Direct Access to Graphics Card Leveraging VT-d Technical Report

Beng Heng Ng, Billy Lau and Atul Prakash

July 20, 2009

Abstract

Following the rapid adoption of Type 1 virtualization [1], the capability to provide native graphics card access will further spur the adoption of the virtualization technology in fields such as graphics processing and rendering, and gaming. In this technical report, we detail the necessary ingredients to provide VGA passthrough in Xen for Dom0 guests. We also present the results of our experimentations on various graphics cards and setups, demonstrating the feasibility of the technology, despite limitations.

1 Introduction

Virtualization is not a new concept, and has been around for at least two decades. However, it is only in recent years that the adoption rate has increased significantly. Xen is one of the major players in the field of virtualization.

One of the notable events in the not-too-distant history of virtualization is the extension of the x86 processors by Intel and AMD to include the VT-x and AMD-V features respectively. These features allow the fulfillment of the requirements for virtualization set forth by Popek and Goldberg, whereby the virtual machine monitor must have three properties: efficiency, resource control and equivalence.

Xen began leveraging the hardware support for virtualization in version 3.0. This advancement allows paravirtualized guests to make hypercalls into the hypervisor, thus obviating the need for any changes to the guest OS.

Shortly after, Intel proceeded to develop the VT-d technology, while AMD developed a similar technology called IOMMU. In general, however, this technology is called IOMMU regardless of the chip maker. IOMMU greatly increases the potential for better performances by the guest OSes when accessing hardware devices. Without the hardware assistance, it is not possible to allow guest OSes secure access to the devices [2]. The IOMMU translates the virtual addresses seen by the devices into physical addresses during operations such as Direct Memory Access by the guest operating systems.

One of the commonly used devices that can benefit from the direct I/O technology is the graphics card. The graphics card is constantly updating the

display device. Sometimes it is required to perform intensive calculations, such as during rendering tasks. By allowing the guest OS direct access to the graphics card, video intensive tasks can be improved substantially. Another benefit of allowing direct access to the graphic card is that the functions of the graphic device can be exposed to the guest. Thus, the guest's video display capabilities are not restricted by the emulated graphics. This is particularly important for hardware accelerated graphics.

In this report, we discuss the necessary requirements to allow a guest OS to have direct access to the graphics card in Xen. We will follow the common terminologies used by Xen. We assume the scenario where we do not install any additional front-end driver in the guest. Thus, we discuss only HVM guests. In addition, since Xen supports Intel's architecture rather than AMD's, we will discuss only graphics passthrough on Intel systems. We focus our work on PCIe graphics cards, which are fast replacing the aging AGP cards.

Jean Guyader and his team at Citrix Systems began work on passing through graphics devices around April 2008 [3]. Since then, Citrix Systems has released several patches that enable this feature [4][5]. Yuri provided a discussion of Xen VGA passthrough [6]. We extend the discussions with more details, as well as experimental results.

We discuss some background information before delving into the requirements for graphics passthrough to work. We then provide the results and evaluations for our experiments, and finally the conclusion.

2 x86 Memory Layout

In the x86 architecture, several real mode address ranges are hardcoded so that the system can locate the initialization routines. Despite being insufficient for today's systems, these standardized address ranges are strictly adhered to for interoperability. Table 1 lists the memory layout of a typical x86 system.

To allow DomU access to the native features of the physical video adapter, certain IO and memory addresses need to be mapped from DomU to the physical address. The relevant real mode address ranges are highlighted in Table 1.

Address Range	Type	Purpose
00000000 - 000003FF	Conventional Memory	Real Mode Interrupt Vector Table
00000400 - 000004FF		BIOS Data Area
00000500 - 00007BFF		Conventional Memory
00007C00 - 00007DFF		Boot Sector
00007E00 - 0009FBFF		Conventional Memory
0009FC00 - 0009FFFF		Extended BIOS Data Area
000A0000 - 000AFFFF	Upper Memory Area	VGA Framebuffer (EGA/VGA/XGA/XVGA Graphic Video Buffer/RAM)
000B0000 - 000B7FFF		Monochrome Text VGA Buffer/RAM
000B8000 - 000BFFFF		Color Text VGA Buffer/RAM (CGA/EGA)
000C0000 - 000C7FFF		Video BIOS ROM
000C8000 - 000CBFFF		IDE Hard Disk BIOS ROM
000CC000 - 000DFFFF		Optional Adapter ROM BIOS or RAM Upper Memory Blocks (UMBs)
000E0000 - 000EFFFF		System BIOS PnP Extended Information
000F0000 - 000FFFFF		System BIOS ROM (entry at 000FFFF0)
00100000 - 0010FFEF	High Memory Area	High Memory Area
0010FFF0 onwards	Extended Memory	Extended Memory

Table 1: Memory Layout of the x86 Architecture.

3 PCI Function Configuration Register Space

The reader may skip this section without missing the necessary information required for VGA passthrough. However, this section will allow the reader a better understanding of the Xen source code, particularly in relation to how PCI devices are enumerated.

The PCI standard supports 256 buses. Each bus can be connected to 32 devices. Every device can implement a minimum of 1 function, and a maximum of 8 functions. For every function, there is a configuration register space of 256 bytes. The configuration space is typically accessed by having the first 8 bits to specify the PCI bus, the next 5 bits to specify the device on the bus, and the last 3 bits for identifying the function to be accessed. This information is usually summarized using the Bus:Device:Function (B:D:F) format [7]. The function configuration register space comprises a standardized 64 byte header, and another 192 byte device specific configuration register space, thus making a total of 256 bytes. The details of the configuration space are shown in Figure 1.

There are three types of standardized headers, 0, 1 and 2. Type 0 headers are used by PCI devices such as network, device and storage devices. A type 1 header defines a PCI to PCI bridge. Type 2 header specifies CardBus bridges where the host type is PCI. We are primarily concerned with type 1 headers.

Some of the registers in the header (shaded in Figure 1) are required to be configured by the PCI manufacturers. They are used for detecting the presence of connected devices, and for identifying them. The OS can also use these registers to load the correct device drivers. Specific device types may require other registers to be mandatory.

When a system boots up, the Boot ROM configuration firmware and the OS can configure these registers. Of particular importance are the Base Address Registers (BARs) that store the address of the resources allocated for use by the PCI device. Two types of resources can be allocated: memory and IO. The BAR formats for the memory and IO types are shown in Figure 2 and Figure 3 respectively. The required address space and type can be queried by writing all 1's to the register and reading back the value. If bit 0 is 0, then memory space is requested. The first bit from bit 7 to 31 that has a value of 1 will indicate the size required. This implies that the smallest memory space that can be allocated is 256 bytes. The address of the allocated space is then written from bit 7 to 31. If, however, bit 1 has the value 1, IO space is requested, and the allocated IO address can be written from bit 2 to 31.

4 Function Level Reset

Function Level Reset (FLR) allows specific functions for a PCI device to be reinitialized [9]. Not all PCI devices support FLR. This feature allows the system to reset the PCI device to a known state, and is important for proper handing over of passed through devices between domains. Much of the support for FLR is implemented in the pciback driver [10].

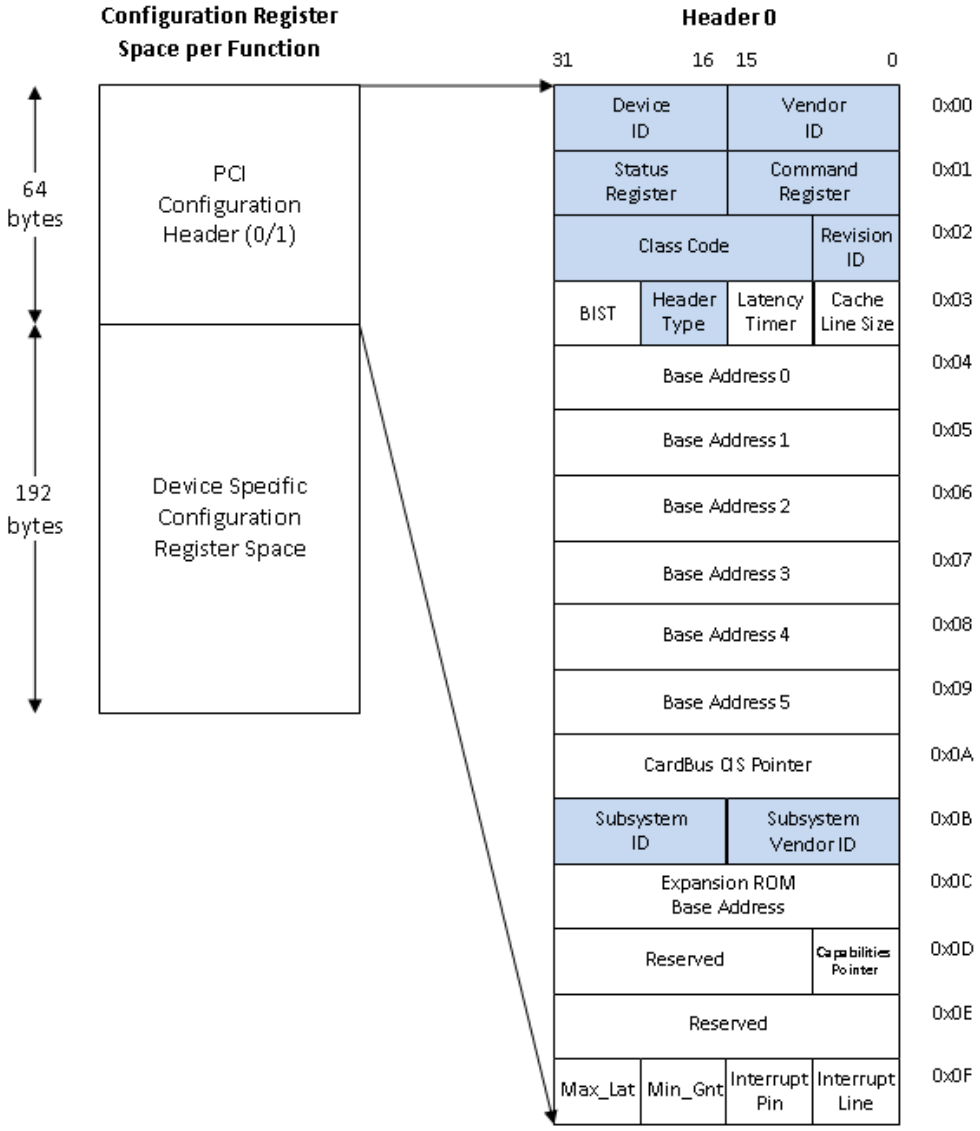

Figure 1: PCI Function Configuration Register Space. The shaded registers in Header 0 denote required configuration registers. Adapted from [8].

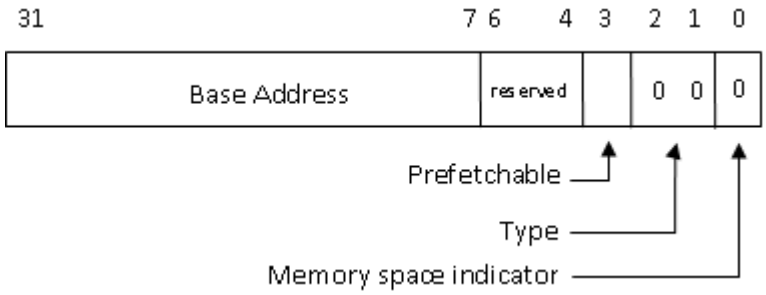

Figure 2: 32-bit Memory BAR format. Adapted from Budruk et al [8].

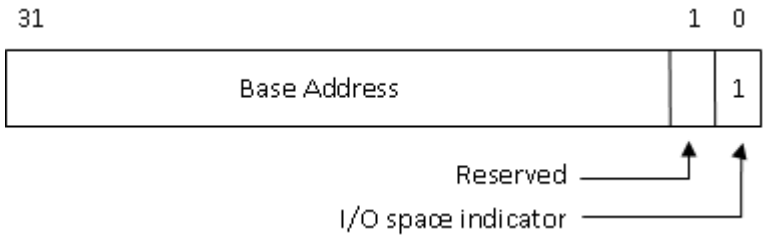

Figure 3: IO BAR format. Adapted from Budruk et al [8].

5 Prerequisites for Graphics Passthrough

Apparently, it is a fundamental requirement that the processor must support VT-d. Dom0 must be either 64-bits or support PAE. However, DomU may include 32-bit OSes as well [11].

One of the requirements for DomU to boot correctly with the physical video adapter is that it must support FLR. . This is essential for video adapter to be initialized properly during DomU's bootup. In the absence of this feature, certain functions of the adapter may be in an unknown state when another domain takes control.

6 Passing Through VGA Card to DomU

The following are the basic steps required for passing through the primary graphics adapter from Dom0 to DomU [12].

- Disable loading of the emulated VGA adapter in Qemu
- Allocate memory and copy VGA BIOS into DomU's address space
- Map legacy VGA IO ports and memory mapped IOs from DomU's address to the host's address
- Intercept reads in the host bridge

Offset	Purpose
0x00	Value 0xAA. ROM signature byte one.
0x01	Value 0x55. ROM signature byte two.
0x02	Initialization size (in multiples of 512 byte). Actual code size copied into RAM.
0x03 - 0x05	Entry point.
0x06 - 0x17	Reserved for application-unique data.
0x18 - 0x19	Pointer to PCI Data Structure.

Table 2: PCI Expansion ROM header format for Intel x86 architecture.

6.1 Disable Emulated VGA Device

Xen supports up to 128 PCI peripherals. When hvmloader starts the PC emulator, it enumerates through all the PCI devices available, allocating and mapping the requested resources as described in Section 3. These devices may be either emulated or passed through devices.

For graphics, Xen emulates either the Cirrus Logic's GD5446 adapter or Bosch's virtual VGA adapter. The emulated Cirrus Logic adapter has vendor id 0x1013 and device id 0xb8. The Bosch's adapter shares the same vendor id as Technical Corp, having the value 0x1234. It is assigned the device id 0x1111. When passing through a physical graphics device, we do not initialize the emulated graphics. This is implemented in the main function in `tools/ioemu-remote/hw/pc.c`.

6.2 Replicate Physical VGA BIOS

The VGA BIOS address usually ranges from C0000 till C7FFF, except for perhaps embedded devices. It is required only during booting of DomU. Once the operating system takes control, the VGA BIOS is no longer required [13].

To allow DomU's system BIOS to initialize the graphics device properly, we need to copy the BIOS into RAM and map it to 0xC0000. We first retrieve the VGA BIOS from the `/dev/mem` device by looking for the PCI expansion ROM signature, 0xAA55 in little Endian format, as shown in Table 2 [8]. We can then read off the size of the code, and copy the VGA BIOS to our allocated memory. The allocated memory is mapped to 0xC0000 using `xc_map_foreign_range`. The machine frame number is specified as 0XC0 since the page size is 4KB.

These are achieved in the functions `xc_get_vgabios` and `setup_vga_pt` which can be found in `tools/libxc/xc_linux.c` and `tools/libxc/xc_hvm_build.c` respectively.

6.3 Passthrough VGA IO Ports and Framebuffer

Various aspects of the VGA graphics card can be controlled by accessing the VGA registers. The registers can be categorized into five groups according to their functionalities: general, CRT controller, sequencer, graphics controller

and attribute controller [14][15]. These registers can be accessed through the IO port addresses 0x3B0 through 0x3BB and 0x3C0 through 0x3DF. To relinquish control of the video card to DomU, DomU must have direct access to these IO ports.

In addition, the video framebuffer ranges from 0xA0000 till 0xBFFFF for the x86 architectures. DomU requires direct access to these address ranges for graphics passthrough to work.

For passing through the IO ports and framebuffers, we perform the mappings using the hypercalls `xc_domain_ioport_mapping` and `xc_domain_memory_mapping`. These steps are implemented in the `register_real_device` function in `tools/ioemu-remote/hw/passthrough.c`.

6.4 Intercept Values in Host Bridge

For Intel IGD, the GMCH Graphics Control Register (GGC) located on the host bridge at offset 0x52 specifies the amount of stolen memory that is allocated [16]. The stolen memory is allocated from the system memory for use by graphics devices that do not have integrated memory, such as the Intel IGD. Since the emulated host bridge is unaware of this requirement, we need to intercept any reads from 0:0:0 offset 0x52 with an appropriate value specifying the memory size and video mode to be supported.

The AGPgart driver module in Linux leverages the system memory to improve graphics performance. The operating system only loads the module if querying the Vendor ID and Device ID identifies a compatible chipset. Similarly, we need to intercept the reads for these values and return valid ones for proper initialization by the guest operating system.

The IGD OpRegion allows features such as ambient light awareness and display output selection using hotkey [17]. Experiments suggest that Windows Vista will attempt to write to an ACPI Non Volatile Storage region if the OpRegion address is available. This appears to cause instability. We can however revert to using System Management Interrupts by returning 0 when the operating system queries for the address of this region [17].

We perform the interception of these values in the function `pci_data_read` in `ioemu-remote/hw/pci.c`.

7 Implementation

We tested graphics passthrough on Xen 3.4.1 rc2 pre. The changeset is 19645.

We experimented with the following graphics devices and installing the latest drivers available as shown in Table 3. Both the ATI and nVidia cards are add-on devices, while the Intel graphics device is an integrated graphics chipset. We performed our tests on Windows XP 32-bit and Vista 64-bit DomU.

As no graphics benchmarking tool is available for Linux, we did not perform our tests on the OS. In addition, graphics passthrough may not work well with

	XP32	Vista64
ATI Radeon HD 2600 XT	ATI Catalyst Software Suite v9.5	ATI Catalyst Software Suite (64 bit) v9.5
Intel Graphics Media Accelerator 4500	Intel Graphics Media Accelerator Driver for Windows XP v14.38.3.5047	Intel Graphics Media Accelerator Driver for Windows Vista 64 v15.13.64.1688
nVidia GeForce 9500 GT	GeForce Driver Release 185 for XP (32 bit) v185.85	GeForce Driver Release 185 for Vista (64 bit) v185.85

Table 3: Drivers installed for graphics cards on test OSes

```
[12.520404] Pnp: PnPACPI:METHOD_NAME__CRS failure for PNP0c02
[12.574567] Pnp: PnPACPI:METHOD_NAME__CRS failure for PNP0c02


Kernel alive
Kernel direct mapping tables up to 100000000 @ 8000-d000
```

Figure 4: Linux boot-up messages before hang.

Linux. When we attempt booting up Linux with graphics passthrough, the boot up failed and hangs after the messages shown Figure 4.

To study the performance of the passed through graphics card, we ran OTS benchmarking applications, fillratetest [18] and 3DMark06 Basic v1.1.0 [19]. DirectX 9.0C December 2005 update was also installed. Our test system has the following specifications.

- Intel DQ45CB Motherboard (BIOS CBQ4510H.86A.0079.2009.0414.1340)
- Intel Core 2 Quad Q9300 2.5GHz Processor
- 4GB DDR2 800 (PC2 6400) RAM
- Samsung 2253BW Widescreen LCD (Native Resolution: 1680x1050)

3DMark06 generates scores using Shader Models. Shader Models are software instructions for programming the GPU. They are used for rendering and typically comprise the vertex shaders, geometry shaders and pixel shaders. The Shader Model scores provide a benchmark for the rendering capability of the GPU. The data files for 3DMark06 are stored on disk.

Fillratetest attempts to measure the video memory bandwidth and the fill rate. The video memory bandwidth is the amount of texture data that the graphics card is capable of reading from the video memory every second. The fill rate refers to the number of pixels that the video card is capable of updating per second. Fillratetest does not rely on any data files, and thus, there is no disk IO when it performs the tests.

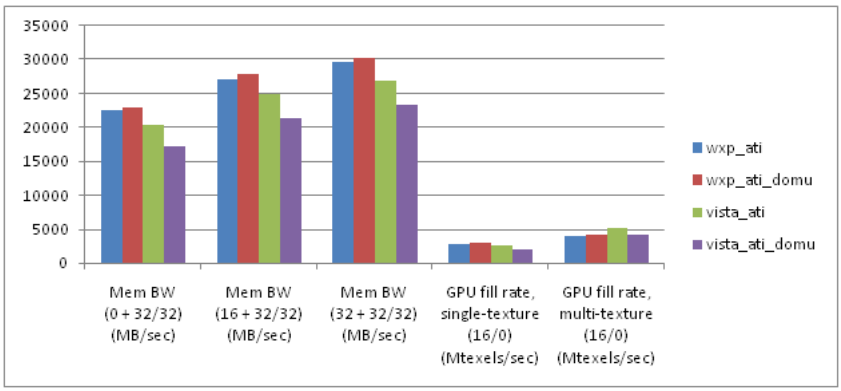

Figure 5: fillratetest results for ATI Radeon HD 2600 XT.

8 Results and Evaluation

We were able to achieve good results on the ATI graphics device, but not the others. We describe our observations here.

8.1 ATI Radeon HD 2600 XT

In the fillratetest, the ATI card performed reasonably well, as shown in Figure 5. However, we observe that in Vista64, the performance is slightly lower. This may be due to the driver implementation as a move to reduce IO space usage [20]. Due to the exhaustion of IO address space, Windows 2003 and Vista allow the use of memory space, which is in abundance in comparison to IO addresses available for PCI devices. The impact is even greater in DomU, since memory mappings have to be performed by Xen from the guest frames to the physical frames.

For the 3DMark06 scores, as shown in Figure 6, the ATI card's native and passthrough performances are comparable as can be seen from the Shader Model scores. However, DomU's CPU scores pales in comparison to the native CPU scores. This is expected as the number of time slices which DomU is allocated depends very much on Xen's scheduler. In addition, 3DMark06 requires reading of its data files from the disk. The large amounts of disk IO is expected to impact the performance. In contrast, fillratetest does not read any data files.

8.2 Intel GMA 4500

From our tests, the Intel IGD does not seem to be stable, even when set up natively. We summarize the results in Table 4. As a consequence, there is no basis for comparison.

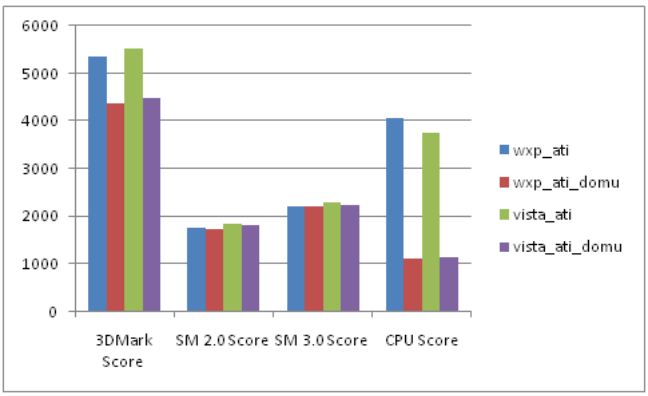

Figure 6: 3DMark06 scores for ATI Radeon HD 2600 XT.

	XP32	Vista64
Native	Ok. 3DMark06 crashed.	Cannot detect 3D Hardware.
Passthrough in DomU	Cannot detect 3D Hardware.	Ok.

Table 4: Result summary for Intel GMA 4500.

8.3 nVidia GeForce 9500 GT

For the nVidia card, graphics passthrough does not seem to work properly. When DomU is started, we could observe the screen being blanked out. Dom0 does not crash. More research is required to fully understand the reason, although we suspect this is due to vendor specific implementation of the graphics device.

8.4 Cirrus 5446

Cirrus GD5446 is used as the emulated graphics device if passthrough is not available. For both Windows XP and Vista64, we were not able to execute the benchmarking software. In all scenarios, the software returned failure to detect 3D hardware.

9 Limitations

Despite some successes, we observed several limitations during our experimentations, which we list in this section.

- When testing Intel GMA 4500, we noticed that Xen failed to load unless another graphics card was present. There were no error messages.

- When DomU (both XP and Vista) is started with Intel GMA 4500 passed through, the resolution is reduced to a maximum of 640x480 pixels. The color depth is 4 bits. However, when the Intel driver is removed, the resolution is increased to a maximum of 1920x1440 pixels. 32 bits color is possible. The benchmarking applications report that no supported 3D hardware is available in both scenarios.
- The HID control is not available until the OS has boot into the GUI login. This implies that if the Fail Safe selection menu appears, such as when the previous shutdown is corrupted, the user will not be able to manipulate the screen.
- Graphics passthrough currently does not work well for Linux DomU and nVidia GeForce 9500 GT.
- Due to the emulated PCI bus, the guest OS will be requested to install driver for the bus. However, currently there is no such driver available.

10 Conclusion

We discussed the necessary requirements to enable graphics device passthrough, and demonstrated that graphics card passthrough allows the graphics performance in DomU to be on par with native graphics adapters. The Intel IGD appeared to be unstable, even when run natively. The graphics passthrough technology also does not work for nVidia GeForce yet. However, unexpectedly, ATI Radeon GeForce performed well despite several limitations.

The ability to passthrough graphics devices to DomU greatly enhances the applicability of virtualization to many areas, such as video rendering. As graphics card manufacturers strive to research and develop graphics devices that support virtualization [21], it would only be a matter of time when this capability is made mainstream.

References

[1] Wikipedia. Hypervisor. Online, 2009.

[2] Muli Ben-Yehuda, Jimi Xenidis, and Michal Ostrowski. The Price of Safety: Evaluating IOMMU Performance. In *Proceedings of the Linux Symposium*, volume 1, 2007.

[3] Jean Guyader. [PATCH] Pass-through a graphic card. Patch, May 2008. `http://xen.markmail.org/download.xqy?id=x4ethv65thwrq3e6&number=1`.

[4] Patch Queue for IOEMU. Patch. `http://xenbits.xen.org/gitweb?p=xenclient/ioemu-pq.git;a=summary`.

[5] Patch Queue for Xen. Patch. http://xenbits.xen.org/gitweb?p=xenclient/xen-pq.git;a=summary.

[6] Yuri Schaffer. Xen VGA passthrough. Technical report, Univeriteit van Amsterdam, 2009.

[7] Wikipedia. PCI Configuration Space. Online, 2009.

[8] Ravi Budruk, Don Anderson, and Tom Shanley. *PCI Express System Architecture*. MindShare, Inc. Addison-Wesley Publishing, 1st edition, 2004.

[9] Intel Corporation, Microsoft Corporation, and Hewlett-Packard. PCI-SIG Engineering Change Notice: Function Level Reset (FLR). 2005.

[10] FLR support for pciback. Patch. http://xenbits.xen.org/gitweb?p=xenclient/linux-2.6.27-pq.git;a=blob_plain;f=master/pciback-flr;hb=HEAD.

[11] Citrix Systems. VTd HowTo. Online, May 2009.

[12] Jean Guyader. Experimental results for VGA passthrough. Online, June 2009.

[13] TechPower. Guide to Video BIOS Flashing. Online, May 2004.

[14] Julio Sanchez and Maria P. Canton. *The PC graphics handbook*. CRC Press, 1st edition, 2003.

[15] Richard F. Ferraro. *Programmer's Guide to the EGA and VGA Cards*. Addison-Wesley Publishing, 2nd edition, 1990.

[16] Intel. Intel 4 Series Chipset Family - For the Intel 82Q45, 82Q43, 82B43, 82G45, 82G43, 82G41 Graphics and Memory Controller Hub (GMCH) and the Intel 82P45, 82P43 Memory Controller Hub (MCH). Technical report, Intel, May 2009.

[17] Michael Larabel. Intel Releases IGD OpRegion Spec. Online, October 2008.

[18] Paul William Mcluckie Roberts. fillratetest (free video benchmark utility). Online, 2007.

[19] FutureMark Corporation. 3DMark06. Online.

[20] Microsoft. I/O Resource Usage Reduction. Online, April 2004.

[21] Derek Perez. NVIDIA SLI Multi-OS Empowers Worlds First Virtualized Graphics Workstation. Online, March 2009. http://www.nvidia.com/object/io_1238408514209.html.