The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges

David Patterson
Director, Parallel Computing Research Laboratory (Par Lab), U.C. Berkeley[1]
September 30, 2009

I believe the Fermi architecture is as big as an architectural advance over G80 as G80 was over NV40. The combined result represents a giant step towards bringing GPUs into mainstream computing. The table previews my take on the Top 10 most important innovations in the new Fermi architecture. This list is from a computer architect's perspective, as a user would surely rank performance higher. At the end of the paper, I offer 3 challenges on how to bring future GPUs even closer to mainstream computing, which the table also lists.

	Top 10 Innovations in Fermi	*Top 3 Next Challenges*
1	Real Floating Point in Quality and Performance	The Relatively Small Size of GPU Memory
2	Error Correcting Codes on Main Memory and Caches	Inability to do I/O directly to GPU Memory
3	64-bit Virtual Address Space	No Glueless Multisocket Hardware and Software
4	Caches	
5	Fast Context Switching	
6	Unified Address Space	
7	Debugging Support	
8	Faster Atomic Instructions to Support Task-Based Parallel Programming	
9	A Brand New Instruction Set	
10	Also, Fermi is Faster than G80	

Let's start with a sore spot highlighted by GPU critics in the past.

1. Real Floating Point in Quality and Performance

Fermi now has first class floating point both in quality and in balanced performance.

To my knowledge, Fermi is the first implementation of the newly revised IEEE 754-2008 standard, with Fused Multiply Add, a 16-bit floating-point memory format, all the rounding modes, and even support for denormalized numbers in hardware.

Regarding performance, double precision floating point is now half the speed of single precision, like other mainstream processors, versus a tenth of the speed as it is in G80. This new ratio allows programmers to make the identical algorithmic tradeoffs that they make for mainstream computing rather than wrestle with the tempting 10:1 incentive to do all computations at single precision, even when it might be dangerous to do so.

---

[1] NVIDIA is one of eight sponsors of the Par Lab. See parlab.eecs.berkeley.edu for the full list.

2. Error Correcting Codes (ECC) on Main Memory and Caches

While no obvious performance benefit, in practice soft memory errors occur often enough in long running programs that virtually every serious mainstream computer offers single error correction, double error detection (SEC/DED) ECC for main memory and caches.

Fermi now joins that responsible group of architectures that protect memory, which the High Performance Computing market demands.

3. 64-bit Virtual Address Space

Although GPUs normally have their own memory and thus their own address space, a 32-bit address space in 2009 is at a well-known computer architecture precipice, as G80s are already shipped with 2^{32} byte memories (4 GB). Mainstream computing switched from 32-bit addresses to 64-bit addresses between 5 to 15 years ago, at considerable effort by the hardware and software engineers of these companies. As two famous computer architects, Gordon Bell and Bill Strecker, wrote in 1976:

> *"There is only one mistake that can be made in a computer design that is difficult to recover from - not providing enough address bits for memory addressing and memory management."* [2]

One of the benefits of the coprocessor interface to GPUs is that their programs are shipped at a much higher level than mainstream computing, which is saddled with binary machine language programs and, consequently, legacy binary code. The Parallel Thread eXecution layer (PTX) allowed the Fermi architects to switch from 32-bit to 64-bit addresses without the traumatic disruption to software that we saw with mainstream computing.[3]

4. Caches

The GPU dogma regarding caches had been: why waste precious chip resources on caches since GPU programs have enough data parallelism, and GPU chips can run enough threads concurrently to hide the long latency to DRAM?

This claim is certainly true for some programs, but some do not have enough data parallelism to hide that latency no matter how many threads the hardware supports, and others have unpredictable data reuse patterns that defy software controlled scratchpad memories. As part of its large step towards mainstream computing, Fermi now has configurable 64 KB private first-level caches with every streaming multiprocessor and a 768 KB shared second-level cache.

---

[2] G. Bell and W. D. Strecker, "Computer Structures: What Have We Learned from the PDP-11," The 3rd Annual Symposium on Computer Architecture Conference Proceedings, pp. l-14, 1976.

[3] While Fermi implements only 40 bits of the potential 64-bit virtual space, the breakthrough is that addresses can be larger than 32 bits, and successors can easily increase virtual address bits when needed, just like mainstream processors.

Note that Fermi includes an interesting feature to get more value from the first level caches that many researchers, including my colleagues at Berkeley[4], have been exploring. That is to try to get best of both worlds of private local scratchpad memories and caches by splitting the 64 KB as either a 48 KB cache and 16 KB local scratchpad or 16 KB cache and 48 KB local scratchpad. (NVIDIA calls the local scratchpad "shared memory," but many architects prefer a more descriptive term.) I believe allowing software to have explicit control of a portion of the cache can simplify programming, improve performance, and improve energy efficiency.

One cautionary note: caches were invented to simplify the programmer's task by automatically managing a hierarchy of memories in speed and size, with levels further from the processor being slower and larger. While Fermi follows the traditional speed hierarchy, it inverts the size hierarchy. The total size of the registers (16 * 128 KB = 2048 KB) is larger than the total size of the L1 caches (16 * 48 KB = 768 KB), and the total size of the L1 caches equals the L2 cache size (768 KB).[5]

It will be interesting to see how well compilers and programs utilize Fermi's caches, given this size inversion.

5. Fast Context Switching

One requirement for mainstream computing is to be able to quickly switch from one program to another, so the operating system can better utilize the hardware depending on the demands of the user. Operating systems programmers complain to architects if processors take too long to switch contexts.

Coprocessors like GPUs classically ignored context switch time, since their goal was to accelerate one program at a time. This neglect changes with Fermi, which dramatically accelerates context switching compared to its predecessors. Fermi takes only 20 to 25 microseconds to context switch between programs, which means that in practice it may get better utilization in some environments.

This feature will likely make individual programs run faster as well, as Fermi makes it possible to allow independent kernels to overlap execution; for example, by letting one compute while another is communicating.

---

[4] H. Cook, K. Asanovic, and D. A. Patterson, "Virtual Local Stores: Enabling Software-Managed Memory Hierarchies in Mainstream Computing Environments," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-131, Sep. 2009, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-131.html.

[5] In case you were wondering, Fermi protects its large register file with ECC in addition to protecting memory and caches with ECC.

6. Unified Address Space

Past GPUs had a variety of different types of memories, each in their own address space. Although these could be used to achieve excellent performance, such architectures are problematic with programming languages that rely on pointers to any piece of data in memory, such as C, C++, and CUDA

Fermi has rectified that problem with by placing those separate memories—the local scratchpad memory, the graphics memory, and system memory—into a single 64-bit address space, thereby making it much more easier to compile and run C and C++ programs on Fermi.[6] Once again, PTX enabled this relatively dramatic architecture change without the legacy binary compatibility problems of mainstream computing.

Like caches, a unified address space increases the types of programs that can run well on GPUs.

7. Debugging Support

Bugs are the bane of all software, so mainstream computers have long had the ability to set traps or breakpoints in code and allow the processor to step one instruction at a time to try to help the poor programmer find bugs. To support debugging in prior GPUs, traps, breakpoints or single-stepping first freeze the entire GPU state, and then the CPU-based debugger reads and writes GPU thread registers, thread state, and GPU memory over the link between system memory and GPU memory. The CPU based debugger then resumes GPU execution.

A significant change in Fermi is that traps, breakpoints, and so on are now handled in GPU trap handler software by GPU threads. GPU trap handler software can interact with CPU code as needed. That lets the GPU be more flexible, enables robust debugger actions, and lets the GPU provide system call support.

This innovation will likely popularize debuggers like GDB or Integrated Design Environments like Microsoft's Visual Studio for programming GPUs.

8. Faster Atomic Instructions to Support Task-Based Parallel Programming

Data parallel programs classically have less need for synchronization than many other types of parallel programs, so GPUs historically had little support to help parallel tasks cooperate.

The hardware challenge is to make a primitive that allows parallel tasks to cooperate safely without slowing them down. The classical solution is called *atomic*

---

[6] While there are still the memories used for graphics that are not part of the unified address space (constant memory and texture memory), I believe it is the rare GPU programmer that to uses them as well as the other memories. Hence, I believe that in practice Fermi has achieved unification of memories in a single address space.

instructions. An atomic instruction reads from a shared location in memory, checks its value, and writes a new value back *without any other processor being able to change the memory value during the execution of the atomic instruction*. This allows tasks to safely signal to each other, for example "leave me alone since I'm still working on the task (0)," or "it's OK now since I'm done (1)."

While G80 had atomic instructions, by allowing these atomic values to be placed in the shared L2 cache in Fermi, atomic instructions can be 5X to 20X faster. Once again, Fermi joins its mainstream brethren in providing a fast version of an important primitive. Like caches and the unified address space, fast atomic instructions increase the types of programs that can run well on GPUs.

9. A Brand New Instruction Set

As long as they were changing the address size, providing a unified address space, and improving atomic instruction, the Fermi architects changed instructions sets completely to a more RISC-like load/store architecture instead of an x86-like architecture that had memory-based operands. Although this dramatic change took dozens of person years to pull off, once again PTX made it much less onerous than when mainstream computing switches instruction sets.

I expect that this new instruction set architecture will be the foundation for NVIDIA processors for a very long time, just as the revised 64-bit instruction sets for mainstream computers will be long lasting, although all groups will surely add to their instruction sets in the future as they have in the past.

10. Also, Fermi is Faster than G80

Given NVIDIA's long track record of significant increases with each GPU generation, its no surprise that Fermi is faster than G80. In addition to the caches and fast double precision floating point hardware, the 3 billion transistors of Fermi:

- Commonly complete 512 (16*32) 32-bit integer or 32-bit floating-point operations per clock cycle versus the typical case of 240 (30*8) per clock cycle in G80;
- Use faster GDDR5 DRAM versus GDDR3 DRAM in G80;
- Have a path to system memory that is now a bidirectional PCI Express link transferring 12 GB/second, as opposed to the unidirectional bandwidth of 3 GB/second in G80; and
- Include support for Simultaneous Multithreading[7] in addition to the fine-grained multithreading found in previous GPUs, which can improve hardware utilization even further.

---

[7] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor." *Proc. 23rd Int'l Symp. on Computer Architecture*, 191–202, May 1996.

Are We There Yet?

Did Fermi take such a large step that there is no longer a difference between GPU and mainstream computing?

No. There are still at least three more challenges for NVIDIA architects to contemplate.

1. The Relatively Small Size of GPU Memory

Implicit in any quote of the performance of a program is the assumption that the program and all its important data fit in main memory. In part because of the high bandwidth requirements of Graphics DRAM, a small number of GDRAM chips are soldered onto the board near the GPU. In addition, GDRAM chips tend to be lower capacity than standard DRAM chips, so this combination limits the amount of memory that the GPU can access. Typically, that number is no more than 4 GB today.

Back in the mainframe days, the famous computer architect Gene Amdahl proposed a rule of thumb for a well-balanced design:

> *"1 byte of memory and 1 byte per second of I/O are required for each instruction per second supported by a computer."*[8]

Thus, according to his rule of thumb, a well-balanced computer that delivers 500 GFLOPS of performance should have 500 GB of memory, making GPU memory about 100 times too small. The relationship between the cost of processors and the cost of memory have changed over the decades, so that this guideline is not a hard and fast rule anymore, but there clearly is a relationship between performance and the size of the data for programs that need that performance. Common sense tells us that a typical use of a faster computer is to solve a larger problem.

Stated alternatively, I am sure there would be a market for GPUs with terabytes of DRAM, even if the DRAM bandwidth dropped a bit to allow for the larger capacity.

I don't think addressing this significant shortcoming technically is rocket science. It's just a question of the market size and the business model.

2. Inability to do I/O directly to GPU Memory

Real programs do input and output from storage devices, and not just to frame buffers. Large programs that need high performance often need to do a lot of I/O in addition to access a lot of memory, as Amdahl's Rule of Thumb suggests. Today's GPU systems must transfer between I/O devices and system memory, and then between system memory and GPU memory. This extra hop significantly lowers I/O performance in some programs, making GPUs less attractive.

---

[8] John Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach*, 4th Edition, Elsevier, 2008.

Gene Amdahl also gave us his famous law[9], which quantified the law of diminishing returns, to help measure such a bottleneck. Suppose a program spends 10% of its time in I/O, and a new GPU makes the computation piece go 10 times faster. The speedup will just be 1/(.1 + .9/10), or a little over 5 times faster, wasting almost half of the GPU potential. If instead the GPU could make the computation run 100 times faster, the speed up will be 1/(.1 + .9/100), or a little over 9 times faster, wasting more than 90% of the GPU potential. Thus, architects should not neglect I/O.

In addition to I/O performance, mainstream computers provide virtual memory in part to page-out unused portions of data or programs to secondary storage. Some programs would benefit from being able to page GPU memory, which is not supported in Fermi.

Plausibly, a Fermi follow-on will make storage I/O a first class citizen just as graphics I/O is today.

3. No Glueless Multisocket Hardware and Software

Semiconductors are a high volume manufacturing business combined with very expensive development. Even simple designs can cost a $100M to produce. To sell more chips from a single design, mainstream computer architects often plan their chips so that 2, 4, or even 8 of them can easily be placed on a single board for customers who want larger systems. Moreover, to increase income, some companies will package the identical designs differently so that only the higher-priced products can be used in multisocket systems.

To help with programming such a multisocket system, mainstream architects offer a single address space with cache coherence to make sure all the copies of data in each cache are consistent.

Note that these multisocket systems can also help address the first challenge mentioned above. For example, suppose the maximum amount of memory per socket is 128 GB. A customer desiring a terabyte (1024 GB) would be forced to buy an 8-socket system, even if the main purpose of the other CPU chips was for connectivity to more DRAMs rather than for more performance. As mainstream computer manufacturers don't usually make DRAMs, there is little downside in forcing customers to buy more CPU chips when they want to buy a lot of DRAM chips.

Thus far, GPUs have not been designed to make it easier to build multisocket systems nor to program them; perhaps future GPUs will.

---

[9] Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," AFIPS Conference Proceedings (30): 483–485, 1967.

Conclusion

We are in entering an exciting decade where the fundamentals of architecture are being re-examined. What will the microprocessor of 2020 look like? It turns out I already made such a prediction in 1995, so let's revisit it [10]:

> *"I have been asked to describe the microprocessor of 2020. Such predictions in my opinion tend to overstate the worth of radical, new computing technologies. Hence, I boldly predict that changes will be evolutionary in nature, and not revolutionary. … I do not think the microprocessor of 2020 will be startling to people from our time… Pipelining, superscalar organization and caches will continue to play major roles in the advancement of microprocessor technology, and if hopes are realized, parallel processing will join them. … If parallel processing succeeds, this sea of transistors could also be used by multiple processors on a single chip, giving us a* micromultiprocessor*. … Looking ahead, microprocessor performance will easily keep doubling every 18 months through the turn of the century. After that, it is hard to bet against a curve that has outstripped all expectations."*

Although the term micromultiprocessor didn't catch on, the predictions about the nature of microprocessors today and mainstream performance through the year 2000 held true. Performance per year for mainstream computing dropped off around 2003, awaiting programs that can reliably turn the increasing number of processors per chip into higher performance and architectures that can make it easier to deliver on that potential.

We are clearly beginning to see convergence from the different camps on the road to the 2020 microprocessor, which will surely be highly parallel. I'm looking forward to traveling down that road to see where it takes us.

I'd like to congratulate NVIDIA and the Fermi team on taking a giant step towards making GPUs attractive for a broader class of programs. I believe history will record Fermi as a significant milestone along the road to 2020.

---

[10] David Patterson, "Microprocessors in 2020," *Scientific American*, 150th Anniversary Edition, vol. 273, (no. 3): 62-67, September 1995.