Virtualized and emulated systems differ in how each instruction is executed. In a virtualized system, applications and most of the operating system run directly on the processor, whereas in an emulated system, the emulator must simulate or translate each instruction in order to execute it. Therefore, an emulator introduces more overhead than a virtual machine monitor for the same platform.[#]

However, even though they use parts of Bochs and QEMU, Xen's hardware virtual machines are virtualized and not emulated. The Bochs code provides the BIOS, which supports the boot process, and QEMU provides emulated drivers for a range of common devices. However, these pieces of code are only invoked at startup and when an I/O operation is attempted. The majority of other instructions run directly on the CPU.

At this point, you might wonder what happened to the much-vaunted advantages of paravirtualization. Surely all these emulated devices, shadow page tables, and additional exceptions would lead to poor performance? It's often true that a naïvely hardware-virtualized operating system performs worse than a paravirtualized operating system, but there are two mitigating factors.

First, the processor vendors are continually developing new features that optimize virtualization. Just as a *memory management unit* (MMU) lets programmers deal with virtual rather than physical addresses, an *IOMMU* does the same for input and output devices. An IOMMU can be used to give a virtual machine (whether hardware-virtualized or paravirtualized) safe, direct access to a piece of hardware (Figure 7-9). The normal problem with giving a virtual machine direct access to hardware is that many devices can perform DMA, and therefore without an IOMMU, it can read or overwrite other virtual machines' memory. The IOMMU can be used to ensure that while a particular virtual machine is in control, only memory belonging to that virtual machine is available for DMA.

Figure 7-9 illustrates a simplified DMA request from a virtual machine (DomIO) using an IOMMU. The hardware driver uses pseudophysical (virtual machine–specific) addresses when communicating with the device (1). The device makes DMA requests using these addresses (2), and the IOMMU (using I/O page tables, which are configured by the hypervisor) converts these to use physical addresses (3). The IOMMU also stops any attempts by the virtual machine to access memory that it does not own.

[#] KQEMU is a Linux kernel module that enables user-mode code—and some kernel-mode code—to run directly on the CPU. Where the host and target platforms are the same, this provides a huge speed-up. The result is a hybrid of emulation and virtualization.