to enable the VLM API, and the API is available only on Win2K's Alpha version. A final restriction is that applications can use the virtual addresses they obtain via the VLM API only with other VLM APIs. This restriction exists because the virtual addresses that the VLM API returns are 64 bits wide, but most standard Win32 APIs take 32-bit parameters. An important point regarding VLM is that the physical memory limits that Win2K imposes on itself affect the total amount of VLM that applications can allocate.

## Address Windowing Extensions

During Win2K's development, Intel introduced the 450NX chipset that lets x86 processors use PAE to break the 4GB physical memory boundary, and Microsoft implemented a portable API that the company aims at systems with large memory. Applications use the Address Windowing Extensions

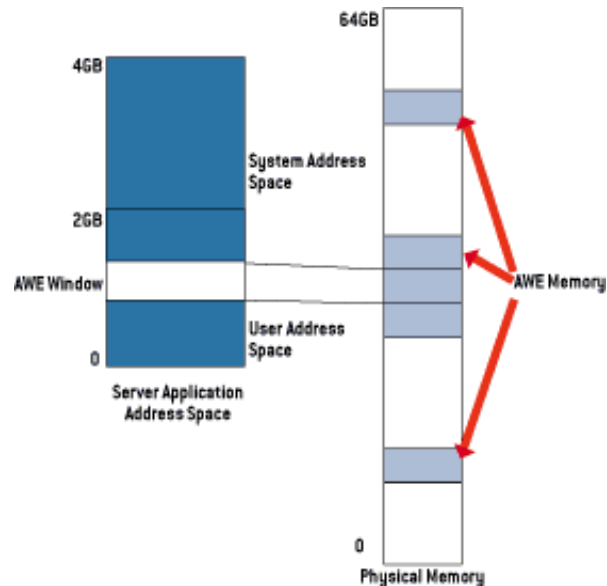
# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

(AWE) API to allocate physical memory for their exclusive use and to gain access to all or part of the physical memory that the applications allocate through a window in their address space.

To use the AWE API to allocate physical memory, an application calls the Win32 function `AllocateUserPhysicalPages`. Then, the application uses the standard Win32 API `VirtualAlloc` to create a window in the private portion of the application's 4GB address space. `VirtualAlloc` accepts the `MEM_PHYSICAL` flag, which signals to the Win2K kernel that the application is creating a physical memory window. After it allocates physical memory and creates the window, the application can map portions of physical memory into the window. For example, if an application creates a 256MB window in its address space and allocates 4GB of physical memory (on a system with at least 4GB of physical memory), the application can use the `MapUserPhysicalPages` or `MapUserPhysicalPagesScatter` Win32 APIs to access any portion of the physical memory by mapping the memory into the 256MB window. The size of the application's window determines the maximum amount of physical memory that the application can access with a given mapping. Figure 7 shows an AWE window with a physical memory mapping.



The AWE API exists on all Win2K versions and is enabled regardless of how much physical memory a system has; however, AWE is most effective on systems with at least 2GB of physical memory. Because applications have only 2GB or 3GB (depending on whether the /3GB boot switch is enabled) of private virtual memory, the AWE API gives applications a mechanism to directly control more memory than their address space would otherwise dictate. For example, on a Win2K AS system with 8GB of physical memory, a database server application can use AWE to implement almost 8GB of memory as a database cache, to which the server has direct access through its AWE windows.

AWE provides two major benefits in addition to the direct access to huge amounts of physical memory that the API enables. First, all 32-bit and 64-bit platforms uniformly support AWE; second, you can use AWE-allocated memory with nearly all the Win32 APIs.

## Improving SMP Memory Performance

In addition to kernel enhancements that let the OS and applications take advantage of large-memory systems, Win2K has several memory-related performance enhancements for operation on multiprocessors. NT 4.0 introduced the lookaside lists feature. A lookaside list is a pool of fixed-size

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

kernel memory buffers that the Win2K kernel and device drivers create as private memory caches to serve specific purposes.

When an application executes a file-system operation, such as a file read, the I/O Manager must allocate a buffer to serve as the I/O request packet (IRP) that describes the request. The I/O Manager hands the IRP to the file-system driver responsible for managing the file that the read targets. When the file system finishes servicing the read, the I/O Manager must free the buffer it used to store the IRP. Without lookaside lists, the I/O Manager must frequently allocate and free memory buffers that store IRPs. To improve performance, Win2K's I/O Manager creates an IRP lookaside list. In a situation in which the I/O Manager would usually free an IRP buffer back to the general memory pool, the I/O Manager instead stores the buffer on its IRP lookaside list. Then, when it needs to allocate a buffer to serve as an IRP, the I/O Manager checks the lookaside list. If the lookaside list stores at least one freed buffer, the I/O Manager doesn't need to call upon the general kernel buffer manager. The kernel tunes the number of freed buffers that lookaside lists store according to how often a device driver or a kernel subsystem such as the I/O Manager allocates from the list; the more frequent the allocations, the more buffers the kernel allows on a list. When a list reaches the size limit that its usage patterns determine, the kernel frees buffers from the list back to the general memory pool.

Win2K adds a twist to the lookaside list performance optimization. On a multiprocessor, the cache coherency mechanism must keep data that the system modifies in the data cache of one processor synchronized with copies of the data that the other processors might cache. Cache coherency adds overhead to a multiprocessor's execution because the cache coherency algorithms need to use the multiprocessor's data bus; this use prevents processors from accomplishing useful work. In NT 4.0, all processors share the kernel's IRP lookaside list, which means that updating the lookaside list can cause the cache coherency mechanism to degrade performance. In addition, having one lookaside list means that processors have to synchronize their access to the list using spinlocks, and spinlocks also cause overhead on the multiprocessor bus and slow down a CPU's processing. Win2K creates separate IRP lookaside lists for each processor to avoid these performance degradations.

A system duplicates roughly 10 kernel lookaside lists across processors. In addition to the I/O Manager with its IRP lookaside list, the Win2K Object Manager and Cache Manager are two other subsystems that use this technique. The kernel's general buffer manager also uses this optimization when the manager creates per-processor lookaside lists for storing 32-byte buffers. When a device driver or kernel subsystem bypasses a lookaside list for a 32-byte or smaller allocation request, the kernel's general buffer manager checks its lookaside list for available buffers.

In addition to per-processor lookaside lists, Microsoft has made several other more subtle optimizations to the Win2K Memory Manager to enhance memory scaling on multiprocessors. For example, Win2K has improved workingset tuning, a mechanism for keeping frequently accessed application data in physical memory.

## Pool Size and Cache Size

NT 4.0 implements nonpaged pool for nonpaged memory. Device drivers and the OS store data structures that must stay in physical memory and not be paged out to disk in nonpaged pool. The Memory Manager bases the pool's size on several parameters, including how much physical memory is present (the pool's maximum size in NT 4.0 is 128MB). The Microsoft TCP/IP driver, which must allocate nonpaged memory for every TCP/IP connection that is active on the computer, relies heavily on nonpaged pool. The size of nonpaged memory can therefore limit active TCP/IP connections. The TCP/IP driver and other drivers that run enterprise-class Web server workloads can push the NT 4.0 nonpaged pool's 128MB limit, so Microsoft raised the maximum size of nonpaged pool in Win2K to

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

256MB. In the process, Microsoft also packed the data structures that manage nonpaged pool more tightly to save space.

Finally, in NT 4.0, the Cache Manager can use a maximum of 512MB of the virtual address space that the Memory Manager assigns to the system. Win2K's Cache Manager raises that maximum to 960MB. This increase lets Win2K's Cache Manager more efficiently manage larger numbers of cached files because the Cache Manager doesn't need to perform as much remapping of physical memory into the cache's virtual memory. However, a larger amount of virtual memory for the cache has no effect on the number of disk I/Os that the Cache Manager performs. A common misconception is that the Win2K and NT file-system caches efficiently use physical memory only up to the virtual size of the cache. In reality, the Win2K and NT Cache Managers efficiently use all the physical memory you plug into a system.

## Multiprocessor Scaling

The changes in Win2K that let its kernel and applications use more physical memory than NT 4.0 supports extend Win2K's capabilities for handling large-server and enterprise-class data sets. Further, performance optimizations related to memory sharing among processors let Win2K run more efficiently on multiprocessors, another characteristic that will help Win2K further penetrate enterprise computing. Next month, I'll continue examining Win2K scalability enhancements and highlight Win2K features and optimizations that help the kernel and applications more effectively use multiprocessors.

I began this two-part series last month by highlighting the scalability enhancements that Microsoft has introduced in Windows 2000 (Win2K) for memory-intensive applications. These enhancements let applications directly address as much as 64GB of physical memory and introduce optimizations that increase the amount of certain types of memory (e.g., kernel nonpaged memory) and the file system cache's virtual size.

This month, I discuss Win2K multiprocessor scalability enhancements. Saying that an OS can use a certain number of processors is different than saying that the OS scales well. Scaling on multiprocessors requires that vendors develop all the levels of software that execute as part of a server application—including the OS, middleware, and layered services—with scaling in mind. If any part of the software in a server application isn't developed for scalability, the server application won't scale. Microsoft has little control over third-party middleware and applications, but the company has made the goal of OS scalability a top priority in Win2K. I conclude this series with a look at the changes Microsoft has made that enable applications to use processing cycles more efficiently and that help the kernel scale more effectively.

## Multiprocessor Support

Before diving into the SMP enhancements Win2K implements, I want to describe the theoretical and licensing SMP limitations Microsoft has placed on Win2K's various versions. So that you can see whether changes from Windows NT 4.0 exist, I'll first review NT 4.0's capabilities. NT 4.0 comes in three basic flavors: NT Workstation; NT Server; and NT Server, Enterprise Edition (NTS/E). All three versions rely on the same NT kernel, but the kernel configures itself differently for each NT type. Theoretically, the NT 4.0 kernel can support systems with as many as 32 processors. The 32-processor limitation comes from the fact that the kernel uses 32-bit values (in which each bit in a value represents a processor) to represent processors in several places internally. For example, the idle mask value identifies which processors aren't performing useful work. NT sets each idle processor's bit in the mask to 1 and sets each active processor's bit to 0. Because NT 4.0 is a 32-bit OS, this use of 32-bit values comes naturally.

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

Although NT 4.0 internally supports up to 32 processors, licensing limitations typically prevent NT from using that many processors. The:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\SessionManager\LicensedProcessors`

Registry value designates the maximum number of processors that the kernel can use. On NT Workstation 4.0, the LicensedProcessors value is 2; on NT Server 4.0, the value is 4; and on NTS/E, the value is 8. Hardware vendors that sell NT systems with more than eight processors bundle OEM versions of NT that Microsoft licenses for the appropriate number of processors.

Regarding theoretical and licensing SMP limitations, Win2K is unchanged from NT 4.0. Win2K continues to use 32-bit values internally to identify processors, and Microsoft has licensed Win2K Professional (Win2K Pro), the Win2K equivalent of NT Workstation 4.0, for two processors. Similarly, Microsoft has licensed Win2K Server for four processors, and Win2K Advanced Server (Win2K AS—the equivalent of NTS/E) for eight processors. Microsoft has licensed Win2K Datacenter Server (Datacenter), which has no NT 4.0 equivalent, for 16 processors. As with NT 4.0, hardware vendors can distribute OEM versions of Win2K that support as many as 32 processors.

## New Quantum Options and Scheduling Classes

The Win2K scheduler is a preemptive multitasking scheduler, which means that the scheduler divides processing time among the active threads running on a system. The scheduler gives each thread a quantum, which is a defined processing period. If a thread yields its turn early (e.g., when the thread stops executing to wait for an I/O operation to complete), Win2K invokes scheduler code to find another thread to take over the processor. If a thread runs to the end of its quantum, Win2K invokes the scheduler to give other threads a chance to execute. The Win2K scheduler (which Microsoft developers call the dispatcher) evaluates thread priorities to determine which thread should run at any time on each processor. The scheduler always chooses the thread with the highest priority, and if multiple threads exist with the same highest-priority value, the scheduler rotates among the threads, giving each thread a quantum of processor time. The term context-switch describes switching execution from one thread to another.

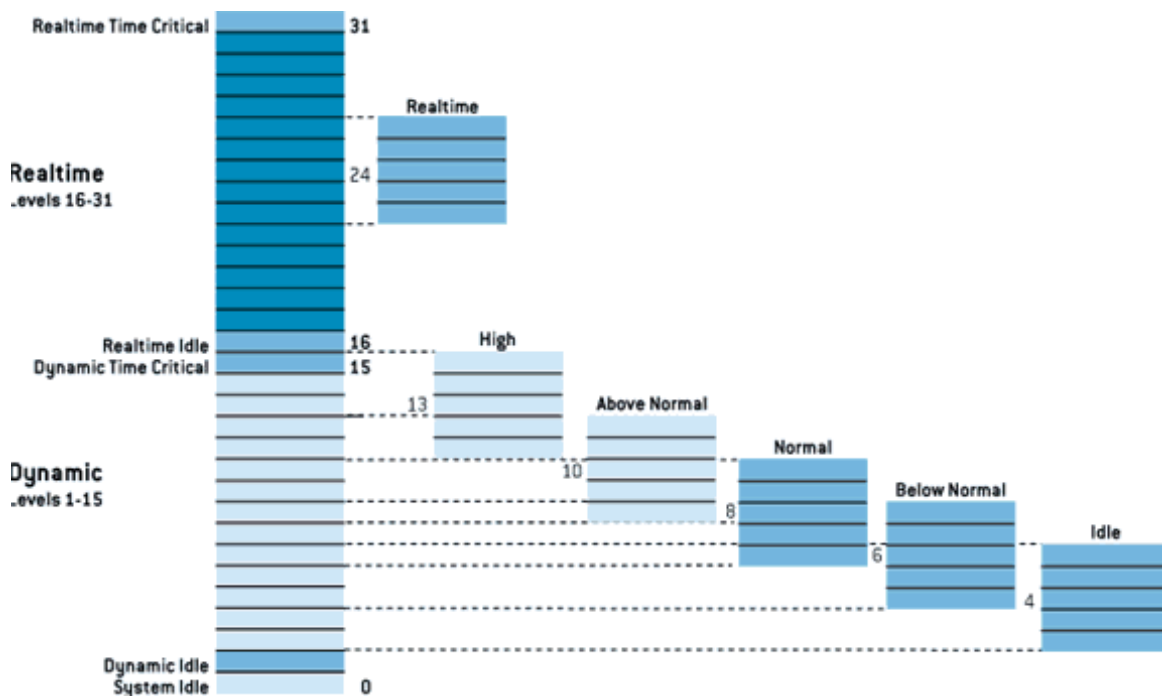
A thread's owning-process priority determines the thread's priority; a value between 1 and 31 designates the priority level. Application developers or administrators manipulate a process' priority by setting the priority to one of several possible priority classes. Each class corresponds to a value in the priority spectrum, and Win2K interprets the priority settings that programming APIs and administrative tools apply to a thread's priority as modifiers of the thread's process priority. NT 4.0 gives developers and administrators four process priority classes to choose from: Realtime, High, Normal, and Idle.

Thread-priority modifiers position a thread's priority within two priority levels of the thread's process class. The Realtime class is the highest priority class. Developers and administrators rarely use the Realtime class because threads running at such elevated priorities can interfere with the kernel's operation. Therefore, most applications use only the remaining three classes. In many cases, developers and administrators want a richer priority spectrum to more precisely prioritize individual threads or entire applications based on the tasks that the threads and applications perform. Win2K therefore introduces two more priority classes: Below Normal and Above Normal. Figure 1 shows the complete Win2K process-priority-class spectrum.

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



Microsoft defines Win2K thread quanta in terms of scheduler clock ticks. A system's hardware abstraction layer (HAL) module defines the length of a clock tick. Typical clock-tick lengths are 7ms, 10ms, and 15ms. Short quanta are appropriate for systems running multiple interactive applications because applications more frequently get turns to respond to user activity. Long quanta are best for systems running a handful of noninteractive server applications because overall performance improves with fewer context switches.

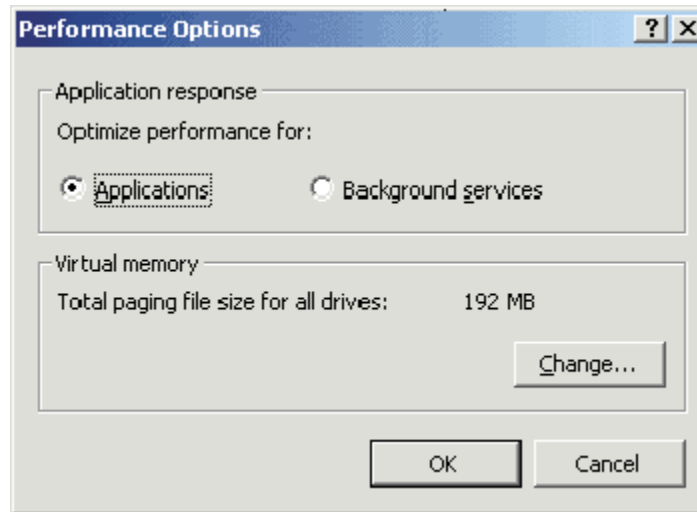
On NT Workstation 4.0, the scheduler assigns quanta that are 2 clock ticks long to threads associated with background windows or applications that have no windows. A foreground application receives quanta that are 2, 4, or 6 clock ticks long, depending on the setting that an administrator chooses for the foreground-application boost slider in the Performance tab of the Control Panel System applet. A 6 clock-tick foreground quantum corresponds to a maximum performance boost on the slider. NT Server 4.0 assigns 12 clock-tick quanta to all threads, regardless of whether the threads are associated with a foreground window. (The slider functions on NT Server but has no effect.)

Although NT 4.0 quanta values are usually suitable for the different workloads that NT Workstation and NT Server run, cases exist in which those quanta values aren't suitable. In such cases, application performance, interactivity, or scalability suffers. As a result, Win2K lets administrators select the best quantum setting for a workload on any Win2K OS. The Win2K Control Panel System applet's Performance Options dialog box, which Screen 1 shows, lets administrators optimize performance for applications or background services. Optimizing performance for applications applies the quanta that NT Workstation 4.0 uses (i.e., background threads receive 2-clock-tick quanta and foreground applications receive 6-clock-tick quanta). Optimizing performance for background services applies the quanta that NT Server 4.0 uses (i.e., all threads receive 12-clock-tick quanta).

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



## The Job Object

Many server workload types comprise related processes that cooperate to perform a computational or data-processing task. For example, a data-mining task might require the cooperation of a database client, middleware application, and custom analysis program. A server will often execute such multiple workloads simultaneously, or run the workloads as background operations while the server executes a client/server application (e.g., a database or Web server) in the foreground. Thus, letting an administrator or programmer control the processes of a task as a unit and apply limits on various characteristics of the task so that the task is prevented from interfering with the server's operation is desirable. For example, an administrator needs to assign a low scheduling priority to all the processes of background operations so that the task minimally impacts foreground execution.

NT 4.0 lacks programmatic and administrative interfaces to manage the processes in a group of related processes. Although interfaces exist to manipulate individual processes, control applications have difficulty determining programmatically which processes make up a common task, particularly when the task dynamically creates new processes as part of its control flow. Further move, other than setting the priority of a process, few options exist for limiting a process' resource usage.

Microsoft has addressed this limitation in Win2K by introducing a new process object, the job object. A job object is a container for related processes; job object programming APIs allow simultaneous management of all processes associated with a job and provide a variety of resource-limitation options. Figure 2 shows an example of a job object. A group of related processes might include special code that the group uses to manage itself, or administrative utilities can let administrators manage arbitrary applications as jobs. For example, Sequent has developed an administrative job object tool that Microsoft will include with Datacenter.

Developers use the AssignProcessToJobObject API to add a process to a job. When a process belongs to a job, you can't remove the process from the job. And, unless the developer dictates otherwise, any processes that a job's process creates or any of the process' descendents also join the job. The TerminateJobObject API provides the ability to terminate all a job's processes at one time. Another API lets a developer associate a completion port synchronization object with a job. An application can wait on a job's synchronization object to receive notification of various job events, such as the addition of a new process to the job or the termination of a job process, either because the process terminated normally or because of an error. The job object's core is the QueryInformationJobObject and SetInformationJobObject APIs. These APIs let an application monitor

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

the collective resource usage of a job's processes and impose resource limitations or performance characteristics on the job.

Figure 3 lists the various limits that a job supports, including limits on memory usage, processor time consumption, number of processes, and various security restrictions. For example, a programmer can specify that an individual process in a job can't use more than a certain amount of virtual memory, or that the job's total memory usage can't exceed a particular value. When the job reaches a memory limit, the Win2K memory manager won't give the job more memory. Job object APIs give developers a choice of two behaviors when a job exceeds a CPU time limit. The first option is for Win2K to terminate the job and all of its processes; the second option is for Win2K to deliver a message to the job's completion port, if the port exists.

**Figure 3: Job Object Limit Types**

Access Token Privileges  
Access Token Security IDs (SIDs)  
Active Processes  
Clipboard Access  
Job CPU Time  
Job Memory Allocation  
Maximum Process Working Set Size  
Minimum Process Working Set Size  
Process Memory Allocation  
Process Priority Class  
Scheduling Class  
Video Display Settings  
Window and Desktop Access  
Process CPU Time

The security restrictions that job object APIs can apply to a job are flexible. For example, giving a background job the ability to reboot a server is undesirable. If an administrator runs a job, the ability to reboot a server would usually occur in the job's processes' security token because tokens reflect the security credentials of the account they are created with. One class of security restriction therefore enables the removal of certain privileges from the security tokens that belong to the job's processes. A programmer could remove the reboot and other privileges to prevent a job from disrupting the server's operation. Other security restrictions remove certain user-account associations from a job. If a job is executing in Joe's account and Joe belongs to the Administrators group, Joe might want to remove the Administrators group from the job processes' tokens so that the processes can't access resources such as files, devices, and synchronization objects that only administrators can usually access.

Performance characteristics that you can control with a job object include process priority classes and thread quantum lengths. Using a job object, you can simultaneously set the process priority class of every process in a job, a useful technique for assigning the job a priority among other system processes. If you want the job to execute in the background, you set a low priority class, such as Below Normal. Win2K designates the quantum lengths that a job object supports as scheduling class values from 0 through 9. On Win2K server versions, each scheduling class corresponds to a different quantum length. Scheduling class has no effect on Win2K Pro.

Table 1 shows quantum lengths for each scheduling class. When you apply scheduling class 9 to processes in the Realtime priority class, special behavior results: The Realtime process-thread quanta become infinite. In other words, the Win2K scheduler never ends the turn of a thread with a priority higher than 15 and a scheduling class of 9. Developers and systems administrators therefore need to be careful with this designation. Setting the scheduling class to a value higher than 5 requires the

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

increase base priority privilege because scheduling-class values higher than 5 give a job longer quanta than standard processes possess.

**TABLE 1: Job Object Scheduling Class Quantum Lengths**

|                  |   |   |   |   |    |    |    |    |    |    |
|------------------|---|---|---|---|----|----|----|----|----|----|
| Scheduling Class | 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7  | 8  | 9  |
| Clock Ticks      | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |

With the job object, Microsoft has introduced a powerful and flexible way to control entire trees of related processes. What might not be immediately obvious is that the performance settings and resource limitations you configure with a job object can either help a particular task scale better or, as is more often the case, help the rest of a system scale better by degrading the capabilities of a particular job.

## Thread Pooling

Virtually every application with the goal of scaling well on multiprocessors includes asynchronous I/O operation and multiple threads that cooperate with one another. If an application has only one thread, the application can execute on only one CPU at a time. However, if the application has at least as many threads as there are processors, the application can perform work on all processors of an SMP system simultaneously. Asynchronous I/O lets an application's threads initiate I/O operations but not idly wait for the operations to complete. Instead of waiting, the threads perform other work.

Implementing multithreaded code is tedious, particularly if threads are necessary for only certain parts of the application or if a developer wants to maintain an appropriate number of threads as the application's workload characteristics vary. Having too many threads can hurt an application's performance just as surely as having too few threads can. Win2K introduces thread pooling, a new API that makes the job of efficiently using threads easier.

Each application has a private pool of threads that the ntdll.dll system library manages. The kernel32.dll Win32 API library presents the thread-pooling interface to an application but relies on NTDLL to implement the APIs. An application uses the thread-pooling API whenever the application wants to efficiently execute specific functions with different threads. The functions might execute as the result of timer expirations, an I/O operation's completion, or simply because the application has queued the function to run in the thread pool. NTDLL dynamically creates and deletes threads for the thread pool as the demands of the application dictate.

When an application associates a function with the thread pool, the application specifies what type of processing the function will perform. Processing options include CPU processing, I/O operations, and long-running activity. These options aid the thread pool in tuning the number of active threads that the pool creates. A thread pool maintains two thread types: one type for CPU-processing, and another type for I/O operations. The I/O-operation threads use wait-semantics to wait for new work items from an application. Wait-semantics lets the threads receive notification when the I/O operations that the threads scheduled within application functions complete. If an application designates a function as long-running, the thread pool is more aggressive about creating new threads to handle subsequent requests because the pool knows that a thread might remain busy while it executes the long-running function.

The thread pool will increase the number of each thread type to an upper limit of about 2000 under extreme conditions, as when an application queues many long-running functions. However, the thread

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

pool tries to keep the number of active threads to fewer than 10. Thus, an application can easily take advantage of multithreading simply by handing work to the thread pool, letting the pool worry about the details of thread management.

## The Queued Spinlock

In addition to the new APIs I've described, Win2K includes numerous subtle multiprocessor-scalability-related changes beneath the surface. First, Microsoft has optimized the kernel's memory-footprint layout and the memory-footprint code locality for most system components such as DLLs and driver files. These optimizations ensure that pieces of code that execute together are stored near one another and thus place fewer demands on a computer's memory subsystem. Microsoft calls this technology Lego. Lego works in two steps. In the first step, executable code generates a Basic Block Testing (BBT) trace, which is a trace of the code's execution. In the second step, postprocessing occurs to analyze the code trace and reorganize the way in which the code is stored. The resulting reorganized code is the code that Microsoft ships.

Another performance optimization is the kernel's use of spinlocks. Spinlocks are synchronization objects that the kernel uses to synchronize access to data structures across an SMP system's processors. For example, if the kernel is accessing the scheduler database on one processor, the kernel must not simultaneously manipulate the database on the system's other processors. The kernel therefore acquires a spinlock before accessing the database and releases the lock after the access operation completes. If the kernel has already acquired the spinlock on a different processor before accessing the database, the kernel will spin in a tight busy-loop while it waits to acquire the spinlock for the database, hence the origin of the term spinlock.

A spinlock serializes the execution of code that it's protecting, and because parallel execution is required for good scalability, spinlocks are detrimental to multiprocessor scalability. Microsoft has extended considerable effort to fine-tune the kernel's use of spinlocks so that the kernel acquires spinlocks only when necessary. Further, Microsoft has introduced a new type of spinlock called a queued spinlock.

Queued spinlocks scale better on multiprocessors than standard spinlocks do. Queued spinlocks work in this way: When a processor wants to acquire a queued spinlock that another processor is holding, the first processor places its identifier in the spinlock's associated queue. When the processor that is holding the spinlock releases it, the releasing processor hands the spinlock to the next processor waiting in the queue. In the meantime, processors waiting in the global spinlock queue spin by monitoring the status not of the spinlock but of a per-processor flag that the releasing processor sets to signal that the turn of the next processor in the queue has arrived.

Two effects follow from the fact that queued spinlocks cause processors to spin on per-processor flags rather than on the global spinlock. The first effect is that the multiprocessor's bus doesn't experience the heavy traffic of interprocessor synchronization. The second effect is that queued spinlocks enforce first in/first out (FIFO) access to the lock. FIFO ordering ensures more consistent performance across processors that access the same locks.

Microsoft hasn't converted all the Win2K kernel's locks to queued spinlocks—only the dozen or so locks that protect the core data structures of the kernel, such as the Cache Manager's database, the scheduler's thread database, and the Memory Manager's physical memory database. Queued spinlocks will undoubtedly give Win2K a significant scalability boost over NT 4.0.

# Inside W2K Scalability Enhancements

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

## Scaling the Enterprise

With the scalability enhancements I've described in this two-part series—including memory sizing enhancements, APIs for accessing large amounts of physical memory and for controlling groups of processes as a unit, and tweaks to resource and processor management—Win2K is well equipped to help applications take advantage of enterprise-class hardware. The result is that Win2K is positioned to pick up where NT 4.0 left off and better address the mid- to high-end server space.