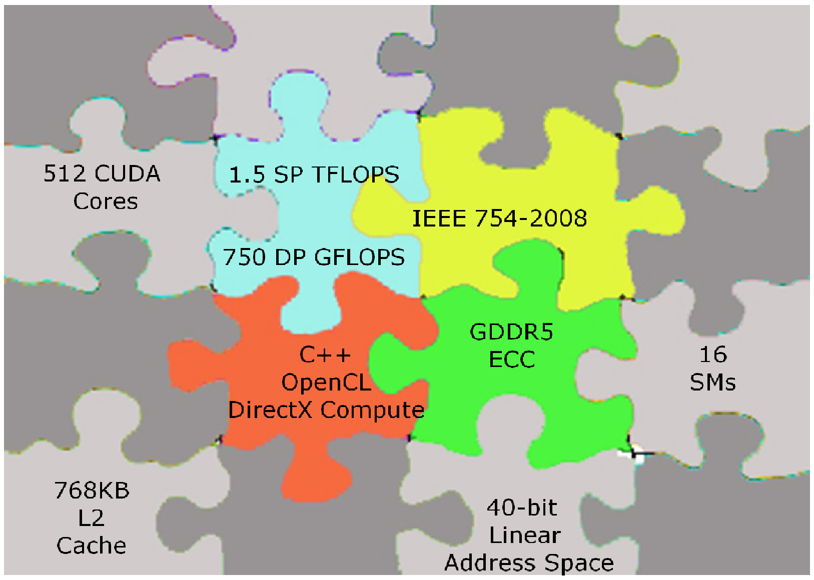
NVIDIA Solves the GPU Computing Puzzle



Nathan Brookwood
Research Fellow
Nathan@Insight64.com

September 2009

An Insight 64 White Paper Sponsored by NVIDIA

Anyone who has ever assembled a jigsaw puzzle knows the drill. You dump all the pieces on the table, sort them by shape, color or pattern, and begin the tedious process of fitting them together. Small clusters emerge, and then merge; only a few holes remain. As the solution develops, the placement of the remaining pieces becomes more and more obvious. Soon you have completed the task. You take a few moments to savor the image you struggled to assemble. Then you put all the pieces back in the box for another day or another solver.

Over the past six years, NVIDIA's design team has tackled three successive GPU computing puzzles. Their first design, the G80-based GeForce 8800 introduced in November 2006, operated at a peak rate of more than 500 GFLOPS. No mean feat, considering that the fastest general purpose CPUs at the time peaked at less than 20 GFLOPS. The G80 architecture replaced the separate vertex and pixel processors found in earlier NVIDIA GPUs with a unified compute processor that could be programmed in C. This change, along with a few other NVIDIA innovations, enabled mere mortals to utilize the GPU's enormous floating point computing capacity in non-graphical applications, and initiated the era of GPU computing. When they looked at the GPU compute processor they had created, NVIDIA's designers recognized that although they had used all 681 million transistors that came in the box, their puzzle still had a few gaps. They put the pieces they had assembled back in the box, and set out to build an even more ambitious device.

In 2008, NVIDIA introduced the GT200-based GeForce GTX 280, its second generation GPU computing design. This puzzle contained 1.5 billion pieces. (And you thought that thousand piece puzzle you did last summer was tough!) The GT200 delivered almost twice the performance of the earlier G80 and operated at a peak rate of over 900 GFLOPS. (A later tweak pushed the chip up to a teraflop per second.) The GT200 also supported double precision (DP) floating point arithmetic, a feature the G80 lacked entirely, and one that made it especially useful to scientists and engineers who need more precision than 32-bit floating point arithmetic can deliver. The GT200's prodigious computational horsepower enabled it to address a broad array of high performance applications at a fraction of the cost of more conventional approaches. NVIDIA's designers and engineers savored the technology they had created, but soon realized that their GT200 lacked a few of the elements they needed to complete their GPU computing puzzle. Once again they put the pieces they had assembled back in the box, and set out to build an even more ambitious device.

With the impending launch of its next generation CUDA architecture, code named Fermi, NVIDIA will finally have assembled all the pieces it needs to solve its GPU Computing puzzle. This time around, that puzzle contains three billion pieces, neatly arranged in 512 computing cores. Those cores, to be sure, deliver dramatically more performance than NVIDIA's prior GPUs. But more importantly from Insight 64's perspective, Fermi fills in the missing elements that precluded the use of NVIDIA's earlier GPUs for serious high performance computing tasks. New L1 and L2 caches boost performance and allow Fermi to tackle problems that stymied earlier GPUs. Error detection and correction on key system elements, including on-chip caches and off-chip memory, eliminates the possibility that soft errors can invalidate calculations. A new unified 40-bit address space allows programs to deal with data in private, shared and global address spaces in a consistent manner, and thus allows software written in C and C++ to be easily adapted for execution on Fermi-based systems.

It is no accident that Fermi's arrival coincides with the emergence of two industry standards specifically attuned to GPU computing requirements. Apple and NVIDIA drove the first new standard, OpenCL (Open Computing Language), but now key industry leaders, including AMD, IBM, and Intel have endorsed it as well. OpenCL gives software developers a consistent programming environment in which to write portable parallel code for heterogeneous systems. Microsoft and NVIDIA drove the second new standard, DirectCompute. This new API, an extension to the DirectX set of 3-D APIs for Windows 7 (and retrofitted for Vista), allows Windows-based applications to access the power of GPU computing to accelerate program calculations in a vendor-independent manner.

The convergence of new, fast GPUs optimized for computation as well as 3-D graphics acceleration and vendor-neutral, industry-standard software development tools marks the real beginning of the GPU computing era. Gentlemen, start your GPU computing engines. In the remainder of this document, Insight 64 will review the history of GPU computing, examine some of the key features NVIDIA designed into Fermi to enable this new era, and predict where this new capability will take the market.

A Brief History of GPU Computing

We live in a three dimensional world. For more than three decades, computer architects and engineers have toiled to endow the systems they design with the ability to recreate real-world 3-D experiences on two dimensional screens attached to computers. The manipulation of a 3-D model inside a computer requires a huge number of mathematical calculations, and the need to update the 2-D view of the object in real time (at least 60 times a second) means these calculations must occur very rapidly, even by the standards of very fast general purpose computer processors (CPUs). To address these unique 3-D computational requirements, engineers developed specialized devices known as Graphics Processing Units (GPUs) with prodigious number-crunching capabilities, optimized (not surprisingly) to handle the floating-point matrix math used to manipulate and render objects inside the computer's 3-D subsystem. This 3-D technology has fueled the growth of the computer game industry and has given scientists the ability to visualize the insides of molecules and atomic structures. As Martha Stewart would say, "It's a good thing."

As GPUs continued to bulk up their number-crunching capabilities in the pursuit of increased 3-D realism, a few insightful users saw an opportunity to tap the GPU's talent for rapid calculations to accelerate the non-graphical aspects of their software codes. These GPU computing pioneers needed expertise in both graphics and programming, since the GPUs of the day had been highly tuned for the limited set of operations needed to accelerate 3-D imaging. Early users were forced to map their general purpose algorithms into a series of shapes and surfaces in order to fool the GPU into thinking it was manipulating 3-D objects. A painful process, but one that yielded a big boost in performance at a relatively low (hardware) cost.

Once GPU suppliers like NVIDIA realized what these pioneers were doing with their chips, they took steps to simplify the use of their GPUs for computational, as opposed to graphical applications. To make this work, they needed to do far more than create a few new compute-oriented APIs; they had to restructure the overall architecture of their devices, replacing the specialized vertex and pixel pipelines that had characterized earlier designs with a single unified processor that could be programmed to handle pixels, vertices, textures and even non-graphical calculations. They provided libraries that allowed programmers writing in C to access GPU features without needing to learn a new language or to manage the vector registers used in traditional GPU programming models. NVIDIA called its initial GPU computing environment CUDA[1] when it first appeared in 2006 with the launch of the GeForce 8800. The company launched its second generation GPU computing offering, the GTX 280 series, in 2008. It included a big boost in performance, 64-bit floating point support, and a broader set of software development tools to access these features. Graduate students around the world quickly adopted CUDA, and countless theses depended on results calculated on NVIDIA GPUs. A few brave end users incorporated GPU computing into proprietary applications focused on oil and gas exploration, finance, and medical imaging. ISVs that focus on rich media applications, including Adobe and Cyberlink, use CUDA to make their applications run faster on GPU-equipped systems.

Over the past two years, CUDA has evolved from a set of C extensions to a software and hardware abstraction layer that supports a host of programming languages and APIs. Now NVIDIA has added OpenCL and DirectCompute to CUDA's repertoire, making it even easier for developers to construct GPU-accelerated applications that run on NVIDIA GPUs. Microsoft added GPU computing support into its new Windows 7 offering, as has Apple with the new Snow Leopard release of its MacOS X. End users can expect to see new versions of Adobe products, including Flash, Photoshop and Premiere that run faster on GPU-enabled systems.

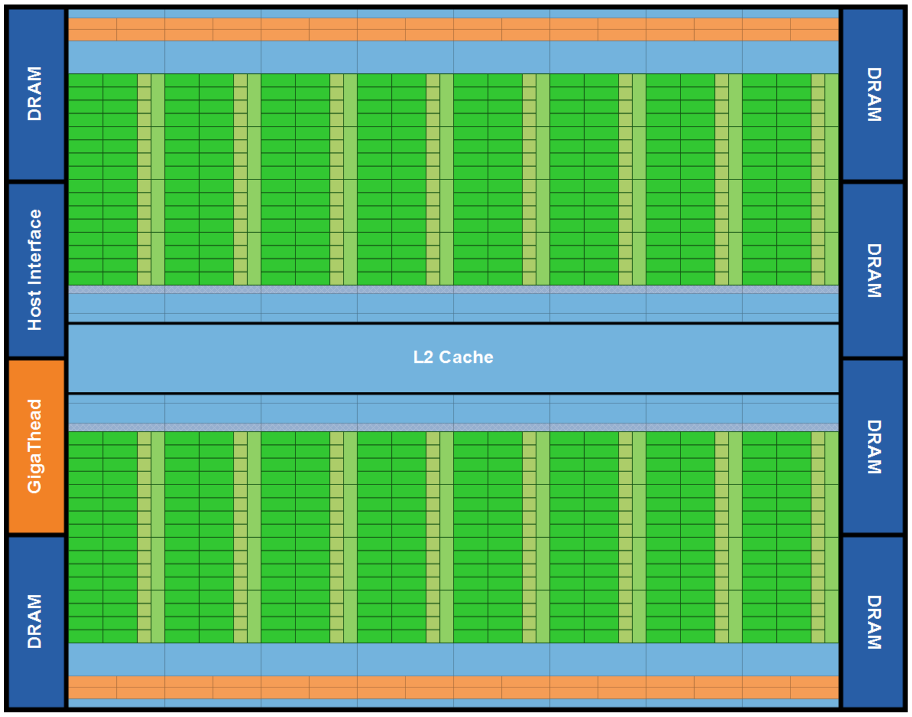
Along with these industry-standard software interfaces, NVIDIA has extended its CUDA architecture to support C and C++.These tools allow developers to construct applications that take full advantage of GPU acceleration on NVIDIA GPUs. These NVIDIA-specific tools simplify the task of adapting C++ applications for GPU computing, and thus enable a broader range of vertical applications to take advantage of this new technology. NVIDIA now gives software developers a choice: use generic GPU computing APIs and gain some acceleration, or use NVIDIA-specific APIs and optimize the performance potential of the hardware/software package.

[1] NVIDIA initially used CUDA as an acronym for "Compute Unified Device Architecture," but has since dropped the acronymic explanation and now uses CUDA as its umbrella brand for GPU computing products and features

A view of Fermi from 30,000 feet

Figure 1 illustrates the major elements of the Fermi GPU. It's a *big* chip, manufactured on TSMC's 40nm bulk silicon process, and contains three billion transistors, more than twice the number deployed in NVIDIA's earlier GTX280. In these days of trillion dollar deficits, three billion transistors may sound like a small number, but in terms of programmable chips, it's huge. It's more than Intel's upcoming Nehalem-EX (2.3 billion transistors) or IBM's upcoming Power7 (a measly 1.2 billion transistors).

Figure 1. Fermi's Major Elements



Source: NVIDIA

Fermi derives its computational power from sixteen Streaming Multiprocessor (SM) units, shown in green in Figure 1. Every SM contains 32 "CUDA cores," each capable of initiating one single precision floating point or integer operation per clock. One of the more common operations, floating point multiply and add, counts as two operations when estimating performance, resulting in a peak performance rate of 1024 operations per clock. NVIDIA hasn't released the final clock frequency for Fermi, but Insight 64 estimates it's likely to come in at around 1.5GHz. This would give Fermi a peak rate of 1.5 trillion floating point operations per second. To put this in perspective, just a few years ago, high performance computing systems sold at prices of $10,000 per TFLOPS. Prices have come down since then, and we doubt NVIDIA could get away with charging $15,000 for a chip, but even so, Fermi represents a real value to users who buy their systems by the teraflop.

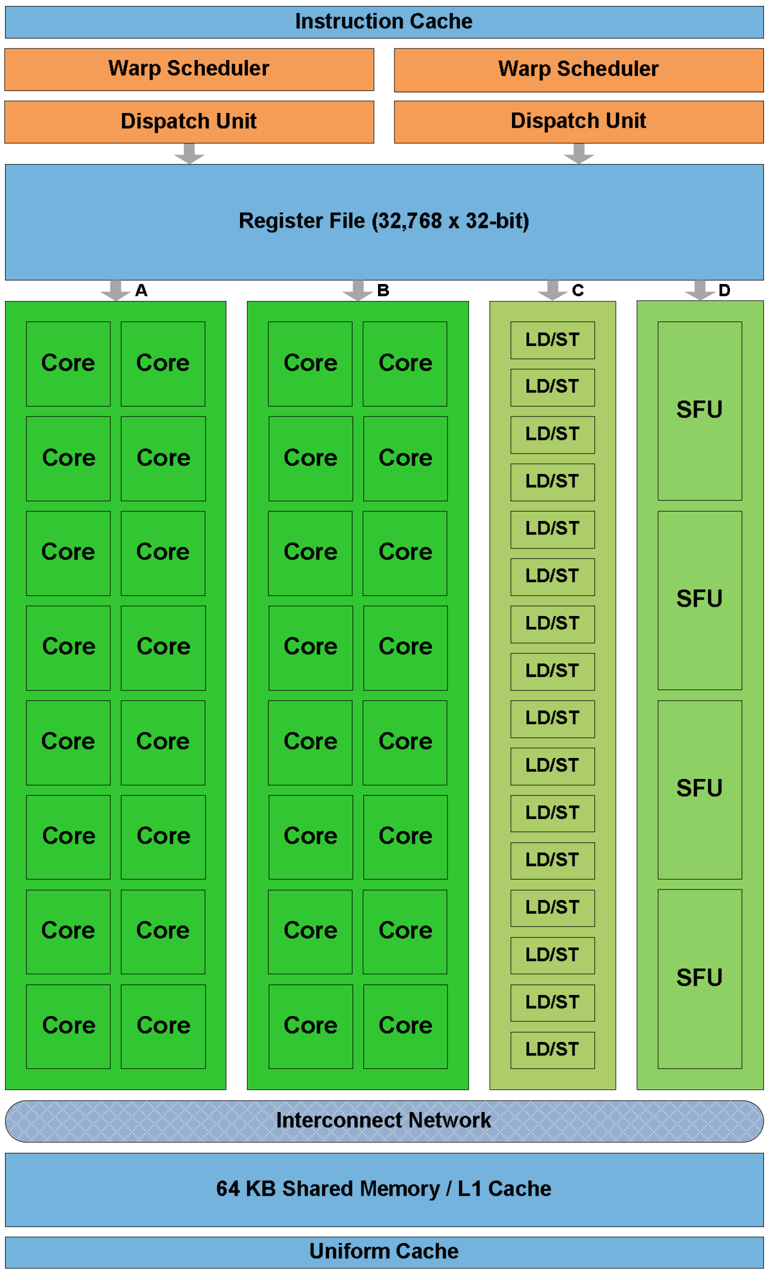
Fermi feeds its CUDA cores through six 64-bit GDDR5 memory controllers. Although NVIDIA hasn't finalized the spec on its GDDR5 DRAMs, Insight 64 guesses they will use 2GHz devices, giving Fermi an aggregate DRAM bandwidth of 192GB/second, about 30 percent more than the eight GDDR3 controllers on the earlier GeForce GTX 280. The move to GDDR5 reduces the width of the memory bus, which in turn should allow future versions of Fermi to shrink in size, since NVIDIA won't need as many pads to tie into its external DRAMs. Ordinarily one might want memory bandwidth to scale with peak computing rates, in order to "feed the beast," but the 768KB unified L2 cache, a feature not present at all on the GeForce GTX 280, should provide more than enough buffer to keep the CUDA cores well fed. Unlike the L1 caches embedded in each SM, where all data is private to the SM, and some data is private to individual threads on the SM, all the data in the L2 cache is shared globally and maintained in a coherent manner for all SMs.

Fermi communicates with its host system via a 16 lane PCI Express V2 interface that moves data to and fro at peak rates of 8GB/second. SMs have the ability to read and write individual data items stored in the host's main memory over this PCIe interface, as we shall discuss later, or the application can request that large blocks of data be streamed at full memory speeds between the GDDR5 DRAM (directly attached to Fermi) and the host's main memory via two GigaThread Streaming Data Transfer (SDT) engines included in the GigaThread block in Figure 1. Fermi can process a high speed SDT transfer for one kernel while it continues to execute other kernels. Clever software designers can exploit this feature to overlap these large block moves with calculations involving previously loaded blocks of data.

Last, but far from least, Fermi's GigaThread Hardware Thread Scheduler (HTS), a separate function within the GigaThread unit, manages the thousands of simultaneously active threads in various states of execution on the Fermi GPU at any given instant. This approach to scheduling threads in hardware, rather than software plays a key role in enabling Fermi's high level of performance. It allows software developers to create applications that can run on GPUs with an arbitrary number of SMs and threads; Fermi's scheduler distributes the work to the available SMs as the program executes, and manages the context switches between threads during execution.

Fermi's enhanced SMs do all the work

Figure 2. Fermi SM



Source: NVIDIA

Fermi contains sixteen "Streaming Multiprocessors" (SMs) that execute GPU programs and manipulate data. As shown in Figure 2, each SM contains 32 CUDA cores, 16 load/store units, four special function units (SFUs), a 64KB block of high speed on-chip memory that can be used either to cache data for individual threads and/or to share data among several threads; and an interface to the L2 cache shared among all sixteen SMs.

Each SM can issue instructions that consume any two of the four green execution columns (A, B, C, and D) shown in Figure 2. The SM can mix 16 operations from column A with 16 from column B or 16 from column C with four from column D, or any other combinations the program specifies. Columns A and B handle 32-bit integer and floating point, column C handles load/store operations, and column D handles "special functions," including square roots and transcendentals. 64-bit floating point consumes both columns A and B. This means an SM can issue up to 32 single-precision (32-bit) floating point operations or 16 double-precision (64-bit) floating point operations at a time. That's a lot of activity per clock. Now multiply that by 16 SMs, and repeat 1.5 billion times per second, and you begin to get a feeling for the phenomenal calculating ability of this chip.

NVIDIA has added several features to Fermi's SMs that make them far more powerful than the comparable units in the earlier GeForce GTX 280 family:

- First, NVIDIA extended the SM's addressing capability to 64 bits, almost always a good thing from Insight 64's perspective.[2] This move allows Fermi to address more local GDDR5 memory on the add-in board (their earlier 32-bit GPUs maxed out at 4GB, but Fermi cards will be available in 6GB configurations). It also allows Fermi to map virtually all the memory in the host system to which it is attached into its own address space, since these hosts moved long ago to memories larger than 32-bit pointers can accommodate.
- Second, NVIDIA added the ability for Fermi to directly address and cache items stored in the host's main memory, eliminating the need to move big blocks of data back and forth between host and GPU memory when only a few bytes of data are needed.
- Third, NVIDIA implemented the full IEEE 754-2008 specification, in hardware, for both 32- and 64-bit floating point operations. The new spec includes the so-called "fused multiply-add" capability that eliminates the intermediate truncation or rounding of the result of the multiply before performing the addition. This not only saves an operation, but enhances accuracy, since that (now eliminated) intermediate operation sometimes resulted in rounding errors. NVIDIA implemented the entire IEEE spec in hardware, including all the weird corner cases involving NaNs, subnormal numbers, and underflows, at full speed. Most IEEE implementations just throw up their hands and trap to software handlers when encountering these conditions, so NVIDIA's engineers deserve extra credit when they turn this project into their CompSci professors.
- Fourth, NVIDIA beefed up Fermi's double precision capability, giving all CUDA cores in each SM the ability to handle 64-bit operations, as contrasted with the more limited approach (just one 64-bit core per SM) it took in the GeForce GTX 280 line. This boosts Fermi's peak DP rating to 750 GFLOPS, far higher than any other CPU or GPU, even AMD's new Radeon 5800 series. The challenge here, as is always the case with "peak" ratings, is to come close to achieving this level of performance in actual use, a problem that is traditionally left to the user to resolve.
- Fifth, NVIDIA dramatically improved the performance of Fermi's atomic operations, sequences that read a value from memory, modify that value, and write it back. Most CPUs include a few of these atomic operations like test and set or compare and swap, but NVIDIA expanded the range of such instructions in its GT200 architecture, adding such niceties as append to queue for managing data structures. Such instructions make it easier for multiple threads in thread-rich environments like NVIDIA's GPUs to coordinate their activities and minimize the need to serialize big blocks of code, a huge performance killer in parallel multithreaded programs. The atomic implementation in GT200 incurred a high overhead, leading many developers to ignore this feature. Fermi speeds up the commonly used atomic instructions by factors of 5 to 20, a move that should encourage more developers to take advantage of this capability.

Thanks for the memories

Memory – physical, logical and virtual – plays a key role in determining how well GPUs (and CPUs) perform. Higher performance demands more memory capacity and more memory bandwidth. Beyond these raw physical parameters, memory organization matters, as does the view of memory from the executing program's perspective. In each of these regards, Fermi raises the bar in its pursuit of high computational performance.

Software developers love linear, non-segmented address spaces. A linear address space vastly simplifies the management of pointers into data structures, and the sharing of those data structures across multiple threads. Unfortunately, it's still not possible to integrate a multi gigabyte memory system on today's CPUs and GPUs. This forces hardware designers to combine a variety of on-chip

[2] Why did you think we called it "Insight 64?"

memory blocks with external DRAMs directly attached to the GPU and external system memory DRAMs. In its earlier GPU designs, NVIDIA forced programmers to deal separately with each of these different subsystems. With Fermi, it has cleaned up its act and given programmers a linear, non-segmented view of all memory accessible to the GPU. Programs address this memory using 64-bit virtual and physical addresses, although only the low-order 40 bits, enough to address a terabyte of memory, are implemented at this point. (Note to software developers: do not assume you can use the high-order bits in 64-bit pointers for status flags or whatever. A terabyte of main memory may seem like more than anyone will ever want or need, but before you know it systems with 32 or 64 terabytes will emerge. Some unlucky developer will curse you as she tries to unwind all those clever tricks you used to jam status bits into the high order "unused" bits in your memory pointers.) A page map determines which regions of virtual memory get mapped into a thread's private memory, which are shared across a block of threads, which are shared globally, which are mapped onto the GDDR5 memory attached to the GPU, and which are mapped onto the system memory. As each thread executes, Fermi automatically maps its memory references and routes them to the correct physical memory segment. Programmers around the world owe NVIDIA a huge debt of gratitude for this innovation.

Fermi uses GDDR5 for its main memory system, the first high-end NVIDIA GPU to adopt this technology. GDDR5 delivers higher bandwidth per pin than the more mature GDDR3 used in earlier NVIDIA designs, and this allows Fermi to use a narrower 384-bit memory interface instead of the wider 512-bit interface used in the GT200. Narrower buses use less power, and require fewer pins, simplifying PCB board layout. NVIDIA hasn't released the specs of the memory it will use with Fermi, but Insight 64 estimates it will go for the 2GHz parts; this would give Fermi a local memory bandwidth of 192GB/second, about a third more than it achieved with GDDR3 in the GT200. GDDR5 DRAMs transfer data over the memory bus at higher rates than GDDR3 chips, but use a new Error Detecting Code (EDC) to protect data integrity in transit. If the memory controller detects transmission errors on the memory bus, it simply retransmits the data. Technology like this has been a standard part of system buses for years, but now it's built into each DRAM chip.

Beyond the performance aspect of the memory system, NVIDIA also addresses the integrity of the data stored in those memories. Virtually all modern electronic devices are subject to problems induced by stray alpha particles and cosmic rays wandering too close to sensitive circuits, but large collections of bits stored in very fast memories are especially vulnerable, given the low amount of charge stored in each cell. Serious computer users typically add an error correcting code (ECC) option to the memory in their servers and workstations, since the ECC detects and even corrects occasional memory lapses in system hardware. This ECC feature should not be confused with the EDC feature noted above. EDC protects data in transit to and from memory. ECC protects data after it's been stored in the memory. Both are needed to ensure the high data integrity serious applications require. Optimists can rejoice that no GPU has ever reported a cosmically induced memory failure. Pessimists can worry that no GPU has ever included the ability to detect such errors, and thus would not recognize such an error, even if it occurred. When using GPUs for traditional graphic applications, errors like these hardly matter. No user would notice the small blip on the computer screen that might result from such an error. But when GPUs accelerate the performance of programs that analyze stresses in airframe designs or new buildings, or decide where to drill the next oil well, little mistakes can have big consequences. Fortunately, Fermi includes the same ECC capabilities usually found in servers and workstations. NVIDIA applies these ECC checks to all the major areas where it stores bits in the system – on-chip caches and register files, and off-chip GDDR5 memory. The consumer boards assembled around Fermi will likely omit this feature, so users planning to use Fermi for serious computing tasks should probably seek out the Tesla and Quadro boards that incorporate the chip if they want to sleep peacefully at night.

GPU computing software tools have come a long way

Developing software to run on parallel processors is at best a challenging task. Analysts generally regard the software industry's inability to take advantage of dual and quad-core x86 processors from Intel and AMD as a constraint on the growth of the PC industry, enabling users to buy less expensive systems, since they see little value in pricier systems with more cores than they can use.

NVIDIA tackled this challenge head on. It provided C language extensions that simplified the structure of multithreaded programs and hid most of the complexity inherent in the task in libraries that were just an application program call away for developers seeking the benefits of GPU computing. Countless CUDA success stories emerged, particularly from academia. Universities are filled with smart developers looking for inexpensive hardware solutions they can use to solve complex problems. Impressive as these stories were, they focused primarily on individual applications developed using the basic CUDA tools developed and maintained by NVIDIA. Few if any CUDA development tools emerged from organizations that earn their living by developing and marketing development platforms. There's a reason for this; platform companies tend to be vendor agnostic, and rarely want to invest their efforts in proprietary platforms.

The advent of OpenCL and DirectCompute dramatically changes this situation. NVIDIA layered its implementations of OpenCL and DirectCompute on top of the CUDA interface, and thus allows software developers to accelerate applications via GPU computing in a manner that works with all GPUs that support the new APIs. NVIDIA-specific features remain hidden behind these APIs. Some developers will be tempted to cheat and rely on NVIDIA's unique features in a manner that trades off application portability for performance. Insight 64 doubts NVIDIA will do much to discourage such a practice, nor should they. The new OpenCL and DirectCompute APIs won't eliminate the need for ISVs to undertake substantial testing of GPU-enabled applications to ensure they work with the officially supported drivers from AMD and NVIDIA, but testing is a far more straightforward task than rewriting code.

Fermi also took a giant leap forward in the software domain with its newfound ability to support C++ applications. Just as NVIDIA had to re-architect its GeForce 8800 GPU to enable programming in C, it once again needed to revisit its GPU architecture to better accommodate the needs of C++. It's hard to imagine how NVIDIA could have delivered a useful C++ implementation without the unified linear address space they designed into Fermi. Now that they've added this capability, users with computationally intensive workloads who have access to the source code for the applications and middleware they use, and who are willing to port those applications will have the tools they need to get the job done.

The future of GPU computing

OpenCL and DirectCompute will spark the growth of GPU computing used in mass market software applications, but won't eliminate the need for the CUDA-based tools in NVIDIA's product line. Just as some Home Depot customers walk out of the store with preassembled bookcases, while others purchase power tools and raw lumber and build their own unique book storage systems, so too will some computer users be satisfied with off the shelf applications, while others will want to roll their own. In general, it's almost always more cost effective to buy a software package that suits one's needs, rather than writing that package from scratch, but there are not many off-the-shelf packages that can be used to scan sonar images to decide where to drill an oil well, or scan tissue samples inside a patient to detect cancerous growths, or decide whether the price of a stock option accurately reflects the value of that option.

Contemporary computer systems cut through most traditional applications like hot knives through butter, but still struggle with tasks that require massive number crunching capabilities. Conventional multi-core CPUs haven't had much impact in this area. Tasks like face recognition, image processing, and speech translation require more performance than today's fastest CPUs can deliver, but even inexpensive GPUs can noticeably accelerate these tasks. Now that ISVs have the industry standard tools they need to incorporate GPU computing into their mass market packages, Insight 64 anticipates a spate of new, previously unthinkable applications. One such package, Cyberlink's new MediaShow 5, gives users the ability to sort through the thousands of digital photos stored on their systems, finding just the ones that include a specific face. A problem shared by many users, and one that until now remained unsolved, given the huge number of calculations involved in the process. Other common problems, such as translating media files from one format (e.g., mpeg) to another (e.g., avi), have solutions that run painfully slowly; GPU computing speeds up these tasks by factors of three to ten, depending on the GPU doing the computing. As end users come to understand the potential for GPUs to improve the performance of non-graphical applications, they will develop a preference for systems with integrated or discrete GPUs that support such features. Such a preference would work in favor of NVIDIA, at the expense of Intel, whose chipset-integrated (and soon, CPU-integrated) GPUs lack this capability completely.

While consumers amuse themselves playing games and sorting photos with their mass market PCs, professionals will gain the ability to move more and more of their tough computing problems onto GPUs. The number of professional users capable of using compute-intensive tools like finite element analysis or Black Scholes option price calculations is only a tiny fraction of the number of consumers who want to speed up the conversion of files for the MP3 players, but their computational needs are far greater. NVIDIA has already sold systems with hundreds, and in a few cases, thousands of GPUs to research institutions and universities that have serious problems they need to solve. These early adopters have used NVIDIA's CUDA C, and in a few cases that old warhorse, Fortran, to program their solutions. Fermi's new C++ capability will significantly expand the opportunities in this market segment.

These two approaches – on the one hand, using industry-standard GPU computing APIs to enable or enhance mass market PC applications, and on the other, using advanced compiler technology to port compute-intensive proprietary applications to GPU-enabled platforms – mark the two directions in which the industry will evolve. One path leads to higher unit volumes, while the other leads to higher product margins. Insight 64 projects that the industry will develop along both vectors, although it's too soon to say which will be the more profitable for GPU suppliers.