Ultra-Performance Pascal GPU and NVLink Interconnect

This article introduces Nvidia's latest high-performance Pascal GPU. GP100 features in-package high-bandwidth memory, support for efficient FP16 operations, unified memory, and instruction preemption, and incorporates Nvidia's NVLink I/O for high-bandwidth connections between GPUs and between GPUs and CPUs.

Denis Foley
John Danskin
Nvidia

GP100 is a GPU optimized for acceleration of highly parallel computation. GPUs can more generally be described as throughput-optimized processors. In contrast to traditional CPUs, or latency-optimized processors, which are designed to execute computations with many serial dependencies at maximum speed, throughput-optimized processors are designed to execute computations with few serial dependencies with maximum efficiency, measured by either system cost or energy to solution. In 2012, the Titan supercomputer at Oak Ridge National Laboratory used a previous-generation Nvidia GPU to reach number 1 on the Top500 supercomputer list.

Pascal and GP100

GP100 is the first chip from Nvidia to use the Pascal architecture. At 610 mm^2, GP100 is the largest chip that Nvidia has built. The chip is fabricated in TSMC's 16-nm fin field-effect transistor (FinFET) process. The design features six graphics processing cores, each with 10 streaming microprocessors (SMs), for a total of 60 SMs, four of which are spares for yield, leaving 56 active SMs. GP100 has 4 Mbytes of L2 cache and more than 14 Mbytes of register file.

GPUs allocate registers to threads flexibly. A larger register file allows either more simultaneously active threads or more registers per thread. More active threads allow for greater latency tolerance. With more registers per thread, threads can block more data into what is effectively a high-bandwidth scratchpad. Based on the Maxwell architecture, GM200 (as used on the M40 module) was Nvidia's previous flagship high-performance computing chip. Relative to GM200, in GP100, 32-bit floating-point (FP32) throughput per clock per SM was cut by half, SM count increased 230 percent, and register file capacity per SM stayed the same at 256 Kbytes per SM. The effect is to double the total register file to floating-point throughput ratio, while increasing total floating-point throughput per clock by 15 percent. This change in balance makes it easier for programmers to achieve high efficiency on codes that can benefit from increased thread count or per-thread register count. When this increased FP32 throughput is combined with a 30 percent increase in clock rate, overall single-precision throughput relative to GM200 increases by approximately 50 percent to 10.6 Tflops. Unlike GM200, which primarily targeted single-precision

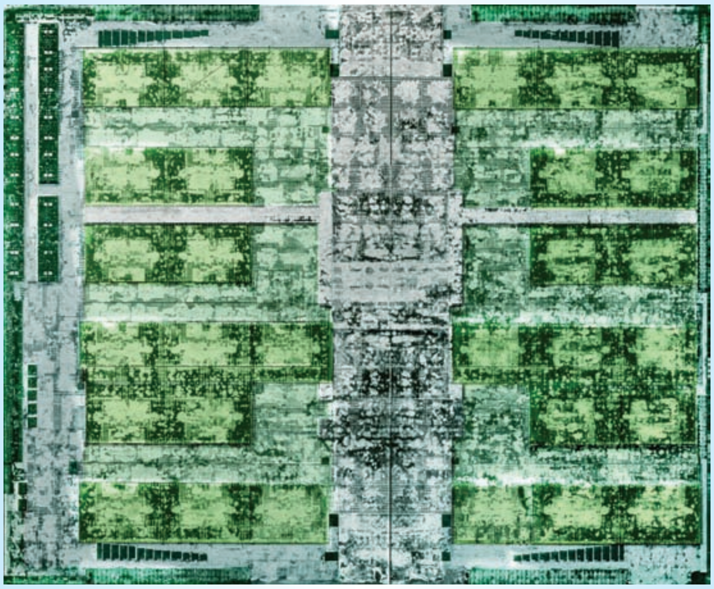
Figure 1. GP100 die photo. High-bandwidth memory (HBM) I/Os are on the top and bottom, and NVLink and another I/O are on the left.

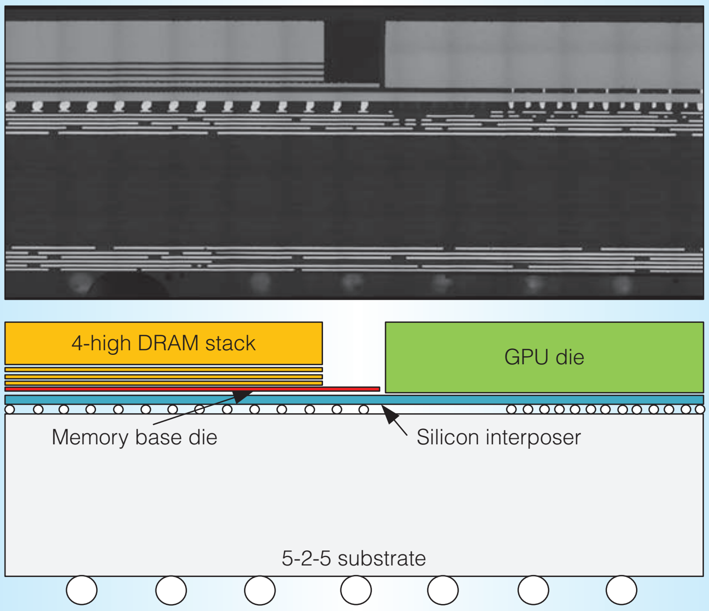

Figure 2. Cross section showing HBM die stack and GPU die. Note the thickness of the uppermost DRAM die in the stack relative to the other DRAM die.

floating-point loads, GP100 supports double precision at half the single-precision rate, or 5.3 Tflops.

GP100 supports 4,096 bits of high-bandwidth memory (HBM) I/O. The design has x16 PCI Express (PCIe) Gen3 and four x8 NVLinks. Figure 1 shows a photo of the GP100 die.

The HBM graphics memory uses a 3D stacking technique. A stack of four DRAM dies and a single memory controller base are assembled using through-silicon via technology. Four of these stacks and the GP100 die are placed on a silicon interposer. GP100 uses a chip-on-wafer-on-substrate process to mitigate warpage of the thin silicon interposer. The silicon interposer is placed on an organic substrate as illustrated in Figure 2. The substrate uses a 5-2-5 buildup (five routing layers above and below a two-layer core) with a thick 1,200-μm core with a low coefficient of thermal expansion. A stiffener ring is used rather than a lid for improved thermals. The final packaged part is 55 mm × 55 mm. The HBM stacks and the GPU die are tested and known good prior to assembly, whereas the substrate is visually inspected but not tested prior to assembly. Note that the top DRAM in the stack is thicker than the others. The HBM stack and the GPU die are ground to the same height. If an HBM stack with eight DRAMs were used in future versions of the product, the extra dies could be accommodated in the same stack height. Figure 3 shows a close-up of the GPU die, silicon interposer, and top layers of the package.

Figure 4 shows the final packaged part and shows how close the GPU die and HBM stack are to each other. This proximity allows for the use of low-power drivers and a wide memory interface, resulting in significant power savings and increased memory bandwidth relative to prior GPUs. The 4,096-bit-wide interface supports a peak memory bandwidth of 720 Gbytes per second (GBps), or three times the bandwidth of prior implementations at the same power. The dense packaging also results in significant area savings on the GPU printed circuit board (PCB). HBM memory capacity is 16 Gbytes.

In addition to standard PCIe form factor GPU PCBs, GP100 introduces a new module form factor called SXM2. The dense packaging lets the GPU, with its attached memory and voltage regulators, fit on a 78 mm × 140 mm mezzanine card for computing applications. The card supports x16 PCIe Gen3, four NVLinks, and a small number of control/clock signals. Figure 5a shows the top side of the SXM2 module, and Figure 5b shows the bottom side. The SXM2 module has two 400-pin connectors to the GPU baseboard that carry all the high-speed SerDes control and power connections. Shorter distances from the voltage regulators to the GPU mean that power losses due to IR drop on the PCB are reduced, making more of the 300-W module limit available for the GPU core. The small form factor lets us place several of these SXM2 modules close together and connect them using NVLink. We refer to the SXM2 GPU module as P100.

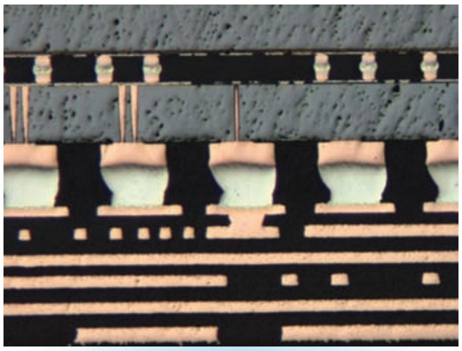
Figure 3. Cross section of the packaged GPU showing the GPU die (on top) connected to the silicon interposer, and all mounted on a multilayer organic package.

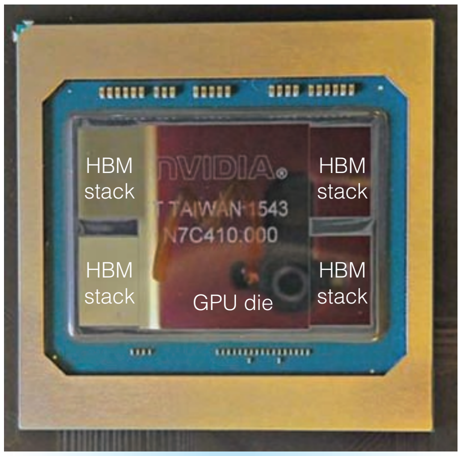

Figure 4. GP100 packaged part. The GPU die is surrounded by four stacks of HBM memory.

Deep Learning

Deep learning is a breakthrough technique in neural networks. Neural networks with many—sometimes tens—of hidden layers are trained to perform tasks traditionally associated with artificial intelligence, including image classification, speech recognition, language translation, and autonomous vehicle control.[1,2] Deep learning is also a key component in the National Cancer Institute Moonshot Initiative.[3] Nvidia sees deep learning as a transformative computing technology. The computations required for training neural networks are highly parallel and can mostly be expressed as matrix operations, which can be implemented efficiently on GPUs.

Although it can be relatively cheap to use a deep learning network, training the network can be expensive. In 2012, Quoc V. Le and colleagues used a cluster of 1,000 16-core CPUs for three days to train a 1-billion-parameter image-recognition network.[4] Jeffrey Dean and colleagues used a cluster of up to about 10,000 CPU cores for 25 hours to train a 1.7-billion-parameter network.[5] Many researchers do not have access to hundreds or thousands of CPUs for days at a time. The time spent to train the same networks using a modest number of CPUs would be prohibitive, especially considering that designing a new network could require dozens of iterations.

In 2013, Adam Coates and colleagues used 16 Nvidia Kepler architecture GeForce GTX 680 GPUs[6] capable of 3 single-precision Tflops each to train an 11-billion-parameter network—6.5 times larger than Dean and colleagues' network—in just three days.[7] The GPU network solved a much larger problem in about the same time as the (more than 50 times larger) CPU network, probably partly due to reduced network overhead.

Scaling up from that 3-Tflops Kepler GPU, the GM200, released in 2015, achieved 6.8 Tflops. In 2016, GP100 achieved 10.6 single-precision Tflops. In addition, because most of the math required to train neural nets can be executed in 16-bit half-precision floating-point (IEEE-754 binary 16), GP100 adds half-precision floating-point instructions that

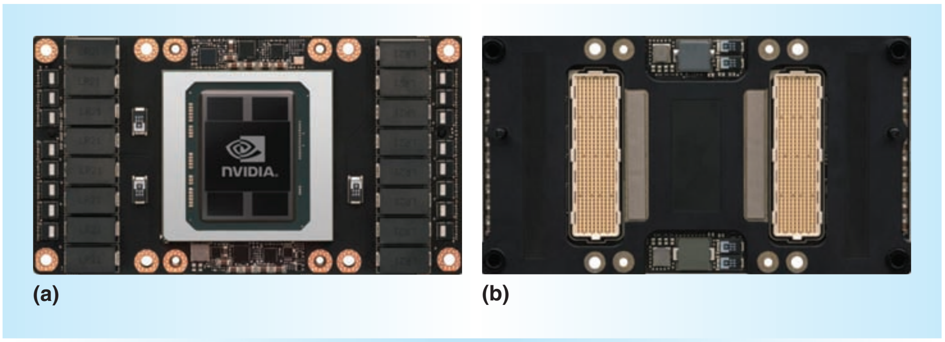

Figure 5. P100 SXM2 module. (a) The top side shows the GPU package and regulator components. (b) The bottom side shows the connectors and some additional regulator circuitry.

Table 1. Feature comparison between M40 and P100.

Features	Tesla M40	Tesla P100
GPU/architecture	GM200/Maxwell	GP100/Pascal
Streaming multiprocessors (SMs)	24	56
FP16 flops/clock/SM	—	256
FP32 flops/clock/SM	256	128
FP64 flops/clock/SM	8	64
GPU base clock	948 MHz	1,328 MHz
GPU boost clock	1,114 MHz	1,480 MHz
Peak FP16 Tflops*	—	21.2
Peak FP32 Tflops*	6.8	10.6
Peak FP64 Tflops*	0.2	5.3
Memory interface	384-bit GDDR5	4,096-bit HBM2
Memory size	Up to 24 Gbytes	16 Gbytes
L2 cache size	3,072 Kbytes	4,096 Kbytes
Register file size/SM	256 Kbytes	256 Kbytes
Register file size/GPU	6,144 Kbytes	14,336 Kbytes
TDP (board power)	250 W	300 W
Transistors	8 billion	15.3 billion
GPU die size	601 mm^2	610 mm^2
Manufacturing process	28 nm	16-nm FinFET

*Peak Tflops figures in this table are based on boost clocks. P100 targets both double- and single-precision floating-point applications, whereas M40 has reduced double-precision throughput.

run at twice the speed of single precision, increasing the effective math throughput to 21.2 Tflops, or more than three times the performance of GM200. In addition, 16-bit floating point cuts bandwidth and storage requirements in half. GP100 completes the 16-bit floating-point (FP16) instruction set with 16-bit atomics, such as atomic ADD, MIN, and MAX, which allow lockless cooperation between threads on the GPU.

The deep learning benchmark situation is fluid, making direct comparisons difficult, but the Nvidia DGX-1 system based on eight P100 GPUs with NVLink reduced training time on the Caffe AlexNet benchmark by a factor of four compared to the

Maxwell-based M40 on the same generation of software. This is 2:1 faster per GPU including scaling,[8] thanks to a combination of larger caches, 3 times the memory bandwidth, and 1.5 times the single-precision floating point. Table 1 shows a comparison between the M40 and P100.

Why are GPUs so efficient at deep learning? Massive floating-point capability is only part of the equation. With thousands of threads, GPUs have the latency tolerance to saturate memory bandwidth without stalling computation. GP100's 14 Mbytes of register file allow explicit blocking of data for efficient matrix multiplication.[9]

Programmability

The primary goal for a GPU architect is to make it economically and practically feasible to carry out computations that were not feasible before. The cost of computation has many dimensions, such as hardware acquisition, power, and space, but the cost of software development should not be minimized. As we will discuss, GP100 improves programmability by increasing the register file size to make it easier to hide memory latency, adding fine-grained preemption, and, most dramatically, making it easier to share pointers between programs running on CPU and GPU.

Latency Tolerance

GPUs have large register files and small caches. GP100 has 14 Mbytes of register file and only 4 Mbytes of L2 cache. This inversion relative to CPUs has several benefits. Because GPUs are latency tolerant, register files can be compact, single-ported RAMs, which are far more power and area efficient than the multiported RAMs used in CPUs and even the associative tag RAMs used in CPU and GPU caches. Register files are the cheapest bulk storage on the GPU.

Although a CPU with only a few threads essentially stops when it has a cache miss, massively parallel GPUs can tolerate frequent cache misses. Latency tolerance is, however, finite, and the high-throughput memories and floating-point units of GPUs can easily become latency bound. Depending on the code characteristics, programmers can be forced to restructure codes to aggressively overlap floating-point and memory operations. Increased latency tolerance reduces this programmer burden.

GP100 doubles the register file to the floating-point throughput ratio of GM200. GP100 has 256 Kbytes of register file per SM. In essence, register file space is allocated to threads according to their register count requirements. Thread count is limited to the quotient of register file space and per-thread requirements. For threads requiring 32 single-precision registers, an SM with 256 Kbytes of register space can run 256 Kbytes/(32 registers/thread * 4 bytes/register) = 2,048 threads. With 56 SMs and 2,048 threads per SM, the total thread count would be 114,688. If each thread can issue four nondependent loads, that's 1.8 Mbytes of outstanding memory references. With memory throughput of 720 GBps, from Little's law we have 2 microseconds of latency tolerance, which is more than enough for even a GPU memory system with its massive page sorting buffers for efficiency.

Increasing thread count is one technique for increasing latency tolerance. Flexible allocation of registers to threads also lets threads use up to 255 registers per thread, which allows for significant loop unrolling as well as local data blocking.

GP100's increased latency tolerance increases both performance and programmability.

Instruction-Boundary Preemption and Restoration

CPUs have had this feature for a long time, but GPUs have a different heritage. In early CPUs, preempting a core required saving approximately 16 to 20 registers, including state such as condition codes. CPUs use low-latency caches to avoid latency penalties, and register files have shrunk relative to memory bandwidth. Meanwhile, GPUs use register files for buffering to cover memory latency, which means that they grow proportional to the product of memory bandwidth and latency (Little's law). This implies that the absolute time to save an entire GPU register file decreases slowly over time. GP100 has approximately 14 Mbytes of register file state, not including shared memory scratchpads and other state. This memory hierarchy

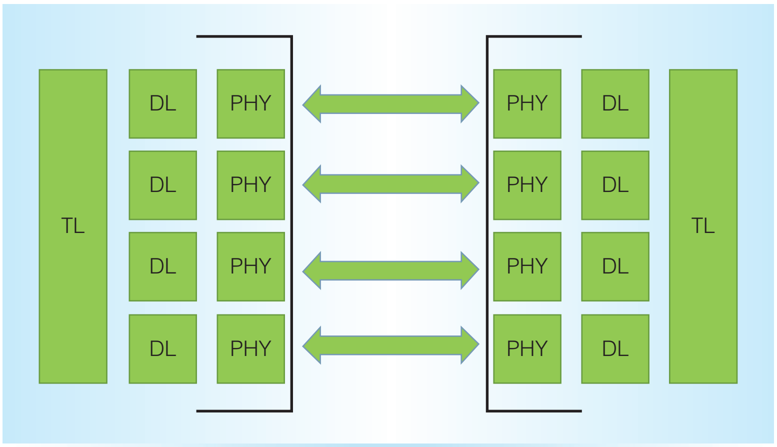

Figure 6. NVLink physicals showing the four NVLinks between two GPUs and the layering of the protocol.

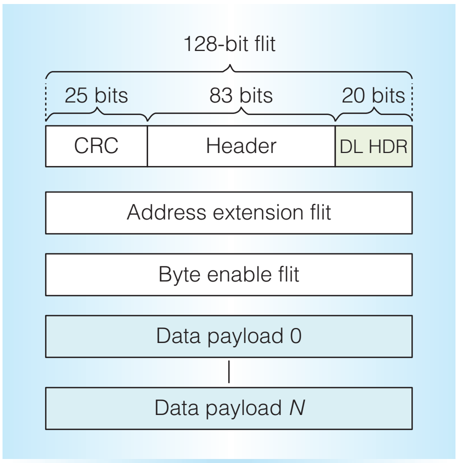

Figure 7. Packet format for a write transaction with header, address extension (AE), byte enable (BE), and data flits. (CRC: cyclic redundancy check.)

inversion has made preemption less attractive for GPUs than for CPUs.

GPUs also have hardware task scheduling at multiple levels. Grids are groups of thread blocks that can have explicit dependencies. Thread blocks are groups of threads that are guaranteed to run together on an SM so that they can cooperate. Thread blocks are divided into 32 thread warps that execute the same instruction together.[10] State for this task management is live inside the GPU during execution, which complicates preemption.

Prior to GP100, Nvidia GPUs supported nondestructive pre-emption between thread blocks, which are arrays of up to 1,024 threads running on a single SM. This coarse-grained preemption meant that a single task could monopolize a GPU indefinitely or until killed.

GP100 adds the capability to preempt and restore computing programs on instruction boundaries. Instruction-level preemption allows virtualization with quality-of-service expectations and allows debuggers to interrupt and restart code dynamically.

Unified Memory

GP100 makes it easier for CUDA programs to manage memory and data, especially sharing between CPU and GPU. The issue is that the CPU and GPU have separate memory systems with no hardware coherence protocol. The memory systems also have vastly different performance characteristics, and neither processor can efficiently access the other processor's memory. In addition, for GPUs before GP100, page faults are nonrecoverable errors.

In CUDA 6, which can run on GPUs earlier than GP100, memory accessed by the GPU is explicitly allocated through CUDA. Programs running on the CPU and GPU can access this memory without explicit data movement when the CPU and GPU processes aren't running together, but the GPU driver must guarantee that all pages that the GPU could access are present in GPU memory while the GPU is running. The main issue with this strategy is that all memory accessed by the GPU must be specially allocated. If the CPU side of a CUDA library is passed data that isn't provably in CUDA managed memory, it has to make a copy in the managed memory before it can pass the data to the GPU. A secondary issue is that all of the managed memory must be physically present on the GPU whenever GPU code is running, whether it is used or not.

GP100 adds a page-fault stall capability. When a thread running on GP100 faults, it stalls until the page table can be updated. As there can be tens of thousands of threads, most of the GPU can usually continue to make progress while the page fault is serviced. This new ability lets us page data into the GPU lazily, which means that it is no longer necessary for the CUDA driver to track which pages the GPU might access. CUDA libraries no longer have to make explicit copies of data from unknown sources. The driver and operating system can also cooperate in page-table-based optimizations, such as multiple read-only copies of pages.

GP100 also increased the virtual address space from 47 bits (128 Tbytes) to 49 bits (512 Tbytes). This is sufficient to support all of the major CPUs and operating systems we know about.

NVLink

NVLink is designed as a high-bandwidth GPU↔GPU and GPU↔CPU interface. It supports load/store semantics that let programmers directly read or write local graphics memory (GMEM) and peer GMEM or the CPU's memory (SYSMEM) using NVLink, all in a common shared address space. Programmers can also target atomic operations to peer GPUs, where they can be completed at the destination. NVLink lets us cluster GPUs or GPUs and CPUs and start to consider them as a single larger computing element.

NVLink is a bidirectional interface. Each NVLink comprises eight differential pairs in each direction for a total of 32 wires. Data is sent at up to 20 Gbits per second, yielding a peak bidirectional bandwidth of 40 GBps for a single NVLink. GP100 supports four NVLinks for a total of 160 GBps of bidirectional bandwidth (see Figure 6). An embedded clock is used, requiring the use of a clock data recovery circuit at the receiver. The interface has 85 ohm differential termination and is DC coupled. Lane reversal and polarity inversion are supported to ease routing. The bit error rate is specified as better than 1 in 1e12. A 25-bit cyclic redundancy check (CRC) is used to detect data errors.

NVLink packets vary in length from a single 128-bit flit up to a maximum of 18 128-bit flits to support a 256-byte transfer. An NVLink transaction comprises at least a request and a response (the exception being that posted operations don't have a response packet) and optionally an address extension (AE) flit, a byte enable (BE) flit, and between 0 and 16 data payload flits.

A request header flit comprises the 25-bit CRC; an 83-bit transaction layer field with request type, address, flow control credits, and tag identifier; and a 20-bit data link (DL) layer field including acknowledge identifier, packet length information, and application number tag. The AE flit contains information that should be relatively static from request to request (sticky bits),

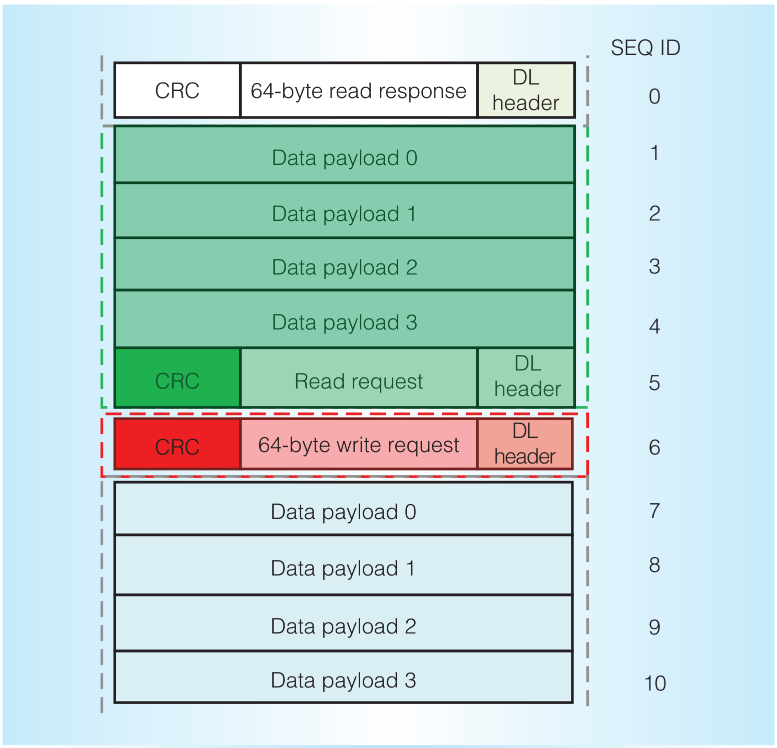

Figure 8. CRC positioning showing how CRC is calculated over the header and previous data payload.

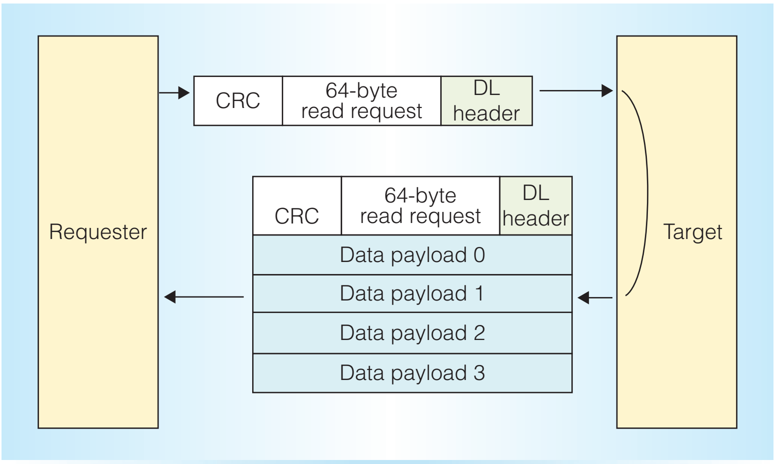

Figure 9. Simple 64-byte read request from the requestor followed by a read response from the target.

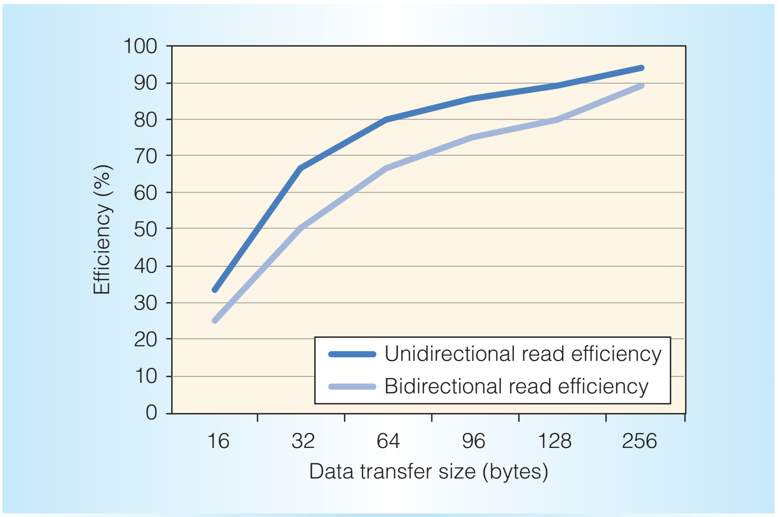

Figure 10. Unidirectional and bidirectional read efficiency showing the roll-off in efficiency as payload size decreases.

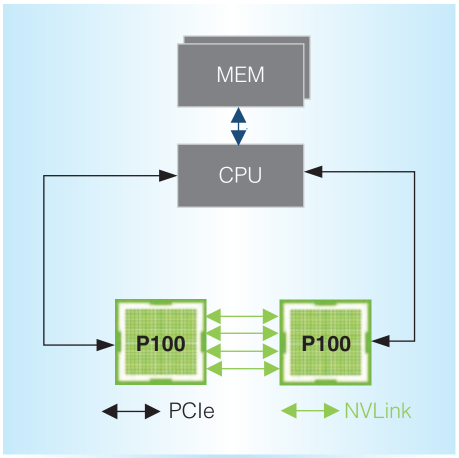

Figure 11. A four-NVLink gang between GPUs providing 160 Gbytes per second (GBps) of data bandwidth between the GPUs.

information that is command-specific, or information that alters the default value for a command type. Sticky information is transmitted when it changes and is stored on the receiver side for reuse for non-AE packets. A BE flit is used for write or atomic commands, and 128 enable bits indicate the data bytes to be written on writes up to 128 bytes. BE cannot be used for 256-byte transfers. Figure 7 shows a packet for a write transaction with a header flit, AE flit, BE flit, and data flits.

CRC

The selected CRC allows the detection of 5 random bits in error in the largest transmitted packet or a burst of up to 25 sequential bit errors on a single differential pair. Packets are stored in a replay buffer. Each packet has a sequence ID. When a packet is received with a good CRC, a positive acknowledgment is sent back to the source. In the event that the source does not receive an acknowledgment within a specified time, a replay sequence is initiated and the packet in error and all subsequent packets are retransmitted.

Packet length is variable, and the length information is conveyed as part of the DL header. Since the header contains packet-length information and the protocol contains no framing symbols, the CRC covering the header must be checked prior to parsing the rest of the packet. Rather than requiring separate CRC fields for the header and for the data payload, the CRC is calculated over the header and the previous payload (see Figure 8). Flits with sequence ID 1:4 are associated with the previous header—the 64-byte read response in flit 0. The CRC for the read request in flit 5 is calculated over flits 1:5, whereas the CRC for the 64-byte write request (flit 6) is calculated solely over flit 6, because there is no previous payload. Flits 7:10 would be covered by a CRC in flit 11 (not shown). If there is no request or response ready to go in flit 11, the CRC is conveyed associated with a null transaction.

Efficiency

A posted write of a maximum-size (256-byte) packet without an AE flit takes 17 flits—one for the header and 16 data payload cycles. A link efficiency of 94.1 percent can be sustained. Backing off to a 128-byte packet drops the efficiency to 88.9 percent. Posted writes consume bandwidth in one direction only. There is no response required. The upstream indication that the packet was received successfully (the ACK) can be carried by an unrelated upstream packet and does not incur any additional overhead.

A nonposted 256-byte write, however, requires both the 17 flits in the downstream direction and a write response in the upstream direction. This doesn't impact the

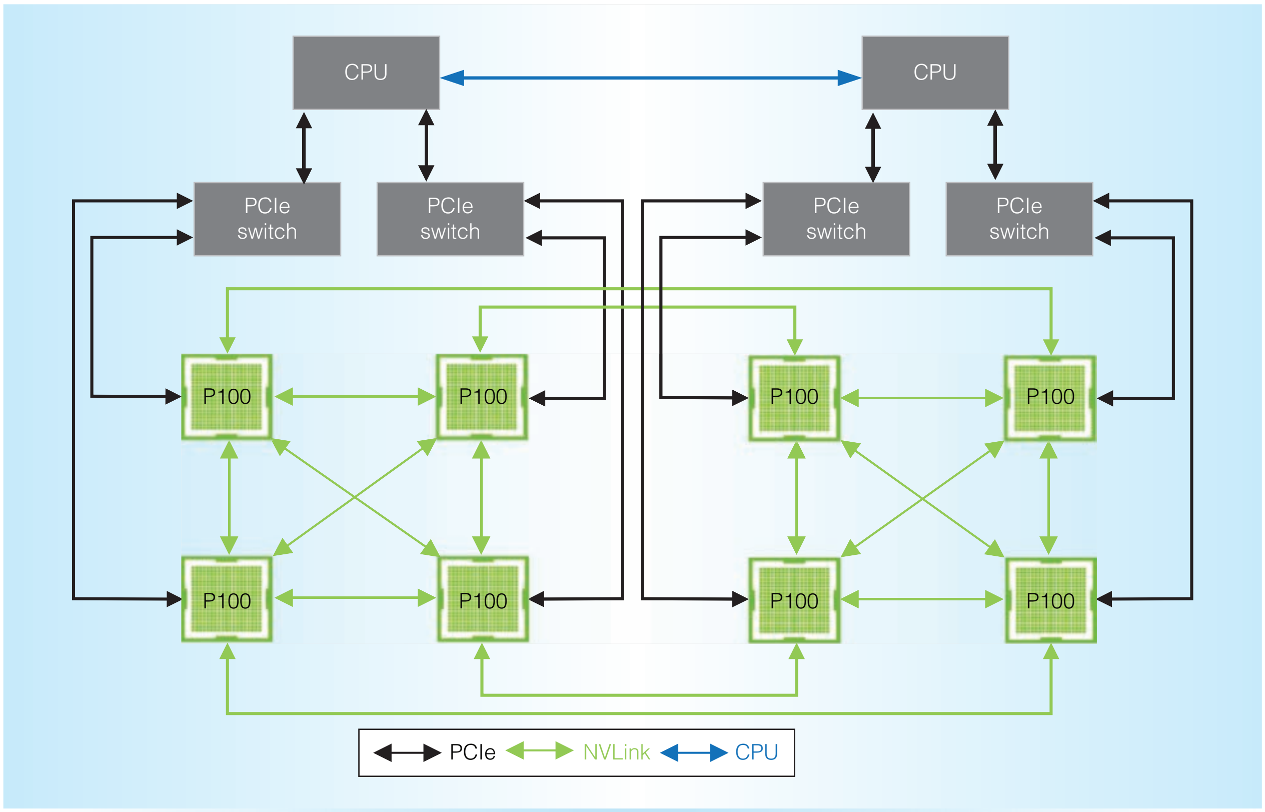

Figure 12. Hybrid Cubed Mesh. Each GPU has an NVLink connection to four other GPUs.

bandwidth in the downstream direction but will impact the upstream traffic. Consider a case in which we have 256-byte nonposted writes going in both directions. The steady state traffic would have 1 write request header flit, 16 data payload flits for the downstream writes, plus the additional write response flit associated with upstream writes, for an efficiency of 88.9 percent. The corresponding number with 64-byte transfers would be 66.7 percent.

A similar situation exists for reads. A 64-byte read request (see Figure 9) comprises the read request header flit in the downstream direction, followed some time later by a read response header flit and four data payload flits in the upstream direction. Sustained 64-byte reads in each direction would result in a link efficiency of 66.7 percent. Note that the minimum read size on NVLink is 32 bytes. A 16-byte or smaller read still has 2 data payload flits. Figure 10 shows the efficiency roll-off as the data payload size decreases for unidirectional and bidirectional reads.

System Configurations

NVLinks can be ganged together to increase available bandwidth between two endpoints or used individually to maximize the number of endpoints. This allows for a degree of flexibility in what systems can be created using the links. A very simple configuration would be two GPUs communicating over a gang of four NVLinks (see Figure 11). In this instance, the NVLink bidirectional bandwidth is 160 GBps—a dramatic increase over the 32 GBps peak bidirectional bandwidth afforded by an x16 Gen3 PCIe peer-to-peer link through a switch.

The configuration selected for Nvidia's DGX-1 Deep Learning Supercomputer is referred to as a Hybrid Cubed Mesh. Consider a cube with a P100 SXM2 module at each corner. The four GPUs on the top face are fully connected with one link to each of the other GPUs. The remaining link for each GPU is used to connect the top and bottom faces of the cube together, as shown in Figure 12.

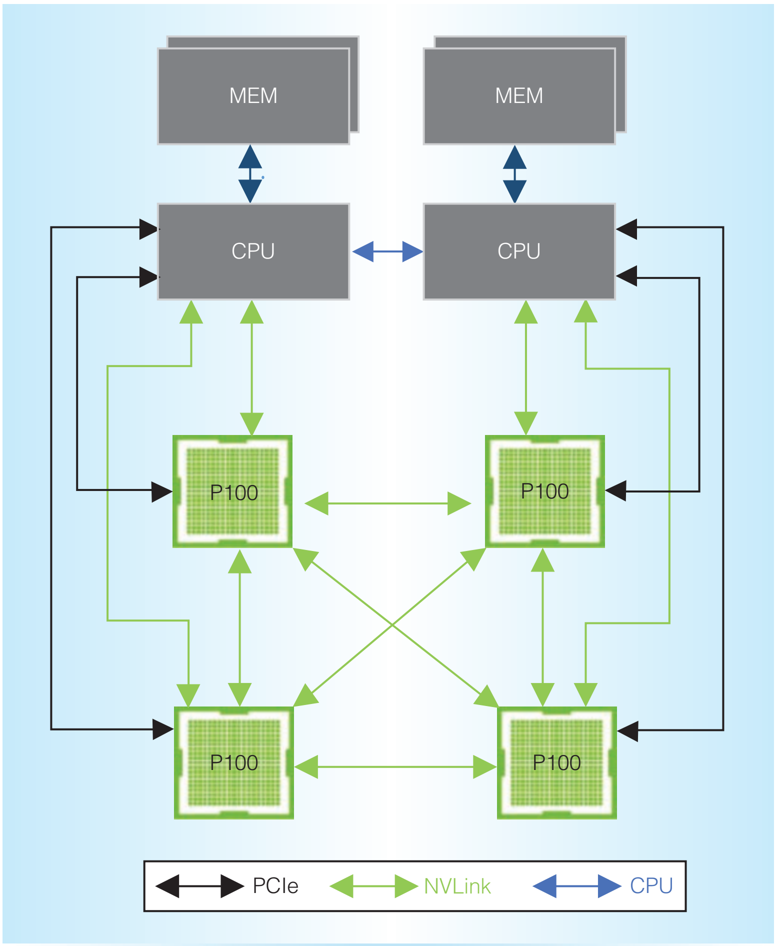

Figure 13. GPU↔CPU connections using NVLink. NVLink can be used for high-bandwidth access to system memory as well as peer memory.

In this configuration, the CPUs are not NVLink enabled. Each GPU is connected to a CPU using a x16 PCIe Gen3 connection through a PCIe switch. Two GPUs share each switch. The CPUs form a two-socket symmetric multiprocessor (SMP) using their own SMP connection.

NVLink can also be used for GPU↔CPU connections allowing for high-bandwidth direct load/store access to the large system memory pool attached to the CPU. Figure 13 shows a two-socket SMP configuration in which each of the CPUs supports two NVLinks. Each GPU has a single link to each of its peers and a single link to one of the two CPUs. In this system, each GPU has up to 20 GBps of read and write bandwidth to and from system memory and 20 GBps of read/write bandwidth to each peer.

Figure 14 shows a system with four NVLinks ganged between a single GPU and a CPU. In this system, the GPU has up to 80 GBps of read and write bandwidth to system memory.

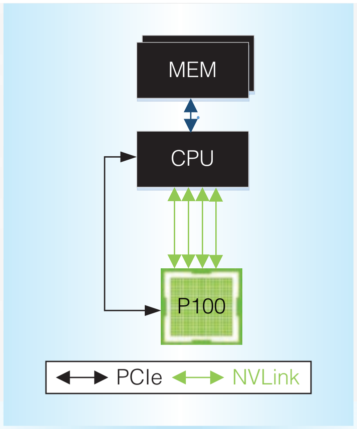

Figure 14. Four NVLinks ganged from GPU↔CPU. NVLinks are ganged to provide 80 GBps per direction access to system memory.

GP100 is Nvidia's highest-performance processor to date. The power efficiency, bandwidth, and density of the in-package HBM—along with new features such as unified memory, FP16 operations, and NVLink—allow Nvidia to produce a machine with unparalleled deep learning capabilities. MICRO

References

1. "Deep Learning," Wikipedia; http://en.wikipedia.org/wiki/deep_learning.
2. C. Chen et al., "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," *Proc. 15th IEEE Int'l Conf. Computer Vision*, 2015, pp. 2722–2730.
3. "Joint Design of Advanced Computing Solutions for Cancer," National Cancer Institute; http://cbiit.nci.nih.gov/ncip/hpc/jdacs4c.
4. Q.V. Le et al., "Building High-Level Features using Large Scale Unsupervised Learning," *Proc. 29th Int'l Conf. Machine Learning*, 2012, pp. 81–88.
5. J. Dean et al., "Large Scale Distributed Deep Networks," *Advances in Neural*

Information Processing Systems, vol. 25, 2012, pp. 1232–1240.

6. *Nvidia GeForce GTX 680*, white paper, Nvidia, 2012.
7. A. Coates et al., "Deep Learning with COTS HPC Systems," *JMLR Workshop and Conf. Proc.*, vol. 28, no. 3, 2013, pp. 1337–1345.
8. I. Buck, "Correcting Intel's Deep Learning Benchmark Mistakes," Aug. 2016; http://blogs.nvidia.com/blog/2016/08/16/correcting-some-mistakes.
9. M. Gebhart et al., "Unifying Primary Cache, Scratch, and Register File Memories in a Throughput Processor," *Proc. 45th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2012, pp. 96–106.
10. "CUDA C Programming Guide," Sept. 2016; http://docs.nvidia.com/cuda/cuda-c-programming-guide.

Denis Foley is a senior director of the GPU Architecture Team at Nvidia, where he leads the team responsible for the architectural development of NVLink. His research interests include high-speed I/O and system design. Foley received a BEng in electrical engineering from University College Cork, Ireland. Contact him at dfoley@nvidia.com.

John Danskin is the vice president of GPU architecture at Nvidia. He is the chief architect for the Tesla lineup, including the Kepler family of GPUs and Pascal. His research interests include GPU architecture, hardware-accelerated deep learning, and scalable computing. Danskin received a PhD in computer science from Princeton University. Contact him at jdanskin@nvidia.com.

myCS Read your subscriptions through the myCS publications portal at http://mycs.computer.org.

2017

IEEE-CS

Charles Babbage Award

CALL FOR AWARD NOMINATIONS

Deadline 15 October 2017

ABOUT THE IEEE-CS CHARLES BABBAGE AWARD

Established in memory of Charles Babbage in recognition of significant contributions in the field of parallel computation. The candidate would have made an outstanding, innovative contribution or contributions to parallel computation. It is hoped, but not required, that the winner will have also contributed to the parallel computation community through teaching, mentoring, or community service.

ABOUT CHARLES BABBAGE

Charles Babbage, an English mathematician, philosopher, inventor and mechanical engineer who is best remembered now for originating the concept of a programmable computer.

CRITERIA

This award covers all aspects of parallel computing including computational aspects, novel applications, parallel algorithms, theory of parallel computation, parallel computing technologies, among others.

AWARD & PRESENTATION

A certificate and a $1,000 honorarium presented to a single recipient. The winner will be invited to present a paper and/or presentation at the annual IEEE-CS International Parallel and Distributed Processing Symposium (IPDPS 2017).

NOMINATION SUBMISSION

Open to all. Nominations are being accepted electronically at www.computer.org/web/awards/charles-babbage. Three endorsements are required. The award shall be presented to a single recipient.

NOMINATION SITE: awards.computer.org

AWARDS HOMEPAGE: www.computer.org/awards

CONTACT US: awards@computer.org