High Performance Computing

August 20, 2008

CUDA, Supercomputing for the Masses: Part 7

- Email
- Print
- Reprint

add to:

- Del.icio.us
- Slashdot
- Digg
- Y! MyWeb
- Google
- Blink
- Furl

Rob Farber

Double the fun with next-generation CUDA hardware

Rob Farber is a senior scientist at Pacific Northwest National Laboratory. He has worked in massively parallel computing at several national laboratories and as co-founder of several startups. He can be reached at rmfarber@gmail.com.

In Part 6 of this article series on CUDA (short for "Compute Unified Device Architecture"), I examined global memory using the CUDA profiler. In this installment, I take a look at some next-generation CUDA hardware.

CUDA and CUDA-enabled devices are co-evolving to deliver more performance and capability with each new generation. NVIDIA's recent introduction of the GeForce 200-series and Tesla 10-series of products, shows the rapidity of this evolution as roughly twice the hardware capability is now available at the same price point of the previous line of products plus the 200-series includes the addition of some valuable (and potentially indispensable) new features.

The 1.4 billion transistors in the GTX280 -- as opposed to the 690 million transistors in the original G80 -- clearly illustrates the 2-for-1 approach NVIDIA has taken with this next generation of CUDA hardware. My experience shows that NVIDIA used these transistors very well indeed in the 200-series. Without changes, my single-precision codes run roughly 2x faster on the new hardware when compared against the previous generation G80 hardware!

The following list summarizes the important new features and capabilities of the 200-series architecture:

- The hardware now supports double precision arithmetic (there are 30 64-bit floating point units in the GTX280).
- Global memory is larger, faster, and easier to use. The coalescing rules have been relaxed -- which makes it easier to achieve high global memory performance -- and global memory bandwidth -- at over 100 GB/s -- is nearly double that of the G80 architecture.
- The number of single-precision registers has doubled per hardware thread (although the number of available registers has not changed relative to the previous architectures when using double-precision).
- The 200-series supports 32-bit atomic signed and unsigned integer functions in shared memory and 64-bit functions in global memory.
- The 200-series now includes warp vote functions.
- The number of thread processors has nearly doubled -- 240 as opposed to 128 -- and supports a greater number of active warps and active threads per multiprocessor.
- The 200-series has an enhanced hardware ability to perform a MAD and a MUL at the same time, which should help some applications better approach peak performance.

Let's take a closer look at what this all means for CUDA.

For many applications, the biggest new feature is hardware double-precision arithmetic. The speed and massive parallelism of the NVIDIA GPUs has brought supercomputing to the masses. A realization that quickly occurs when working with super-sized problems and data sets is that numerical noise can quickly accumulate (due to floating-point inaccuracies) and cause garbage results. Physical simulations, for example, can suddenly exhibit spectacular non-physical behavior and previously working simulations can become unstable and start generating infinity, NaN, or other nonsense values. Although not necessary for all computing problems, the use of higher precision floating-point representations (such as 64-bit double-precision floats) can really help. (In a future article, I'll look at the ways you can combine fast single precision with double-precision calculations to speed up your results.)

The 200-series architecture is the first to include hardware double precision. As might be expected with any first generation product, there is some room remaining for performance improvement (since thread-processors in a multiprocessors all share a single double-precision hardware unit). My experience indicates a minor couple of percent decrease in performance occurs when using double-precision so long as it is only used where required to preserve numerical accuracy. This is, of course, problem dependent and your mileage may vary.

Most programmers will find it much easier to attain high performance with the 10-series architecture.

By doubling the number of registers, NVIDIA has made it easier for the CUDA programmer to load sufficient single-precision data into the registers to reduce (or possibly) eliminate much of the bottleneck imposed by global memory. Since double-precision values require twice the storage as single-precision (8 bytes versus 4 bytes), the number of double-precision registers on the 200-series is identical to the number of single-precision registers on the G80 and G92 architectures.

The bandwidth of global memory has also nearly doubled.

The 10-series boards provide over 100 GB/s of global memory bandwidth. For 32-bit floats, global memory bandwidth lost ground in the 10-series relative to the G80 and G92 architectures, as seen in Table 1, because the number of thread processors nearly doubled.

Architecture / card	Global Memory Bandwidth (GB/s)	Number of processing units	Million 32-bit operands available per processing unit per second from Global Memory (considering bandwidth only)
G80 / 8800 Ultra	104	128	203
10-series/ GTX 280	141	240	146

Table 1

Double-precision (64-bit floats) performance is complicated by the fact that there is one double-precision unit per multiprocessor (8 thread processors) and that doubles consume twice as much bandwidth to move around. CUDA developers on the 200-series may discover that an application that is (marginally) bandwidth limited when using single-precision floats could become compute limited after switching to double-precision.

This table does demonstrate that the NVIDIA hardware design team did an excellent job because they delivered a new product with nearly twice the global memory bandwidth plus double-precision capability for the same price-points as the previous generation products.

A big win for all CUDA applications is the fact that global memory is easier to access in a high performance manner due to the relaxation of the coalescing rules in the 200-series. Section 5.2.2 of the CUDA Programming Guide goes into more detail on the protocol used to issue memory transactions. Following are three characteristics that are well worth highlighting:

- Coalescing is achieved for any pattern of address requests including multiple requests to the same address. Previous architectures needed to access words in sequence. In essence, this means that many more CUDA applications will get excellent global memory performance, and that CUDA programmers will no longer have to invent (if possible) workarounds to get their global memory access patterns "just right".
- If a half-warp addresses words in n different segments, then only n memory transactions are issued. If **n=2**, for example, only two memory transactions would be issued as opposed to the 16 transactions that would occur in previous architectures (e.g. 4x fewer transactions).
- Unfortunately, unused words in a memory are still read, so they waste bandwidth even though the hardware will issue the smallest memory transaction possible.

The 200-series supports atomic operations for signed and unsigned integer. The exception of **atomicExch()**, which is also supported for single-precision floating-point numbers. An atomic function performs a read-modify-write atomic operation on one 32-bit or 64-bit word residing in global or shared memory. For example, **atomicAdd()** reads a 32-bit word at some address in global or shared memory, adds an integer to it, and writes the result back to the same address. The operation is atomic in the sense that it is guaranteed to be performed without interference from other threads -- no other thread can access this address until the operation is complete.

Warp vote functions available in the 200-series. If you need them, they provide an indispensable capability to perform fast predicate operations (e.g., checking of a condition is true or false) across all the threads of a warp:

```
int __all(int predicate);
```

Evaluates the specified predicate for all threads of the warp and returns non-zero if and only if predicate evaluates to non-zero for all of them.

```
int __any(int predicate);
```

Evaluates specified predicate for all threads of the warp and returns non-zero if and only if predicate evaluates to non-zero for any of them.

Finally, improvements to the ability of the hardware to perform simultaneous MAD and MUL operations should help some applications achieve FLOP (floating-point operations) rates closer to peak performance.

For further information, look at the CUDA Zone forums. I also recommend downloading the latest version of the CUDA Programming Guideon the NVIDIA website. The current version is 2.0b2, which includes discussion of the new features and API.

This may be an excellent time to upgrade to the 200-series of CUDA-enabled devices. Fierce competition (http://www.tgdaily.com/content/view/38243/135) has forced some significant pricing adjustments, which might make this the perfect time to shop for a deal!

For More Information

- CUDA, Supercomputing for the Masses: Part 9
- CUDA, Supercomputing for the Masses: Part 8
- CUDA, Supercomputing for the Masses: Part 7
- CUDA, Supercomputing for the Masses: Part 6
- CUDA, Supercomputing for the Masses: Part 5
- CUDA, Supercomputing for the Masses: Part 4
- CUDA, Supercomputing for the Masses: Part 3
- CUDA, Supercomputing for the Masses: Part 2
- CUDA, Supercomputing for the Masses: Part 1

Click here for more information on CUDA and here for more information on NVIDIA.

Courtesy of: www.ddj.com