NVIDIA CUDA TECHNOLOGY IN ADOBE PREMIERE PRO AND THE NEW MERCURY PLAYBACK ENGINE

Joe Stam
Sr. Applications Engineer, NVIDIA

Whitepaper

NVIDIA CUDA TECHNOLOGY IN ADOBE PREMIERE PRO AND THE NEW MERCURY PLAYBACK ENGINE

Professional video editing demands enormous computational power, especially when you have HD video firmly established as the de facto standard and the increasing popularity of 4K and other higher-resolution formats. Simply manipulating high resolution footage is a laborious task, even with the most powerful workstations; editors apply changes and then wait patiently for renders to complete. Full quality previews of complicated timelines are only possibly after a lengthy export and encode process. In the past, users demanding more interactive control or those with high productivity requirements had no choice but to spend tens of thousands of dollars on custom hardware solutions.

Adobe Premiere Pro CS5 takes a bold step forward with its new Mercury Playback Engine by leveraging the massive parallel computational power of a Graphics Processor unit (GPU). With a quantum leap in performance afforded by the GPU, HD video editing of complex projects suddenly becomes interactive. Most high-end workstations today feature powerful GPUs, brining to all Premiere users an editing experience previously available only to professionals with the highest production budgets.

The Mercury Playback Engine was an ambitious project and required close collaboration between Adobe and NVIDIA®, the GPU manufacturer. This whitepaper is intended for those readers interested in the technical background of GPU computing, and specifically how the parallel processing capability of NVIDIA GPUs provides such a remarkable advancement for video processing. Although the technical detail presented herein is certainly not required for effective use of the Mercury Playback Engine, it can serve to better educate hardware configuration and purchasing decision and to satisfy the technical curiosity of those hardware-savvy users.

WHAT IS A GPU?

Invented at the end of the 1990's, GPUs contained custom digital processing circuitry needed to accelerate the rendering of 3D graphics for Computer Aided Design, 3D content creation, and gaming. GPUs perform all the complicated mathematical transforms necessary to manipulate the points, lines, and triangles that make up three-dimensional graphic objects and compute how those primitives are to be drawn on a screen. Standard programming interfaces such as OpenGL and Microsoft's DirectX allow programmers to specify the geometry and various appearance properties while the GPU takes care of rendering these objects to the screen. As you can imagine, manipulating vertices and pixels is a highly parallel task so it is no surprise that GPUs contain many parallel processors.

GPUs have become increasingly more flexible and programmable. Initially this increased programmability facilitated more creative visual effects; such as simulated materials and advanced lighting. However, ambitious programmers, attracted by the GPU's massive parallelism and processing capabilities, began to develop applications outside of traditional graphics. Inspired by this trend, modern NVIDIA GPUs now contain a massive array of very general-purpose programmable processor cores that are great for graphics, but are equally applicable to video processing and other parallel tasks.

To facilitate flexible programming of applications beyond graphics, NVIDIA developed CUDA®., CUDA is a hardware and software framework designed for GPU computing. NVIDIA GPUs now contain the hardware components necessary to flexibly control and assign workloads to hundreds of processor cores. CUDA also provides device drivers for many modern operating systems including Microsoft Windows XP, Vista, Windows 7, Mac OS X, and many different versions of Linux.

CUDA provides compilers to use common programming languages to write software for the GPU rather than the unique specialty languages previously required for graphics programming. CUDA currently supports programming in C, C++, Fortran, OpenCL, Direct Compute, Python, Perl and Java. This list continues to grow with offerings from NVIDIA and third parties. Finally, many CUDA libraries provide various mathematical and image processing functions, allowing the developers to benefit from GPU acceleration without needing to recreate them from scratch. Figure 1 shows a representation of the CUDA framework.

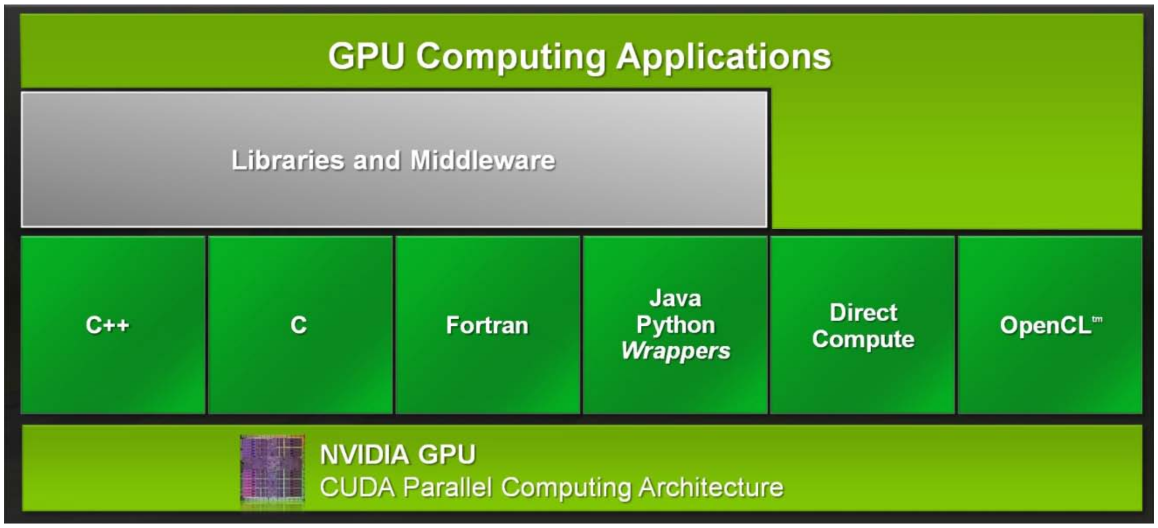

Figure 1. The CUDA Framework

GPU COMPUTING

There is a massive change underway in personal computer architecture. Have you noticed that your CPU clock speed today is probably about the same as your CPU clock speed five years ago? Sometimes physics gets in the way, and physics abruptly halted the trend of rapidly increasing CPU clock rates which we have all enjoyed for the past 25 years. If you cannot go faster, you must go wider; and thus the move toward multi-core CPUs and other architectural innovations to allow CPUs to perform more work for each clock cycle. Unfortunately, in order to take advantage of a multi-core CPU, software developers need to rewrite their applications to divide the workload across the parallel cores. Indeed Adobe engineers have worked extensively to allow Premiere Pro CS5 to take advantage of all available CPU cores.

Today's CPUs are brilliant devices, designed to run a phenomenally diverse set of applications, switch between many different tasks, run an operating system, and communicate with peripherals. CPUs are also designed to minimize the time it takes to perform a computation; they are latency-optimized processors. This is a critically important feature for many tasks that require the results of one computation as an input to a subsequent computation. One way the CPU achieves this is to have large on-chip cache memory. Cache memory is many times faster than a computer's standard memory (RAM), and thus the latency to read a value from cache, perform computations, and write them back to cache is very low. The price of very low latency is that frequently over half of a CPU's chip area is consumed by cache and the logic necessary to pre-fetch data from external RAM and write it back.

CPUs have evolved over many years to include the right balance of computation cores, cache, I/O, and other resources to provide a good experience for a wide range of popular applications. However, the CPU does fall far short when it comes to computational-intensive applications that have a large amount of parallelism. Does that sound like a small niche? Not really, virtually any type of *visual* computing fits that description. Everyone already expects great graphics on a new PC, while digital imaging and video continues to be an area of explosive growth. Many scientific and industrial applications are also massively parallel.

We know GPUs have a lot of parallel processors, but let's look at another key distinction related to video and imaging: When a CPU core processes an image it processes one pixel, then the next, then the next and so on. Since the CPU is latency-optimized, each individual pixel is processed extremely fast (if the pixel value is in the CPU's cache). So, what is more important; the time it takes to process *each pixel*, or the time it takes to process the *entire image*? It is by far the time it takes to process the entire image of course, and it is the GPU architecture that takes this into account.

The GPU is a throughput processor designed to process *huge* amounts of data (for example, pixels, 3D vertices, etc.) in the *least* overall time. Pixels are independent, and it does not matter if each pixel is processed slower as long as the total computation time for all pixels is less. Reading and writing data to external RAM is slow and frequently takes longer than performing computations on that data. GPUs work on many pixels simultaneously, constantly juggling between pixels which are waiting to be read or written to RAM and pixels that are ready to be processed. In fact, a single GPU core can be working on as many as 48 pixels simultaneously. The latest NVIDIA GPUs contain up to 480 cores, which means processing up to 23,040 pixels in parallel!

What's more, the GPU uses very little cache memory. This translates into more of the silicon chip area can be dedicated to processor cores. Electrical power is not wasted to store values in cache, more of this power goes to processing pixels, which makes the GPU much more electrically efficient. This may seem counterintuitive to those who have installed a GPU in their workstations and noticed the additional power connections and heat generation. But despite appearances, the GPU saves energy by performing computations much quicker and thus reducing total power consumed.

Another characteristic of GPU computing is the massive redundancy apparent in applications like graphics and video. Many video operations perform nearly the exact same computation on every pixel. There may be some occasional differences with image borders and masks, and some of the values used in computations may vary, but for most pixels the formula remains identical. The design of the GPU exploits this redundancy heavily. It is much less work and takes much less circuitry to apply the same computation to 32 pixels at once than it takes to apply the computation 32 times sequentially to 32-pixels. Once again, the GPU wins for efficiency. For the technically minded, this is referred to as a SIMT (single-instruction-multiple-thread) architecture; which means apply the same Instruction (computation) to Multiple Threads (pixels) simultaneously.

CPUs are awesome at serial, latency-sensitive operations, and do an excellent job of running many of today's operating systems and common desktop applications. GPUs are awesome at computationally intensive tasks on large amounts of data, like those in graphics and video processing. So let's use both!

That is the exact revolution occurring in personal computer architecture today. Build a workstation with a CPU that can efficiently handle all the desktop applications you need, and then add a powerful GPU which acts as a co-processor to handle the computationally-intensive tasks. The CPU and GPU communicate through the ultra-fast PCIe bus. The CPU manages the operating system and user interaction, and hands off the computational tasks to the GPU. Technically this is called *heterogeneous computing*. (Figure 2 shows a representation of heterogeneous computing.) More simply, the CPU serves as a central nervous system for the application handling command, control and communication; while the GPU provides the computation muscle to the heavy lifting.

The result: an optimized, efficient and visually stunning system that can fly through video and imaging operations with ease.

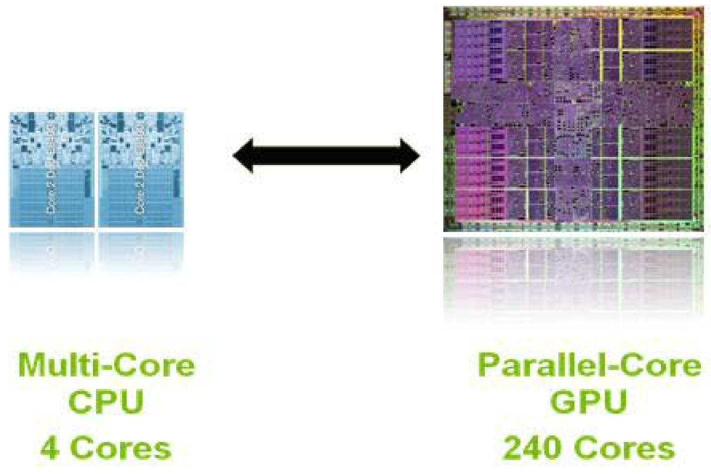

Figure 2.   Heterogeneous Computing

WHY USE GPUS FOR ADOBE PREMIERE PRO?

Let's take the information previously discussed and apply it to a specific example for video processing. We will discuss a few simple concepts here, no fancy math. Each frame of a 1080p HD video contains 1920 x 1080 pixels, or 2,073,600 pixels total. As a basic example, consider a contrast and brightness adjustment on a video frame:
For each pixel, the source value must be multiplied by a contrast adjustment factor and then added to a brightness offset factor. Sounds simple, but it is still a bit of math when applied to an entire frame. And guess what, it is also a massively parallel problem. Each pixel's new value can be computed independent of every other pixel. What do you think is faster, having a couple cores processing two million pixels sequentially or having hundreds of cores working in parallel? At the most elementary level that's why GPUs are wonderful for video processing.

Pixels can have a variety of formats used to encode their luminance and color (with smart people hotly debating the merits of various formats). While video is stored in a variety of formats, the GPU render in the Mercury Playback Engine uses a BGRA format for all internal processing, that means each pixel contains a value for the Blue, Green, Red and color components along with an *Alpha* component representing the transparency of the pixel for use during blending and compositing operations.

The *Bit Depth* refers to how many digital bits are used to represent each component, for example an 8-bit/channel representation allows up to 2^8 or 255 different values for each color channel, 16-bits /channel allows 2^{16} or 65535 different values.

An even more precise way to represent each color value is to use a 32-bit *floating-point* number, with a nearly continuous range of values far exceeding the limits of color discrimination. Many experienced professionals are able to notice banding and other limitations of an 8-bit/channel representation, but a 10- or 12-bit/channel image certainly captures all the colors anyone could hope to distinguish, so why would anyone need a 32-bit floating point pixel? Video editing often requires the concatenation of multiple effects. Any error or rounding in one effect may be negligible, but when the error from many effects is accumulated over a pipeline, there can be degradation. A histogram of the image may show banding, where the range of luminance distribution contains gaps. A 32-bit per color channel eliminates this worry altogether.

Dynamic range is yet another reason to prefer using a floating point format. A 16-bit, 65535-to-1 range sounds like a lot, but that is actually paltry compared to the human visual system's ability to see over a 1,000,000-to-1 dynamic range. Outside of some specialized applications, today's video cameras cannot achieve much beyond 12 or 14 bits of dynamic range (*HiDy* techniques with digital still cameras are permitting acquisition of images nearing or exceeding the capabilities of a human). Similarly, video output devices do not achieve this range, so why is *floating-point* important? Because it gives the editor the flexibility to perform various adjustments to modify tonal range, integrate sources shot under different lighting conditions, target different output devices, and provide plenty of working headroom as the quality of input video continues to improve.

So what was the point to this diversion to talk about 32-bit floating point? Because the GPU loves it and the GPU-rendering Mercury Playback Engine uses it exclusively. NVIDIA GPUs are native floating point devices and perform computations on 32-bit floating point numbers as fast as on 8-bit numbers. There is no longer a need to choose between a fast 8-bit mode or a slow 32-bit mode. It is all 32-bit, all the time, and it is always fast!

HD images are big—33 MB for a full float 32-bit uncompressed 1080p frame. These images take a lot of memory, and that memory must be fast. Here is yet another critical feature of GPUs which make them perfect for video editing. GPUs have very fast dedicated video RAM, which is needed to keep all those cores filled with data to process. Modern GPUs can move over 100 Gigabytes/second of data to and from their memory—that is almost 10X the speed of a CPU!

Note: Unlike the memory on your computer's motherboard, GPU memory cannot be upgraded or replaced. That is unfortunate, but the connectors needed to make memory replaceable would slow down the electrical signals. When speed is king, it must be soldered down permanently

As mentioned previously, the CPU uses cache to overcome the latency of reading and writing to RAM. It is an effective approach when working with small amounts of data, but there is no way to fit many HD images in cache. Furthermore, each square millimeter of CPU chip area which is spent on cache memory means less area for computation cores. The GPU's ability to keep each core busy with many pixels to process, combined with the massive bandwidth to video RAM allows high throughput without using valuable silicon area on cache.

Finally, let's turn back to *heterogeneous computing* (using a CPU and GPU together, each doing the job it is best at). Premiere Pro is truly the perfect heterogeneous application. As stated, the Mercury Playback Engine makes great use of the GPU to compute complex video effects. But there is still a lot for the CPU to do: it needs to handle user input, figure out which video clips need to be loaded from disk, decode a huge variety of source video formats, and parse the timeline to determine which effects need to be rendered at every frame. The CPU acts as the master scheduler for Premiere Pro, manages video data movement to the GPU and kicks off rendering tasks on the GPU. That is actually quite a bit to do, and now that all the intensive rendering is moved to the GPU there is plenty of free time for the CPU to concentrate on its' own workload. The result is an optimized and responsive video editing experience!

INSIDE THE GPU

This final section seeks to satisfy the most technically-curious reader, and provide a high-level overview of a workstation configuration and details of the GPUs internal architecture. Figure 3 illustrates the major components of a workstation based upon the latest Intel architecture found on Core i7-based systems. The CPU connects directly to its' DRAM memory, commonly DDR3 memory, with a bandwidth ranging from 10-20 GB/second. The CPU also connects to a peripheral interface chip such as the X58 Express I/O hub, through a proprietarily Intel Quick Path Interconnect Bus (this component is sometimes called the motherboard's *chipset* for historical reasons when many discrete chips were needed). The X58 chip implements various interfaces to peripherals such as USB, SATA, networking, and PCI Express 2.0 (PCIe). The PCIe bus is the main interface to the GPU, with the PCIe 2.0 16-lane interface on NVIDIA's GPU delivering a sustainable 5.5 – 6.0 GB/second of bandwidth. The GPU has its own dedicated ultra-fast memory, either GDDR3 or GDDR5 memory with a bandwidth of up to 150 GB/second. The other output of the GPU is, of course, display connections for various monitor types.

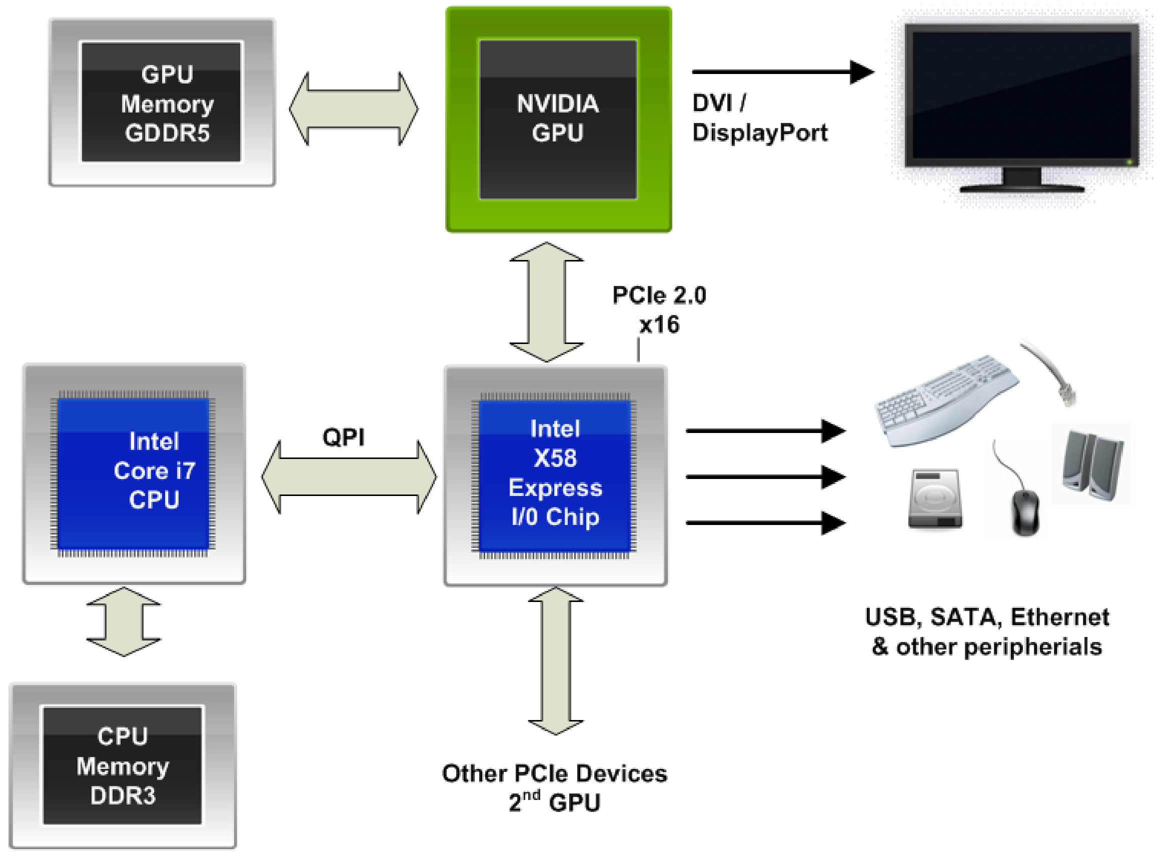
GPU Memory GDDR5
NVIDIA GPU
DVI / DisplayPort
PCIe 2.0 x16
Intel Core i7 CPU
QPI
Intel X58 Express I/0 Chip
USB, SATA, Ethernet & other peripherials
CPU Memory DDR3
Other PCIe Devices 2nd GPU

Figure 3. Modern Workstation Architecture

Now, let's examine Figure 3 again but this time in the context of *video data flow*. Source video data typically resides on disk, by far the slowest item in the chain even with high speed RAID arrays. The CPU reads data through the I/O hub from disk into its memory where, if compressed, the CPU then decodes the video into uncompressed frames (alternatively some popular formats may also be decoded using the GPU's video engine). The GPU then performs a *copy* which pulls the data from CPU memory, through the I/O hub and into GPU memory. Once the data is in GPU memory, the rendering computations executed on the GPU have extremely high bandwidth access to the data. Preview of rendered video output in GPU memory on displays connected to the graphics card is naturally very efficient. Depending on the output format desired, final results may be encoded on the GPU and then transferred back to CPU memory for final disk storage or encoded on the CPU and then stored.

Looking at the inner workings of the GPU, Figure 4 show a high-level block diagram of NVIDIA's newest generation of GPU, code-named *Fermi*. While the most obvious feature of the diagram is the large number of cores, first consider some of the supporting components. A host interface, connected to the PCIe bus, communicates with the host CPU and receives commands and data. An important part of the host interface, the Copy Engine, performs Direct Memory Access (DMA) transfers of data between CPU memory and GPU memory. The copy engine operates independently, and may perform these data transfers in the background with both the CPU and GPU cores busy doing other computations. The newest Quadro GPU is based on Fermi architecture and contains two copy engines, facilitating full speed data transfer both to and from CPU memory simultaneously.

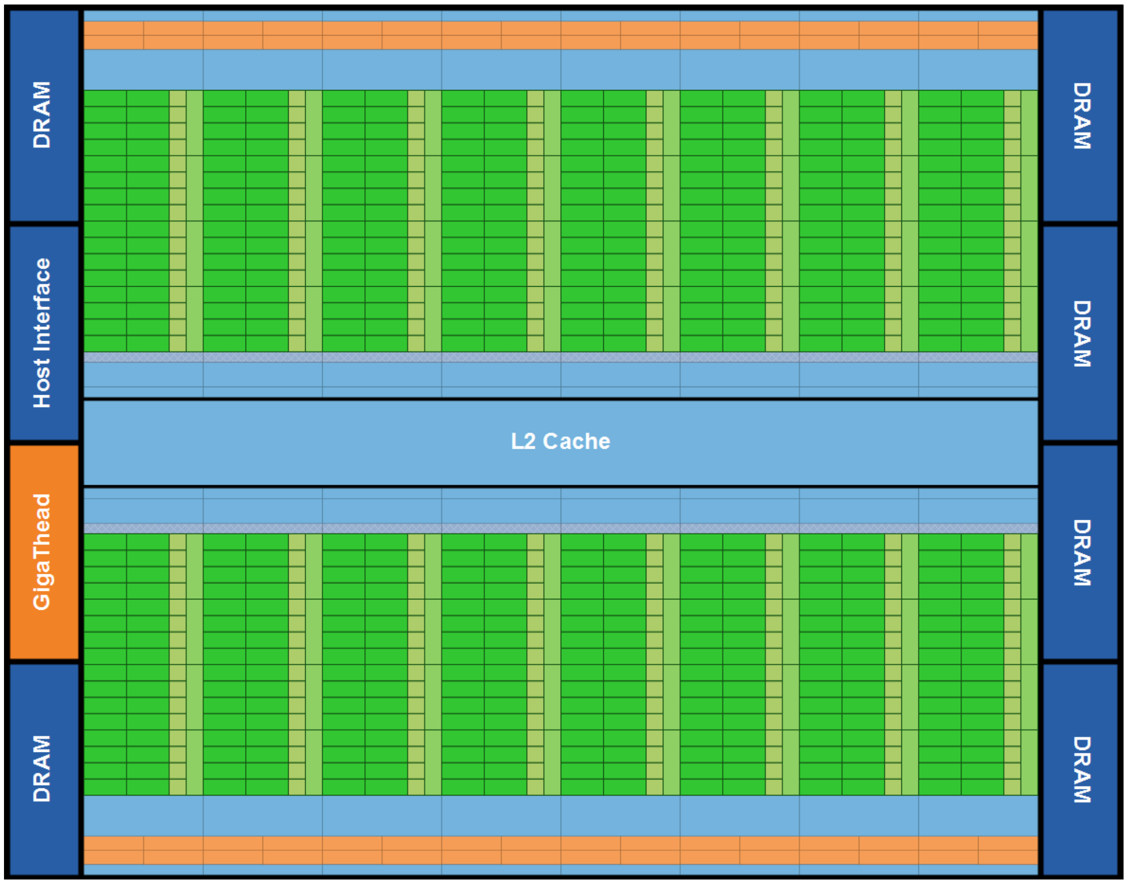

Figure 4. NVIDIA Fermi GPU Architecture

GigaThread

A thread scheduler, called the GigaThread™ engine, creates threads in hardware and automatically distributes work to the hundreds of cores. GPUs are very different than CPUs in this regards, as CPUs rely on the operating system to create threads and schedule work on each CPU core.

Memory Controller

A Memory Controller with several memory interface ports attaches to the external GPU memory (DRAM), and provide data to the cores through a common central L2 cache. Also, in contrast to a CPU, the cache is not terribly large and serves more as a buffer to group memory reads and writes rather than a large data storage. As discussed previously, the GPU shuffles between a large concurrent workload rather than rely upon huge caches to hide memory latency. Error Correcting Code (ECC) memory is now available on Quadro GPUs. Professional users have long demanded ECC CPU memory on workstations to prevent occasional bit errors and increase reliability and data confidence. Professional grade GPUs from NVIDIA now provide this same level of memory reliability

Streaming Multiprocessors

The GPU contains many *Streaming Multiprocessors* or *SMs,* shown with more detail in Figure 5. Each SM contains 32 cores, each of which can operate on 32-bit floating point or integer values (although not commonly used for video processing, 64-bit *double precision* math is also available). The largest current Fermi GPU contains 15 SMs for a total of 480 cores. Special Function Units (SFUs) on each SM implement fast, sophisticated mathematical operations such as trigonometry and square roots.

The SM also contains 64-kilobytes of *Shared Memory* or *SMEM*. Think of SMEM as extremely fast scratch-pad memory that stores temporary data accessible by all 32 cores in the SM. It is extraordinarily useful for video operations; a small tile of an input video frame loaded into the SMEM and then processed by all the cores. Many video operations produce an output pixel value which is some function of the corresponding pixel in the input frame and other pixel values in the local neighborhood. Blurring is a perfect example, where the output results from a weighted average of pixels in the surrounding neighborhood. Storing a tile of the source frame in SMEM eliminates multiple reads to the external memory, which greatly increases the overall performance.

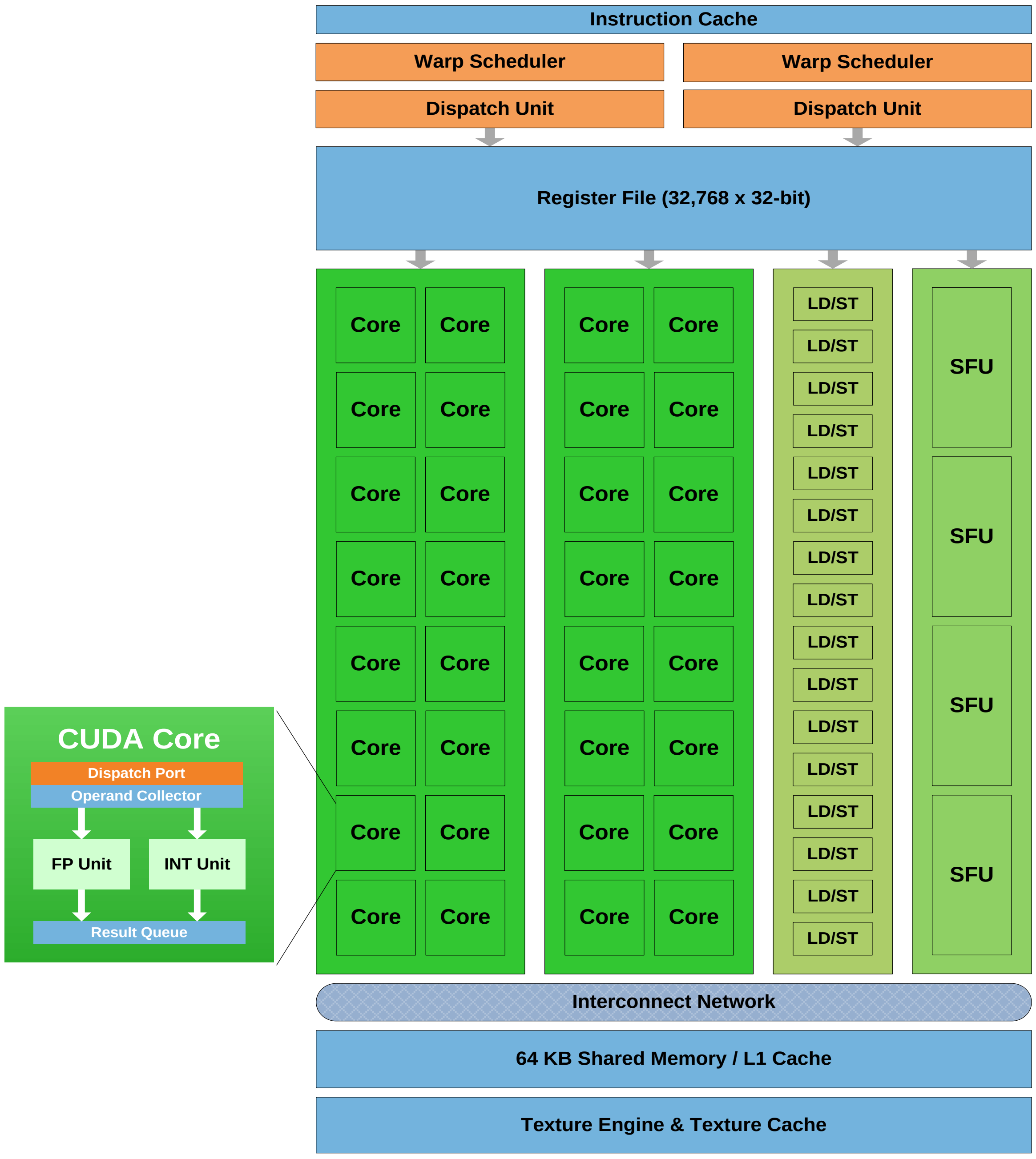

Figure 5. Fermi Streaming Multiprocessor

Texture Engines

Texture Engines in each SM provide another fast and useful way to read image data. Texturing is a ubiquitous operation in computer graphics, in which a source image is wrapped around 3D geometry to provide a particular surface appearance. Projecting or warping the image in 3D space requires very irregular reads to the GPUs memory. Texture hardware optimizes these irregular reads and contains hardware logic to compute memory addresses and clamp out-of-bounds reads, and even interpolate values between pixels. Texture hardware accelerates many of the effects in the Mercury Playback Engine, especially those which involve spatial transformation of the image such as rotations, scales and translations.

Parallel programming has long been a niche and very sophisticated discipline of computer science. Programming tools were scarce and difficult to master. NVIDIA GPU hardware provides a high volume of massively parallel architecture available on every desktop and NVIDIA CUDA provides the programming framework to allow mainstream application developers to easily produce parallel applications and unleash the power of this hardware. The CUDA Parallel Computing Architecture is a combination of hardware features and device driver software for creating and executing parallel programs. Developers use the free CUDA toolkit from NVIDIA (optionally in combination with 3rd party tools) to write code using many popular languages. The application communicates with the GPU through the device driver to transfer data and execute the GPU code. The end user simply needs to have an NVIDIA GPU with the appropriate driver installed.

CONCLUSION

Within the last decade, the GPU has totally revolutionized the personal computing experience, bringing rich, interactive 3D graphics to every desktop. Now CUDA has unleashed the GPUs power for general purpose parallel computing, and promises to revolutionize the computing experience once again. Video editors will be among the earliest beneficiaries of this technology as Adobe introduces the *Mercury Playback Engine* in Premiere Pro CS5. Before long, all video and imaging tasks are likely to migrate towards using the GPU by default, just as all real-time graphics applications do today.

A modern optimized workstation contains both a CPU and a powerful GPU, with the CPU handling most conventional tasks and the GPU dedicated towards computationally intensive throughput tasks. This heterogeneous model provides the highest performance, and also the most energy efficient approach to computing.

Together Adobe and NVIDIA have revolutionized the video processing experience. By embracing GPU computing early, Premiere Pro is catapulted into the lead as the highest performance video editing solution available without the need for expensive proprietary hardware.

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ROVI Compliance Statement

NVIDIA Products that are ROVI-enabled can only be sold or distributed to buyers with a valid and existing authorization from ROVI to purchase and incorporate the device into buyer's products.

This device is protected by U.S. patent numbers 6,516,132; 5,583,936; 6,836,549; 7,050,698; and 7,492,896 and other intellectual property rights. The use of ROVI Corporation's copy protection technology in the device must be authorized by ROVI Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by ROVI Corporation. Reverse engineering or disassembly is prohibited.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, CUDA, Fermi, Quadro, and GigaThread are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2010 NVIDIA Corporation. All rights reserved.