In-Stat

White Paper

Looking Beyond Graphics

Analyst: Tom R. Halfhill
thalfhill@reedbusiness.com
+1.408.425.6728
September 2009
Sponsored by NVIDIA

Looking Beyond Graphics

NVIDIA’s Next-Generation CUDA Compute and Graphics Architecture, Code-Named Fermi, Adds Muscle for Parallel Processing

A typical PC has at least three cooling fans...and one case heater. That “heater” would be the graphics-processing unit (GPU) — usually a separate, highly specialized microprocessor chip dedicated to graphics. It does a brilliant job when the PC is running a graphics-intensive game or playing a video. At other times, it’s largely underutilized, radiating unused power as heat.

In fact, a discrete GPU is the most underutilized component in a PC. Although it’s capable of amazing things, it spends much of its time performing routine chores, like scrolling the screen. Yet it has the potential to be the swiftest processing engine in the system — and it’s already there, just waiting for something to do.

For four years, NVIDIA has waged a campaign to redefine the role of GPUs. Not that graphics aren’t important. Pushing pixels has been good business for NVIDIA, the world’s leading graphics-processor company. Since the early 1990s, NVIDIA’s GPUs have offloaded most of the graphics processing from CPUs in hundreds of millions of personal computers and videogame machines. NVIDIA continues to design its GPUs for superb graphics performance. But since 2005, NVIDIA has cultivated another fast-growing market — GPUs for computing applications beyond graphics. At first, these applications were called “general-purpose GPU” (GPGPU) computing. Today, NVIDIA prefers to call it “GPU computing.” In the professional market, it’s often called high-performance computing.

By harnessing the massively parallel-processing resources originally designed for 3D graphics, clever programmers can apply GPUs to a much broader range of computing applications. Some of those applications, such as video transcoding, still involve graphics to some degree. Whereas the CPU might spend an hour or more converting a recorded video for uploading to YouTube or burning a DVD, a GPU can tear through the job in minutes. The GPU can also enhance the video frame-by-frame, using consumer versions of sophisticated software once available only to NASA, law-enforcement agencies, and surveillance experts.

Other GPU-computing applications are data-intensive tasks having little or nothing to do with graphics. Examples are stock-trading calculations and seismic-data analysis for oil exploration. Additional examples, such as high-resolution medical imaging, combine graphics with heavy-duty number crunching. On these kinds of workloads, an ordinary GPU can blow away the latest multicore CPUs. Link several GPUs together in a workstation or a cluster of systems, and you’ve got a “desktop supercomputer" — a brute-force number-crunching machine at a fraction of the price and the operating costs of a room-sized supercomputer.

Originally, NVIDIA used the term Compute Unified Device Architecture (CUDA) to describe its GPU-computing platform. Later, CUDA became a blanket term for NVIDIA's GPU architecture, run-time platform, and software-development tools. Now NVIDIA is raising the bar.

NVIDIA's next-generation CUDA architecture, code-named Fermi, adds powerful new features for general-purpose computing. Fermi processors will continue shouldering the graphics workloads in PCs and videogame consoles, but they are taking the largest step yet toward becoming equal-partner coprocessors with CPUs. No longer must GPUs be an underused resource.

New Features for GPU Computing

NVIDIA has already received the first sample silicon of a GPU based on the Fermi architecture. If the project proceeds on schedule, the first Fermi GPUs could hit the market this year. Here's a summary of Fermi's improvements:

- Error-correction codes (ECC) protect the external DRAM, L1 data caches, L2 cache, and register files. ECC is unimportant for graphics but vital for financial customers and others running mission-critical software.
- Fermi GPUs will have as many as six I/O interfaces to external DRAM, each 64 bits wide. Although NVIDIA has not announced which types of memory Fermi chips will support, these interfaces will provide greater bandwidth when coupled with the latest GDDR5 DRAM.
- The memory-address space (both virtual and physical) has been unified and expanded from 4GB to one terabyte (1TB), vastly increasing the capacity for large data sets.
- The memory hierarchy is fully cached, including new L1 caches shared by tightly coupled groups of 32 processor cores. For each group, programmers can partition 64KB of SRAM to serve as a hardware-managed L1 cache or a software-managed shared memory.
- Fermi can run concurrent "CUDA kernels" or global functions, whereas existing NVIDIA GPUs are limited to running multiple threads within one kernel. This feature adds another level of parallelism for compute applications, such as PhysX processing.
- For the first time, CUDA runs object-oriented C++ code, not just procedural C code. This improvement will make it easier to program the GPU.
- Fermi-class GPUs can perform double-precision floating-point math up to eight times faster than existing NVIDIA GPUs. Double-precision math is now exactly half as fast as single-precision math. A new fused multiply-add instruction improves precision and conforms to the latest IEEE 754-2008 floating-point specification.
- NVIDIA has revamped the entire instruction-set architecture (ISA) to make it an easier target for high-level compilers. Among other enhancements, all instructions are predicated, eliminating the need for separate branch instructions in some code sequences.

- An improved instruction scheduler can issue two operations per clock cycle, versus one operation per clock before.
- A new load/store unit can execute 16 operations per cycle, greatly improving I/O performance. Atomic read-modify-write instructions are 5 to 20 times faster.

Some of these enhancements are unimportant for 3D graphics but were requested by NVIDIA's existing and prospective GPU-computing customers. Indeed, some features (such as ECC) would actually reduce graphics performance. To avoid compromising the GPU's central role as a graphics coprocessor, Fermi has provisions for disabling or bypassing features that are irrelevant for graphics. For instance, GeForce-branded GPUs for consumer PCs will omit ECC; Tesla- and Quadro-branded GPUs for professional workstations and supercomputers will include it.

GPU Computing Takes Off

Fermi surpasses anything announced by NVIDIA's leading GPU competitor, AMD. Conceptually, Fermi's closest competitor is Intel's Larrabee. Expected to debut next year, Larrabee chips will attack the market from the opposite direction. They will use general-purpose x86-architecture cores for both 3D graphics and GPU computing — a radically different approach than adapting a specialized 3D-graphics architecture for general-purpose computing. Until the first Larrabee chips appear, it's not clear whether the x86 can make such a leap.

No wonder this market is attracting big-gun competition. By NVIDIA's estimates, the total available market for GPU computing is worth about $1.1 billion a year. Figure 1 breaks down the market into broad categories. By comparison, the total available market for GPUs in desktop PCs is worth $1.5 billion to $2.0 billion a year. However, GPU computing is still in its infancy. It's sure to grow as GPUs become more powerful, software-development tools improve, programmers become more experienced, customers find new applications, and more companies realize that GPU computing gives them a competitive edge.

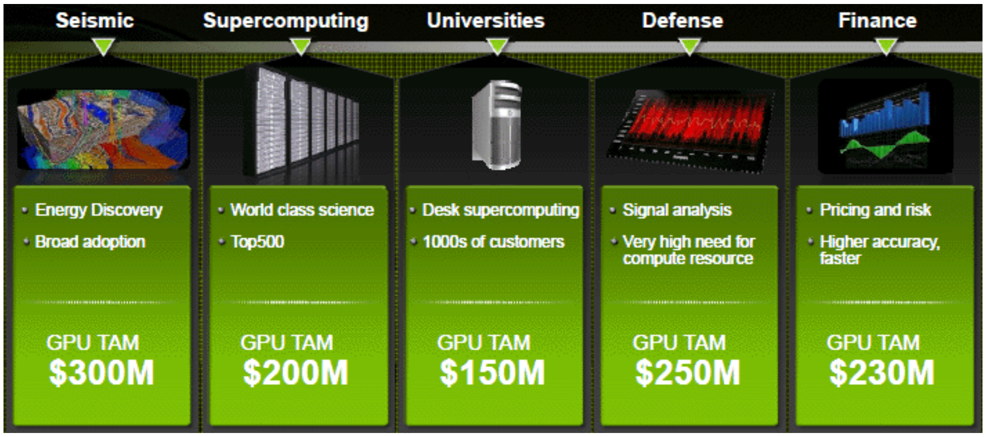


Figure 1. NVIDIA estimates that the total available market (TAM) for GPU computing is at least half as large as the desktop-PC market for GPUs. The upside potential looks greater. Whereas the PC market is maturing, GPU computing barely existed four years ago and is growing fast.

Fundamentally, Fermi processors are still graphics processors, not general-purpose processors. The system still needs a host CPU to run the operating system, supervise the GPU, provide access to main memory, present a user interface, and perform everyday tasks that have little or no data-level parallelism. However, NVIDIA's new features and improvements will transform Fermi GPUs into hybrid coprocessors, capable of tackling a much wider range of applications — for consumers and professionals.

Understanding the Fermi Architecture

NVIDIA's terminology can be confusing for those accustomed to general-purpose CPU architectures like the Intel x86. The confusion isn't deliberate. It's partly a consequence of the graphics heritage of NVIDIA's architectures, but another reason is their massively parallel approach to multithreading. Whereas an Intel x86 processor core with Hyper-Threading typically executes two threads per core, an NVIDIA GPU frequently works on thousands of threads, switching among them on every clock cycle.

CUDA architectures have more in common with the highly specialized, massively parallel CPU architectures that have occasionally appeared since the mid-1990s. Like CUDA, these architectures typically have large numbers of very simple processor cores and massively threaded programming models. However, almost all the specialized architectures soon failed in the marketplace. Some met resistance because programming was too difficult. Others were doomed because their parent companies were startups, underfunded or poorly managed. Sometimes they disappeared only because no one gave them a chance.

Although NVIDIA is overcoming many of the technical hurdles of those architectures, a bigger factor contributes to CUDA's success. CUDA is from a large, mature company, and it piggybacks on the high-volume consumer graphics market. In effect, games are subsidizing CUDA's development while the GPU-computing market becomes self-sufficient. Also, the multicore CPUs from Intel, AMD, and other companies are driving new demand for better parallel-programming languages and tools. CUDA rides all these waves.

Fermi is a significant advance over previous GPU architectures. NVIDIA's first architecture expressly modified for GPU computing was the G80, introduced in November 2006 with the GeForce 8800. The G80 departed from the traditional approach of using dedicated shaders, separate vertex/pixel pipelines, manually managed vector registers, and other features that were efficient for graphics but unwieldy for general-purpose computing. The G80 was also NVIDIA's first GPU architecture programmable in C — an essential feature for compute applications.

G80-class GPUs had 128 programmable shaders, later called streaming processor cores and now known simply as CUDA cores. Eight CUDA cores were grouped into a "streaming multiprocessor" whose cores shared common resources, such as local memory, register files, load/store units, and thread schedulers. G80-class chips had 16 of these streaming multiprocessors. (16 SMs x 8 CUDA cores per SM = 128 cores per chip.) With time slicing and fast thread switching, a streaming multiprocessor can run thousands of parallel threads on these cores.

In June 2008, NVIDIA introduced the GT200 architecture. The GT200 retained eight CUDA cores per streaming multiprocessor but bumped the number of those multiprocessors to 30, for a total of 240 CUDA cores per chip. GT200 chips are sold as the GeForce GTX280 (for consumer PCs), Quadro FX5800 (for workstations), and Tesla T10 (for high-performance computing). These were also the first NVIDIA GPUs to support double-precision floating-point operations — unnecessary for 3D graphics but vital for many scientific and engineering programs.

Fermi supersedes the GT200 architecture. It has 32 CUDA cores per streaming multiprocessor — four times as many as the GT200 and G80. Initially, Fermi GPUs will have 16 streaming multiprocessors, for a total of 512 CUDA cores per chip. This expansion alone would significantly boost throughput, but additional enhancements deliver even more performance.

Fermi, From the Ground Up

The following diagrams illustrate the basic structure of the Fermi architecture. Figure 2 starts at ground level, showing an exploded view of a single CUDA core and its relationship to the streaming multiprocessor to which it belongs. Although a CUDA core resembles a general-purpose processor core, like those found in a multicore x86 CPU, it's much simpler, reflecting its heritage as a pixel shader.

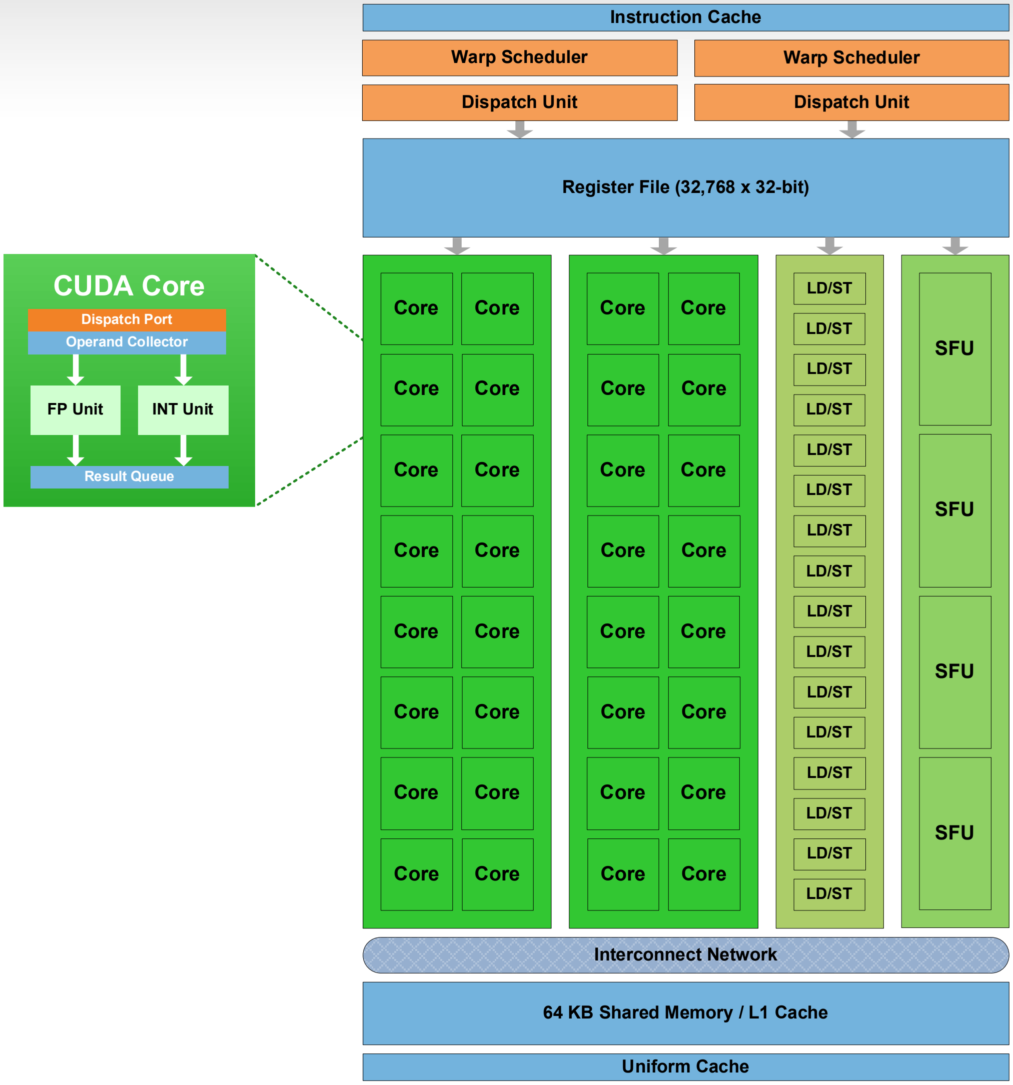


Figure 2. CUDA-core block diagram. CUDA cores evolved from pixel shaders, so they are very simple processor cores. The cores lack their own general-purpose register file, L1 caches, multiple function units for each data type, and load/store units for retrieving and saving data.

Each CUDA core has a pipelined floating-point unit (FPU), a pipelined integer unit, some logic for dispatching instructions and operands to these units, and a queue for holding results. And that's all.

Unlike the processor cores typically found in general-purpose microprocessors, CUDA cores lack their own register files or L1 caches. Nor do they have multiple function units for each data type (floating point and integer). In fact, they don't even have a load/store unit for accessing memory. CUDA cores are very simple processing engines designed to execute only one instruction at a time per thread, then switch quickly to another thread. They are optimized not for single-threaded performance as individuals, but for multithreaded performance when operating en masse.

Figure 3 shows a Fermi streaming multiprocessor with 32 CUDA cores and additional elements. This diagram explains why CUDA cores can get by without their own register files, caches, or load/store units — those resources are shared among all 32 CUDA cores in a streaming multiprocessor. Those 32 cores are designed to work in parallel on 32 instructions at a time from a bundle of 32 threads, which NVIDIA calls a "warp." (This organization has implications for the CUDA programming model, as we'll explain below.)

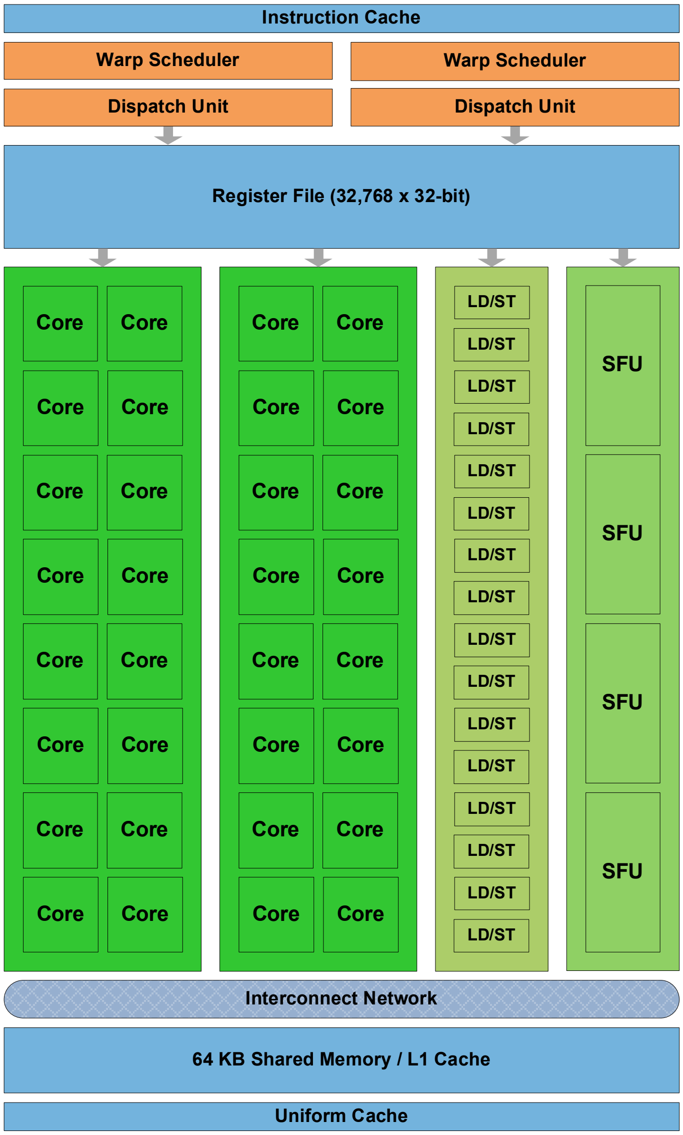


Figure 3. Streaming-multiprocessor block diagram. In the Fermi architecture, each streaming multiprocessor has 32 CUDA cores — four times as many as the previous GT200 and G80 architectures. All 32 cores share the resources of their streaming multiprocessor, such as registers, caches, local memory, and load/store units. The "special function units" (SFUs) handle complex math operations, such as square roots, reciprocals, sines, and cosines.

At the top of Figure 3, notice the L1 instruction cache — a new feature in Fermi. Each streaming multiprocessor also has 64KB of configurable local memory that programmers can partition as L1 data cache and general-purpose shared memory. Each partition can be either 16KB or 48KB. In contrast, the G80 and GT200 architectures had only 16KB of shared memory per streaming multiprocessor, and the memory couldn't operate as hardware-managed L1 caches.

Massively Parallel Multithreading

Another shared resource in a streaming multiprocessor is the instruction-scheduling and dispatch logic. Fermi improves on the scheduler in previous CUDA architectures by issuing two instructions per clock cycle instead of one. However, this capability isn't quite the same as the dual-issue scheduling in conventional superscalar microprocessors.

In a Fermi GPU, the dual-issue pipelines are decoupled and independent. They cannot issue two instructions per cycle from the same thread. Instead, they issue two instructions per cycle from different warps. Each streaming multiprocessor can manage 48 warps. Because each warp has 32 threads, a streaming multiprocessor can manage 1,536 threads. With 16 streaming multiprocessors, a Fermi-class GPU can handle 24,576 parallel threads.

Of course, there aren't enough CUDA cores to execute instructions from every thread on every clock cycle. Each core executes only one instruction per cycle, so instructions from "only" 512 threads can be executing at a given moment. Switching among threads is instantaneous, so instructions from 512 different threads can execute on the next clock cycle, and so on. This massively parallel threading is the key to CUDA's high throughput. Figure 4 illustrates how the warp schedulers and dispatch units interleave the warps and the instructions from different threads.

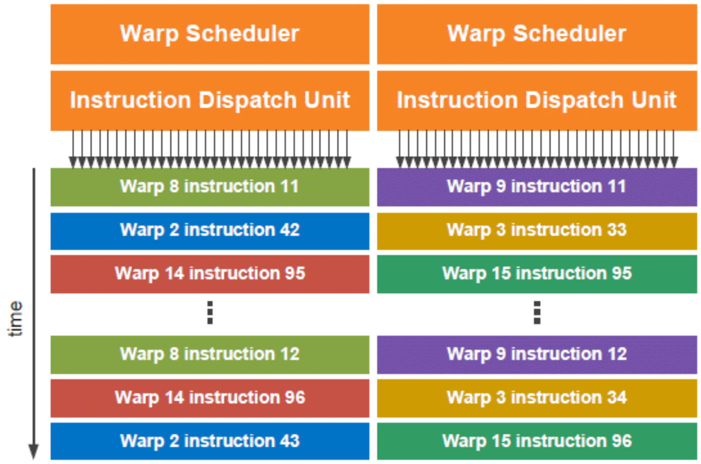


Figure 4. CUDA multithreading in the Fermi architecture. To achieve massively parallel multithreading, CUDA bundles 32 threads into a "warp" and executes one instruction from each thread per clock cycle. Each streaming multiprocessor can manage 48 warps. The warp schedulers and dispatch units interleave the threads and warps to keep the CUDA processor cores busy. The GPU can switch from one thread to another on each clock cycle.

Another shared resource in a streaming multiprocessor is a new load/store unit, which can execute 16 load or store operations per clock cycle. It does even better when using a special "uniform cache," seen at the bottom of Figure 3. Matrix-math operations often load scalar values from sequential addresses belonging to a particular thread, and they also load a common value shared among all threads in a warp. In those cases, a streaming multiprocessor can load two operands per cycle.

Figure 5 is the highest-level view of the Fermi architecture. All 16 streaming multiprocessors — each with 32 CUDA cores — share a 768KB unified L2 cache. By the standards of modern general-purpose CPUs, this cache is relatively small, but previous CUDA architectures had no L2 cache at all. Fermi maintains cache coherency for all the streaming multiprocessors sharing the L2 cache.

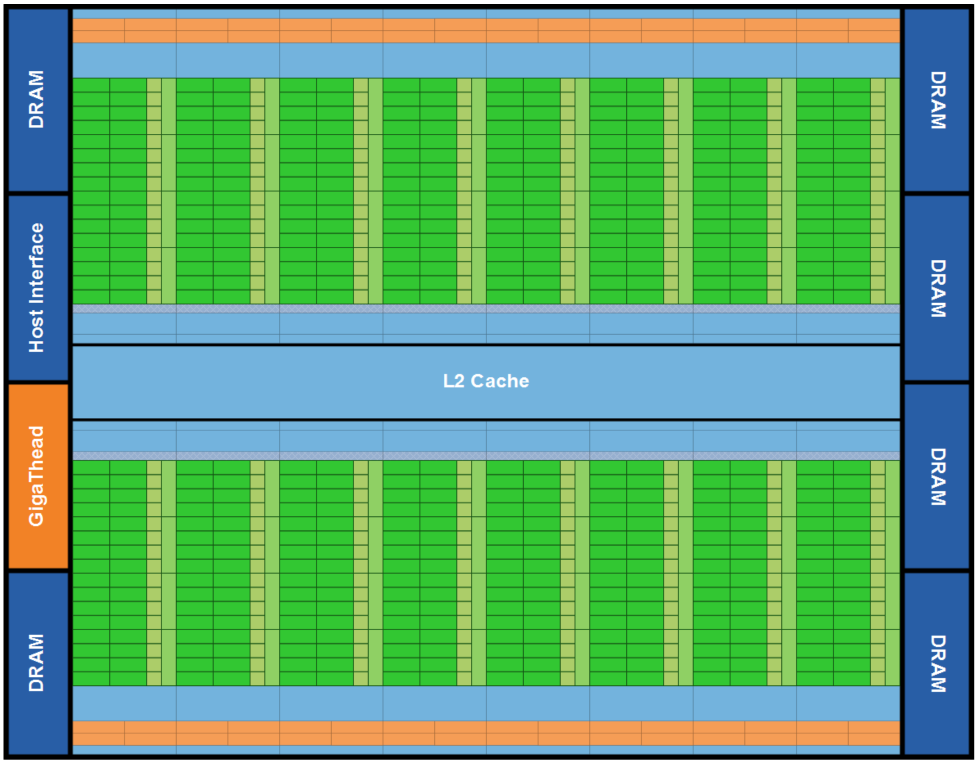


Figure 5. Fermi architecture block diagram. This top-level view of the architecture shows the 16 streaming multiprocessors, the six 64-bit DRAM interfaces, the host interface (PCI Express), and the GigaThread hardware thread scheduler. This improved scheduler manages thousands of simultaneous threads and switches contexts between graphics and compute applications in as little as 25 microseconds — ten times faster than NVIDIA's previous schedulers. (Switching among threads within a graphics or compute instruction stream requires only one clock cycle, but alternating between graphics and compute workloads takes longer, because the caches must be flushed and refilled.)

Fermi's Memory Hierarchy

The memory hierarchy of a Fermi GPU is somewhat different than the better-known hierarchy for a general-purpose CPU. For one thing, a GPU has a large frame buffer — as much as a gigabyte of

DRAM in today's systems. Originally, the frame buffer was dedicated to graphics data. But programs can stash any data in this buffer, a valuable feature for general-purpose computing. For nongraphics applications, NVIDIA sometimes refers to the frame buffer as "local DRAM."

Another important difference between GPUs and CPUs is their proximity to system memory. CPUs have a direct path to main memory through an integrated memory controller or the north bridge of the system chipset. GPUs must reach out over the PCI Express bus by sending a request to the host CPU. The long latency inherent in this transaction requires GPU programmers to use local DRAM, caches, and shared local memory more wisely. Figure 6 illustrates the memory hierarchy.

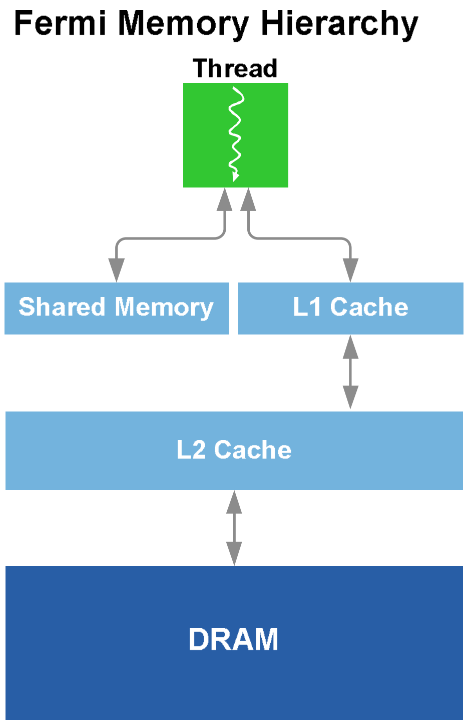


Figure 6. Fermi's memory hierarchy. The shared memory and L1 cache are local to each 32-core streaming multiprocessor and shared by all those cores. The L2 cache is coherently shared among all 16 multiprocessors in the GPU. In this diagram, "DRAM" refers primarily to the GPU's local DRAM (frame buffer), attached directly to the GPU. Additional DRAM is available in the system's main memory, but it requires an exchange over the PCI Express bus.

High-level languages like C and C++ usually have no concept of memory hierarchy. All memory looks the same to them (flat). Left alone, compilers would allocate memory for variables inefficiently. To solve this problem, NVIDIA added new keywords to its CUDA implementation of C and C++. These keywords precede variable declarations, allowing programmers to specify where the program will store the data referenced by the variable. This solution avoids unwanted references to distant memory. Variables declared without a special keyword will be local to a thread.

For example, the _device_ keyword specifies that a variable stores data in the GPU's local DRAM. The _shared_ keyword specifies the shared memory of a streaming multiprocessor, so it limits access to a block of threads. The _global_ keyword makes a variable available to all blocks of threads, so the data probably ends up in system memory. However, all types of memory (except shared memory, which is already local) is cacheable, which minimizes slow references to system memory. Figure 7 illustrates the relationships between local variables, shared variables, global variables, and the memory regions referenced by the variables.

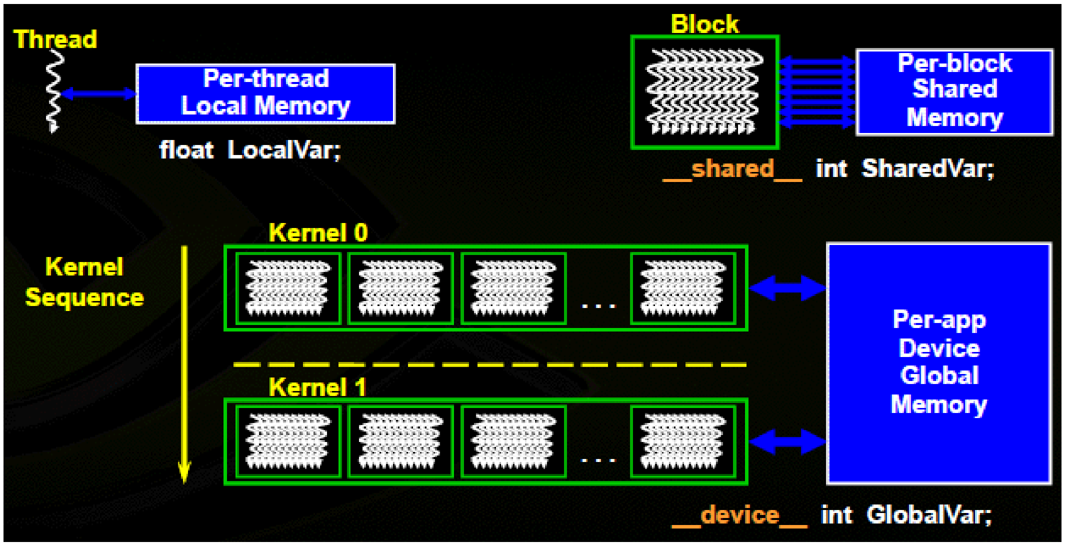


Figure 7. Memory allocation with CUDA. By using special CUDA keywords added to C, programmers can specify the memory regions in which variables store their data. This is important not only for determining the scope of the variables, but also for keeping local data close to the CUDA processor cores. The wider the scope, the more distant the memory.

The six DRAM interfaces seen in Figure 5 provide fast access to local DRAM, essential for graphics processing as well as for general-purpose computing. Each 64-bit interface can work independently. Together, they can access up to 6GB of DRAM. NVIDIA hasn't announced which types of DRAM will be supported in Fermi GPUs or the bus speeds of the chips. Currently, NVIDIA's GTX280 uses GDDR3 DRAM to deliver 141.7GB/s of peak bandwidth. With GDDR5 DRAM, Fermi will certainly exceed that performance. Nevertheless, as with all modern microprocessors, memory bandwidth remains the most likely performance bottleneck.

A Closer Look at Fermi's Features

Fermi's enhancements can be grouped into four broad categories: new features intended exclusively for general-purpose computing; improvements to memory and I/O; features that boost instruction throughput; and features that aid software development. They run the gamut from hardware to software. Some enhancements are useful for accelerating 3D graphics as well as for GPU computing. Although Fermi has additional new features exclusively for graphics, they are not the subject of this paper.

The most-wanted new feature for GPU computing is ECC. It's also a key feature that will set NVIDIA's GPUs apart from AMD GPUs. ECC adds error-checking hardware to the I/O pins of the chip's external DRAM interfaces, but it doesn't stop there. NVIDIA has added ECC throughout the Fermi architecture — to the register files and shared memories of the streaming multiprocessors, and to the coherent L2 cache shared by all multiprocessors. Few microprocessors implement ECC as thoroughly as Fermi does.

ECC protects memory transactions against transient soft errors, such as those induced by electromagnetic interference and the radiation constantly bombarding the Earth from outer space. Soft errors are growing worse as transistors keep shrinking in step with Moore's law. (NVIDIA is manufacturing the first Fermi chips in TSMC's 40nm CMOS fabrication process.) When ECC checksum codes don't match the data, the GPU raises an exception. A program can react by retransmitting the data or by displaying an error message to the user.

ECC is particularly important for data-center applications, where the failure rate of an individual component is amplified by the large scale of the system.

Currently, developers resort to various work-arounds for the lack of ECC in existing GPUs. Some programmers implement ECC in software, severely impairing performance. Others tell users to run their programs two or more times on the same data, then compare the results for consistency. By adding ECC to its next-generation Fermi processors, NVIDIA is giving notice that it's serious about mission-critical computing.

There's always a price to pay for ECC, even when implemented in hardware — and especially when ECC is implemented as extensively as it is throughout Fermi chips. NVIDIA estimates that performance will suffer by 5% to 20%, depending on the application. For critical programs requiring absolute accuracy, that amount of overhead is of no concern. It's certainly much better than the overhead imposed by existing work-arounds. For less-critical applications, the performance hit could be a killer. That's why NVIDIA allows programmers to disable ECC in Tesla-branded GPUs for the professional market and won't enable ECC at all in GeForce GPUs for the consumer market.

Double Precision Gets a Boost

Another Fermi enhancement that focuses primarily on high-performance computing is faster double-precision floating-point math. Traditionally, GPUs have been optimized for single-precision (32-bit) floating point, because it's sufficient for 3D graphics. However, many scientific and engineering applications need the greater accuracy and range of double-precision (64-bit) numbers.

For NVIDIA, these competing interests pose a difficult trade-off. Implementing 64-bit FPUs and datapaths throughout the processor would approximately double the amount of floating-point computational logic and wiring. The chip would be larger (e.g., costlier) and use more power, yet it would be no better for its primary market — consumer 3D graphics.

As a result, GPUs didn't even support double-precision floating point until very recently, and the initial support has been lackluster. The first NVIDIA GPU to have double precision was the GT200-based GeForce GTX280, introduced in June 2008. With Fermi, NVIDIA is taking a big step forward. The new architecture is up to eight times faster with double-precision floating point and enhances the accuracy of some single-precision operations, too.

Fermi's floating-point datapaths are still 32 bits wide, minimizing the amount of logic and wiring throughout the chip. To perform 64-bit operations, these 32-bit paths are paired, and NVIDIA has significantly improved their efficiency. A 64-bit floating-point operation is now exactly half as fast as a 32-bit floating-point operation. Although that relationship seems only logical when pairing two 32-bit datapaths, today's GPUs are much slower, because the synchronization is less efficient.

Specifically, a Fermi-class streaming multiprocessor can execute 16 double-precision fused multiply-add (FMA) instructions per clock cycle. That's eight times faster than the FPUs in today's GT200 chips. An FMA performs two operations with a single instruction (D = A x B + C). Therefore, a Fermi-class streaming multiprocessor can execute 32 double-precision floating-point operations per cycle. With 16 streaming multiprocessors per chip, a Fermi GPU can execute 512 billion double-precision floating-point operations per gigahertz.

Currently, the streaming multiprocessor in a GeForce GTX280 chip runs at 1.29GHz, limited by an older 65nm fabrication process. Manufactured at 40nm, a Fermi-based GPU should easily exceed that clock frequency. (NVIDIA has not yet disclosed the clock rates for any Fermi GPUs.) At 1.5GHz (our conservative estimate), the GPU would deliver 768 double-precision gigaflops — and twice as many single-precision gigaflops. (Fermi's enhanced FMA instruction supports 32-bit operations, unlike previous generations.) At 2.0GHz, double-precision performance would reach the magic teraflops threshold, once the province of multimillion-dollar room-size supercomputers.

Intel's Larrabee aspires to similar levels of performance. A single Larrabee core can perform 32 floating-point operations (single precision) per clock cycle. Double precision is half as fast, like Fermi. Therefore, at 1.0GHz, the maximum theoretical throughput is 32 gigaflops per core. Intel hasn't announced any specific implementations of Larrabee, so the number of cores per chip and their clock speeds are unknown. At 1.0GHz, a Larrabee chip would need 32 cores to reach one teraflops. Alternatively, 16 cores running at 2.0GHz would reach the same threshold.

In public presentations, Intel has estimated the relative (not absolute) performance of Larrabee chips with as few as eight cores and as many as 64 cores. (We expect to see different implementations for different markets.) It's likely that higher-end Larrabee designs will challenge the performance of NVIDIA GPUs based on the Fermi architecture. Whether Larrabee can deliver that performance at a similar cost and with similar power consumption remains to be seen.

Better Multiply-Adds

In addition to boosting floating-point throughput, the Fermi architecture improves precision. In fact, GPUs can now perform floating-point operations more accurately than x86 processors do. Fermi maintains full precision during the intermediate calculations of an FMA — even for single-precision operations — instead of truncating or rounding the intermediates. Previously, only double-precision FMAs were handled this way.

(Precise single-precision arithmetic is important for 3D-graphics as well as for high-performance computing. It reduces the “sparkling” effects marring the edges of objects when imprecise calculations change the positions of pixels from one frame to another.)

In accordance with the latest IEEE 754-2008 specification, Fermi’s floating-point hardware supports all four rounding modes (nearest, zero, positive infinity, and negative infinity). There’s also new hardware support for handling “subnormal” numbers — those that fall between the smallest-possible positive number and largest-possible negative number of the floating-point number system.

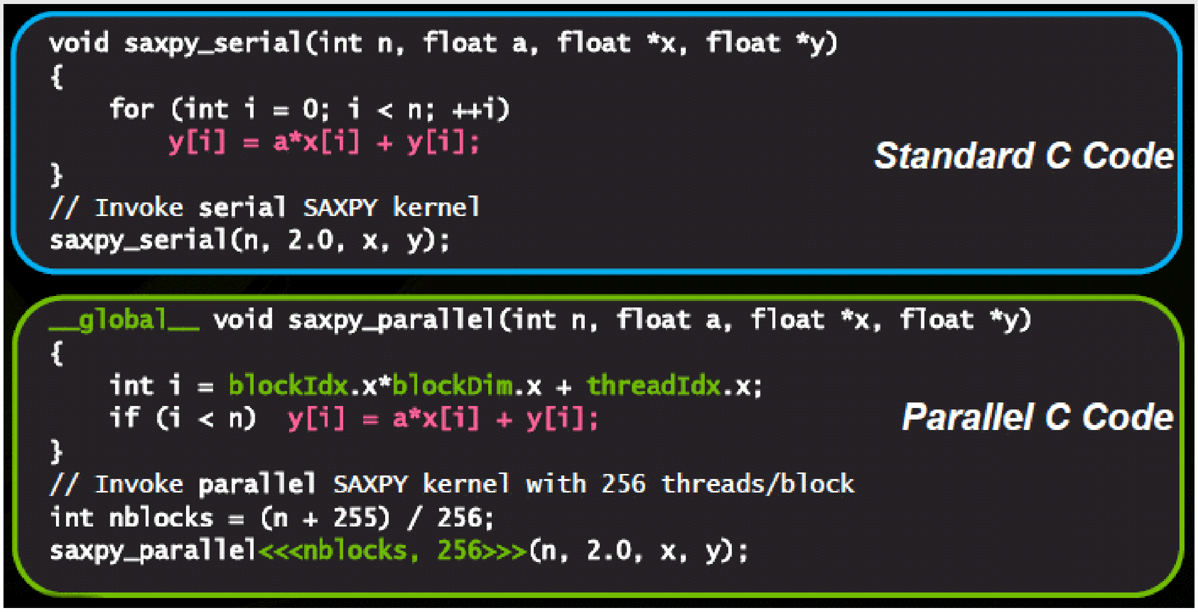
Today’s GPUs simply flush subnormal numbers to zero, losing accuracy. An x86 CPU throws an exception in these cases and spends thousands of clock cycles handling the exception in software. Fermi handles subnormals in hardware, allowing them to gravitate toward zero without sacrificing accuracy or performance.

NVIDIA’s enhanced FMA instruction and full support for IEEE 754-2008 put the Fermi architecture on par with other high-performance microprocessor architectures and exceed the capabilities of some of them. Intel’s latest Streaming SIMD Extensions (SSE4.2) for the x86 lack an FMA, and the x86 incurs rounding errors when performing its version of a multiply-add (MADD). However, Intel is introducing new FMA instructions for the wide 16-lane vector-processing units (VPUs) in Larrabee. Intel’s Itanium architecture also has FMA, as does IBM’s Cell Broadband Engine (but only for double precision, not single precision).

Along with much faster double-precision throughput, the greater precision of these operations makes the Fermi architecture a serious contender in the realm of high-performance computing for science and engineering.

New Concurrency for Global Kernels

Another significant performance improvement in the Fermi architecture is the ability to run concurrent global functions, or “CUDA kernels,” as NVIDIA calls them. Existing CUDA architectures can execute multiple threads within a kernel, but they can’t run more than one kernel at a time. Figure 8 compares a function written in standard C code with a CUDA kernel rewritten for parallel execution.



```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Standard C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

Figure 8. CUDA kernels are modified versions of conventional functions written in C. The _global_ keyword is a special CUDA extension – it tells the compiler this function is a CUDA kernel. The other changes tell the compiler to divide the workload into blocks, each with 256 threads.

Concurrent global kernels add another level of parallelism to CUDA. Previous CUDA architectures are limited to exploiting data-level parallelism when a single algorithm operates on a single data set. A global kernel may call a sequence of algorithms in a step-by-step fashion, and the data may reside in two or more arrays, and the function may spawn thousands of threads. However, at any given moment, the streaming multiprocessors can execute only one global kernel. Executing another global kernel requires waiting for the previous global kernel to finish.

The Fermi architecture overcomes that limitation and allows up to 16 independent global kernels to execute concurrently. This feature adds a degree of instruction-level parallelism to CUDA's forte, data-level parallelism.

One remaining limitation is that concurrent global kernels must belong to the same program. Fermi still cannot manage application-level parallelism. For example, as is the case today, the GPU must switch contexts to handle the separate instruction streams of graphics and compute tasks. Nevertheless, the additional parallelism of concurrent kernels within a program is a welcome enhancement to CUDA's prodigious capacity for data-level parallelism.

Table 1 summarizes the outstanding differences between the Fermi architecture and NVIDIA's previous GT200 and G80 architectures.

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double-Precision Floating Point	None	30 FMA ops per clock	256 FMA ops per clock
Single-Precision Floating Point	128 MADD ops per clock	240 MADD ops per clock	512 FMA ops per clock
Warp Schedulers per Streaming Multiprocessor (SM)	1	1	2
Special Function Units per SM	2	2	4
Shared Memory per SM	16KB	16KB	Configurable 48KB or 16KB
L1 Cache per SM	None	None	Configurable 16KB or 48KB
L2 Cache	None	None	768KB
ECC Memory Protection	No	No	Yes
Concurrent Kernels	No	No	Up to 16

Table 1. Comparison of NVIDIA's three CUDA-capable GPU architectures. The G80, introduced in 2006, opened the door to CUDA and was NVIDIA's first architecture modified for GPU computing. The GT200 architecture followed in 2008. Although it improved performance, it retained several limitations of the G80. The new Fermi architecture overcomes most of those limitations and adds features not seen before in GPUs, such as error-correction codes (ECC).

Instruction-Set Improvements

NVIDIA has overhauled the Fermi instruction-set architecture (ISA) to make it friendlier for high-level language compilers. For instance, all instructions are now fully predicated. An instruction can decide whether or not to execute by checking a status bit, which may be set or cleared by a previous instruction. In some code sequences, predication eliminates the need for separate compare and branch instructions, allowing a compiler to generate more-streamlined code.

Additionally, the Fermi ISA improves the efficiency of "atomic" integer instructions. These are read-modify-write instructions that read a value in memory, modify the value by performing an arithmetic or logical operation, then quickly write the result back to the same memory location. Atomic instructions must be uninterruptible. Otherwise, an incomplete sequence of operations would leave the data in an unpredictable state, with possibly serious consequences for the program.

NVIDIA GPUs have included atomic instructions since 2007, but they were relatively slow. Fermi makes them 5 to 20 times faster. For the sake of efficiency, atomic instructions are handled by special integer units attached to the L2 cache controller, not by the integer units in the CUDA cores. Executing atomic instructions in the cores would introduce too much latency as the data is shuttled across the chip.

Efficient handling of atomic instructions is important for software developers. Two emerging standards for parallel programming have embraced the concept of atomic operations: OpenCL and DirectCompute. OpenCL is an open industry standard managed by the OpenCL Compute Working Group, which is part of an industry consortium called the Khronos Group. DirectCompute is an application programming interface (API) from Microsoft and was formerly known as Compute Shader or DirectX 11 Compute. DirectCompute is part of the larger DirectX 10 and DirectX 11 APIs in Windows Vista and Windows 7. Developers can use OpenCL or DirectCompute instead of (or in addition to) NVIDIA's CUDA tools.

The Fermi architecture allows compilers to optimize the memory layout for arrays when using OpenCL or DirectCompute. Efficient memory layout is a crucial feature, because arrays are central to the programming model. All data sets on which the parallel threads operate are stored in arrays.

Programmers Get a Break

Just a few years ago, GPUs weren't even programmable in C. GPGPU pioneers had to struggle with an unconventional assembly language and try to fit the square pegs of their general-purpose code into the round holes of a graphics-oriented architecture. NVIDIA's CUDA development tools were a godsend. They allowed programmers to write code in fairly conventional C, albeit with a few missing features and some extensions specific to CUDA.

The Fermi architecture makes the GPU programmable in object-oriented C++, not just in procedural C. Moreover, Fermi supports the entire C++ language, not a subset. Programmers can use virtual functions, try/catch statements (which handle exceptions), and make system calls, such as stdio.h (the standard input/output channel). NVIDIA provides a CUDA C/C++ compiler, but any third-party tool vendor can target the CUDA architecture.

A key factor in bringing C++ to CUDA is Fermi's new unified memory addressing. Previous CUDA architectures allocated different memory regions for local memory, shared memory, and global memory. The fragmented memory map required different load/store instructions for each region and prevented NVIDIA from fully implementing pointers in C and C++. In particular, C++ relies heavily on memory pointers, because every "object" in this object-oriented language requires at least one pointer. A typical program creates hundreds of objects.

The Fermi architecture fixes those problems by unifying everything in a single memory space. Load/store instructions can now access any type of memory within this unified space. At run time, Fermi uses a new address-translation unit to convert pointers and other memory references to physical memory addresses. In addition, Fermi expands the memory-addressing range from 32 bits (maximum 4GB of memory) to 40 bits (maximum 1TB of memory). Together, these changes bring higher-level languages to CUDA, simplify the programming model, and vastly expand the memory available for large data sets — vital features for high-performance computing.

Unlike most other compilers, CUDA compilers don't translate source code directly into native machine code. Instead, they target a low-level virtual machine and Parallel Thread eXecution (PTX) instruction set. The PTX virtual machine is invisible to users and delivered as part of the GPU's graphics driver.

Few people even know it’s there, but it’s very important to CUDA software developers. PTX provides a machine-independent layer that insulates high-level software from the details of the GPU architecture.

When users install a CUDA-enabled program, the driver automatically translates PTX instructions into machine instructions for the specific NVIDIA GPU in the system. It doesn’t matter whether the GPU is based on the G80, GT200, or Fermi architecture — the installed program will be optimized for that GPU. In this way, CUDA programs run natively (full speed) but are delivered in a machine-independent format that’s ready for installation on any CUDA-capable system. (NVIDIA says more than 100 million PCs have CUDA-capable GPUs.)

Besides bringing C++ to CUDA, the Fermi architecture with PTX 2.0 makes it easier to use other high-level programming languages and compilers. Fermi supports FORTRAN — a vital language for scientific and engineering applications — as well as Java (via native methods) and Python. Many financial programs are written in Java, and Python is popular for rapid application development. PTX 2.0 also adds features for OpenCL and DirectCompute, such as new bit-reverse and append instructions.

Fermi enables better tool integration, too. For the first time, programmers can use Microsoft’s Visual Studio development environment to write CUDA software. CUDA integration allows full use of Visual Studio’s source-code editors, inspectors, debuggers, profilers, and other productivity features. Because Fermi fully supports exception handling at the hardware level, programmers can set breakpoints in their CUDA source code and single-step through a program while debugging. Visual Studio is a popular development environment, so programmers will welcome the new CUDA integration. (Visual Studio with CUDA is currently in beta release.)

Everyday users will be oblivious to all these behind-the-scenes improvements for software development. However, the Fermi architecture is certain to attract new developers and broaden the base of CUDA software.

Conclusions

With the Fermi architecture, NVIDIA is making a big push to move its GPUs beyond consumer graphics and games. Not that games aren’t important — they remain the driving force behind the evolution of discrete GPUs. Serving the consumer market will continue to be Fermi’s “day job.” But GPUs are becoming more than mere coprocessors for offloading graphics chores from the system’s host CPU. Their role is steadily growing.

For consumers, GPU computing can speed up media-intensive programs that run sluggishly on even the latest multicore CPUs. To take just one example, digital video is wasting untold hours of time as PCs grind through a lengthy transcoding task. CUDA-enabled transcoders like Elemental Technologies’ Badaboom can reduce the waiting period to minutes. And cleaning up the video frame-by-frame was unthinkable until MotionDSP’s vReveal came along. The wide appeal of these programs will force consumers to take a closer look at the specs of a new PC. Until now, users who disdained games could be satisfied with a low-end graphics processor integrated with the system chipset. GPU computing opens up entirely new possibilities, potentially making a discrete GPU a must-have feature.

For professionals, GPUs are even more compelling. Whereas the performance of multicore CPUs is rising incrementally, high-performance computing on GPUs is growing by leaps and bounds. As tools improve and developers get the hang of programming these beasts, the gains can exceed two orders of magnitude. And time is money — sometimes figuratively, sometimes literally. If an energy company can expand its oil reserves by analyzing seismic data more quickly, it can pump more crude and capture more market share. If an investment company can calculate the optimum strategy for trading stock options, it can gain a crucial edge in financial markets.

One obstacle — the much-discussed difficulty of writing parallel software — is gradually being overcome by better GPU architectures and development tools. NVIDIA's Fermi architecture is a significant leap forward in this regard. It's not just a faster chip. By supporting C++ (and other high-level languages), OpenCL, DirectCompute, and Visual Studio, it will be easier to program, too. As developers look around and see their competitors porting applications to GPUs, resistance to new programming paradigms will fall, and we'll all get faster software.

Perhaps the most exciting prospects are the applications yet to be discovered. Every leap in computing power leads to new applications that were impractical before. Unlocking the power of parallel processing has been a dream of computer scientists for decades. The advent of affordable chip-level multiprocessing may be the catalyst that makes the dream come true. If only one lesson can be drawn from the history of computing, it's that the world has an insatiable appetite for computing power. Someone will always find a way to use it.

NVIDIA already has the momentum in this field, ahead of AMD's graphics processors and Intel's yet-to-be seen Larrabee. The Fermi architecture has the potential to increase that lead on all fronts. AMD is pursuing a somewhat different strategy with its "Fusion" chip architecture. Fusion will integrate a graphics processor with the x86 CPU on the same chip – an option unavailable to NVIDIA, which doesn't make x86 processors. CPU/GPU integration has various trade-offs, but it will probably boost graphics performance in lower-cost systems. Intel's Larrabee remains an unknown factor, too, and it will offer different trade-offs than conventional GPU architectures.

Fermi's only vulnerability may be its attempt to combine world-class graphics performance with general-purpose compute performance in one chip. With three billion transistors, a Fermi GPU will be more than twice as complex as NVIDIA's existing GT200 chips. At some point, if not now, features intended to boost compute performance may compromise the chip's competitive position as an affordable graphics processor. At that juncture, the architectures may have to diverge — especially if the professional market grows larger than the consumer market.

But we're getting ahead of ourselves. Right now, the important point is that Fermi significantly advances the state of the art in this field. It's up to NVIDIA to implement the architecture in a good chip design, manufacture the parts without hiccups, and ship the finished products — before competitors can steal back the momentum.

Appendix

In-Stat's *Microprocessor Report* has published the following related articles:

"Parallel Processing With CUDA: NVIDIA's High-Performance Computing Platform Uses Massive Multithreading," *MPR* 1/28/2008.

"Summer Shopping Spree: Intel Buys Cilk Arts and RapidMind; Virage Logic Wants ARC," *MPR* 9/14/2009.

"Going Parallel With Prism: New Analysis Tool Helps Programmers Refactor Serial Code," *MPR* 4/27/2009.

"AMD's Stream Becomes a River: Parallel-Processing Platform for ATI GPUs Reaches More Systems," *MPR* 12/22/2008.

"Intel's Larrabee Redefines GPUs: Fully Programmable Manycore Processor Reaches Beyond Graphics," *MPR* 9/29/2008.

"Tools for Multicore Processors," *MPR* 7/28/2008.

"Think Parallel," *MPR* 3/31/2008.

"Parallel Processing For the x86: RapidMind Ports Its Multicore Development Platform to x86 CPUs," *MPR* 11/26/2007.

"The Dread of Threads," *MPR* 4/30/2007.

"Number Crunching With GPUs: PeakStream's Math API Exploits Parallelism in Graphics Processors," *MPR* 10/2/2006.

In-Stat Offices

Arizona	Massachusetts	China
+1.480.483.4440	+1.781.734.8674	+86 10 6642 1812

This report is the property of In-Stat and is made available only upon these terms and conditions. The contents of this report represent the interpretation and analysis of statistics and information that is either generally available to the public or released by responsible agencies or individuals. The information contained in this report is believed to be reliable but is not guaranteed as to its accuracy or completeness. In-Stat reserves all rights herein.