Evolution of the Graphics Processing Unit (GPU)

William J. Dally and Stephen W. Keckler, *NVIDIA Corporation, Santa Clara, CA, 95051, USA*
David B. Kirk, *Independent Consultant*

NSTR - JUST HISTORY.

Graphics processing units (GPUs) power today's fastest supercomputers, are the dominant platform for deep learning, and provide the intelligence for devices ranging from self-driving cars to robots and smart cameras. They also generate compelling photorealistic images at real-time frame rates. GPUs have evolved by adding features to support new use cases. NVIDIA's GeForce 256, the first GPU, was a dedicated processor for real-time graphics, an application that demands large amounts of floating-point arithmetic for vertex and fragment shading computations and high memory bandwidth. As real-time graphics advanced, GPUs became programmable. The combination of programmability and floating-point performance made GPUs attractive for running scientific applications. Scientists found ways to use early programmable GPUs by casting their calculations as vertex and fragment *shaders*. GPUs evolved to meet the needs of scientific users by adding hardware for simpler programming, double-precision floating-point arithmetic, and resilience.

The availability of easily programmed GPUs with high floating-point performance enabled the current revolution in deep learning. AI researchers have found them the ideal platforms to train deep neural networks for vision, speech recognition, natural language processing, recommender systems, and other applications. GPUs evolved to meet the needs of these applications by adding deep-learning-specific data types and instructions. Today, they are the dominant platform for deep learning training and inference.

EARLY 3-D GRAPHICS HARDWARE

The high computational demands of graphics rendering led to the development of special-purpose hardware to carry out the computations needed to generate an image. The Evans and Sutherland Computer Corporation was founded in 1968 to build special-purpose 3-D graphics hardware. Using the small-scale integration technology of the day, these expensive multirack systems were used for demanding applications such as flight simulators.

> *THE AVAILABILITY OF EASILY PROGRAMMED GPUs WITH HIGH FLOATING-POINT PERFORMANCE ENABLED THE CURRENT REVOLUTION IN DEEP LEARNING.*

Moore's law scaling of integrated circuit technology drastically reduced the cost of graphics systems and led them to evolve in two directions. In the late 1980s, Silicon Graphics built a new generation of 3-D graphics systems and high-end professional workstations. At the same time, low-cost 2-D graphics capabilities were supported both in mass market products such as PCs and in video game consoles. Early consoles were sprite-based, copying virtual rectangles to the screen to display backgrounds, scenes, and characters. The earliest PC graphics cards were simply 2-D accelerators and display controllers. They copied pixel values from PC system memory to frame buffer memory, and ultimately to a CRT screen generating addresses and sync signals and providing D/A conversion. 2-D graphics accelerators evolved to support text and window acceleration in hardware, copying, stretching, and blending images (textures) together.

The advent of 3-D video game consoles such as 3DO, Sega Saturn, and Sony Playstation ushered in the era of mass-market 3-D gaming. The desire to support 3-D graphics on PCs for gaming inspired the entire field of 2.5-D and 3-D graphics accelerators, an opportunity so compelling that over 60 companies were founded to address this market. Founded in 1993 by Jensen Huang, Chris Malachowsky, and Curtis Priem, NVIDIA aimed to bring sophisticated 3-D graphics to the

0272-1732 © 2021 IEEE
Digital Object Identifier 10.1109/MM.2021.3113475
Date of current version 19 November 2021.

PC platform and to compete with game consoles. The inspiration for 3-D PC graphics came from the sophisticated and expensive graphics pipelines in 3-D professional workstations such as those created by Silicon Graphics. NVIDIA's founders observed that the larger and more complex chips enabled by Moore's law would allow increased integration and functionality to make PC-based 3-D gaming affordable.

The first generation of 3-D PC graphics cards performed fragment (or pixel) computations of rasterization, color interpolation, texture mapping, Z-buffering (hidden surface removal), and shading. Given the coordinates and parameters of a triangle's vertices in a 2-D screen space, they would generate the depth and color of each pixel in the triangle and composite these pixels into the frame buffer. The vertex computations to transform the vertices from world space to screen space, clip the triangles against the viewport, and compute the lighting of each vertex remained on the host CPU. NVIDIA's first successful product of this type was RIVA-128 (NV3), which was released in 1997.

The evolution of these 3-D accelerators included the ability to blend multiple textures simultaneously for more sophisticated lighting and shading. NVIDIA's RIVA-TNT and TNT2 (NV4 and NV5) had these capabilities, as well as parallel processing of multiple pixels at several points in the pipeline, including rasterization, shading, and at the Z-buffer/memory interface.

GeForce 256: THE FIRST GPU

Introduced in 1999, the GeForce 256 was the first chip to combine the vertex computations for transformation and lighting and the fragment computations on the same chip. This fully integrated graphics engine was named a "graphics processing unit" or GPU. Offloading the vertex computations from the host enabled higher geometric complexity in games at the cost of requiring significant floating-point performance. For example, the perspective transformation requires a 4×4 matrix–vector multiply and a perspective division operation. The vertex and pixel computations in the GeForce 256 were configurable fixed-function hardware, organized in a hardwired pipeline.

As PC games became more sophisticated, demand grew to modify the vertex and pixel computations to achieve more complex graphical effects. Programmable "shaders" had been used in offline software graphics since the 1980s. The GeForce 3 (NV20) GPU introduced a programmable vertex shader in 2001, and in 2002, the GeForce FX introduced programmable fragment shaders.

GPU COMPUTING

As the combination of high floating-point performance and programmability made GPUs with programmable shaders an attractive platform for scientific computing, the field of general-purpose GPU (GPGPU) programming emerged. A GeForce 6 had a peak performance of 108 billion single-precision floating-point operations per second (108 GFLOPS), in contrast to a contemporary CPU that provided only 8 GFLOPS. However, programming these GPUs was challenging. Input and output were very restricted. Fragment shaders only received input from interpolated vertex attributes and textures and only deposited output into the frame buffer. Programmers had to cast their applications as rendering texture mapped and blended triangles to harness the FLOPS of these GPUs. Early GPGPU programs contained pages of inscrutable shader code that forced scientific algorithms into this form.

To harness the increasing capabilities of the hardware, GPUs ultimately leveraged stream processing programming models that hid much of the complexity of shader programming from scientific programmers. Emphasizing parallelism and producer–consumer locality, stream programming systems were originally developed for dedicated stream processors such as Imagine[1] and Merrimac.[2] These systems, however, were ideally suited to GPU programming. The Brook programming language, originally developed for Merrimac, was adapted to GPUs as Brook-GPU.[3] Brook was later adopted by AMD as the language used to program their unified shader GPUs.

The stream processing model influenced the design of GPUs intended for computing. Starting in 2003, Bill Dally, leader of the stream processing project at Stanford, began consulting with NVIDIA working with John Nickolls and Erik Lindholm to transfer stream processing ideas into the design of the GeForce 8 (NV50) GPU. In 2004, Ian Buck, the principal developer of Brook, joined NVIDIA and with many collaborators developed the CUDA programming language.[4]

The GeForce 8 GPU or G80 introduced streaming multiprocessors (SMs) that were used to run both vertex and fragment shaders. These SMs could also run "compute shaders" independent of the graphics pipeline specifically in the form of CUDA cooperative thread arrays. A shared memory facilitated communication between the threads in an array. The G80 "Tesla" GPU and CUDA were launched in 2006, and they greatly reduced the barriers to developing GPU-based scientific and other general-purpose applications.[5,6] Figure 1 shows a die photo of a G80 GPU, which consists of 681 million transistors and delivers 346 GFLOPS of FP32 throughput and 86 GB/s of DRAM bandwidth.

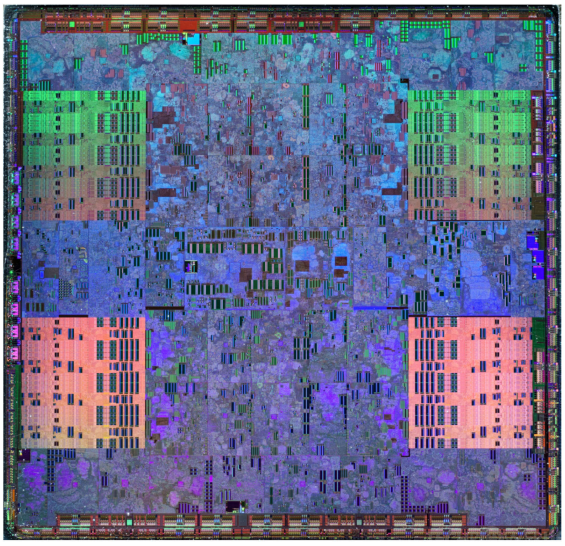

FIGURE 1. NVIDIA G80 GPU launched November, 2006. Photograph courtesy of Fritzchens Fritz.

However, revolutionary hardware and a new programming language do not by themselves make a revolution. Legacy high-performance computing (HPC) applications were primarily written in FORTRAN and were single-threaded serial programs. G80 was a massively parallel threaded processor, and CUDA was a thread- and data-parallel C-like programming language. David Kirk, NVIDIA's Chief Scientist, and Wen-mei Hwu, Professor of Electrical and Computer Engineering at University of Illinois at Urbana-Champaign (UIUC), prototyped a graduate class at UIUC to produce a new generation of parallel programmers. The lecture notes from this class evolved into the popular parallel programming textbook *Programming Massively Parallel Processors*,[7] which has been translated into many languages and printings, and will soon be released in its 4th edition.

At the same time, NVIDIA funded support for CUDA Centers of Excellence at many universities to promote the teaching of CUDA and adoption of GPU technology and parallel programming. Kirk and Hwu also supported many "teach the teachers" events including the annual PUMPS summer school at UPC/BSC in Barcelona, Spain.

Feedback from the large and growing GPU computing community led to the development of HPC-specific features in the Fermi generation of GPUs in 2010. These features included improved support for double-precision floating-point operations, 750 GFLOPS DP in Fermi versus zero in G80, and reliability, availability, and serviceability (RAS) features including ECC on main memory and parity protection for on-chip memories.

With HPC features, high performance, and efficiency, GPUs quickly became the technology of choice for large scientific computers. A major milestone for GPU computing was the Titan system consisting of 18,688 NVIDIA Kepler (K20) GPUs at Oak Ridge National Labs (ORNL), which took the #1 position on the Top 500 list in November 2012 with a Linpack performance of 17.6 PFLOPS. In June 2018, the Summit System at ORNL continued the tradition, taking the #1 position with a Linpack performance of 122.3 PFLOPS, powered by 27,648 NVIDIA Volta (V100) GPUs.[8] Six of the top ten machines on the June, 2021 Top 500 list were powered by NVIDIA GPUs.

THE CURRENT REVOLUTION IN DEEP LEARNING WAS ENABLED BY GPUs , AND FURTHER PROGRESS IN DEEP LEARNING IS LARGELY GATED BY GPU PERFORMANCE.

The success of GPUs in HPC has required the development of a large and diverse software ecosystem, whose maturation took place over the course of a decade. Based on the CUDA programming system, the GPU programming ecosystem now includes other methods of programming GPUs, such as CUDA Fortran and OpenACC. The next level of support is a set of numerical libraries including CuBLAS, CuSparse, and CuFFT that provide highly optimized code for the key functions of many numerical programs. Applications can then leverage these libraries to exploit GPU capabilities. Today, over 600 HPC applications are accelerated by GPUs,[9] including molecular dynamics codes such as GROMACS, NAMD, AMBER, and LAMMPS; weather codes such as WRF; fluid dynamics codes such as ANSYS and OpenFOAM; chemistry codes such as Gaussian, VASP, Quantum Espresso, and GAMESS; and structural analysis codes such as LS-DYNA and ANSYS.

DEEP LEARNING HARDWARE

The current revolution in deep learning was enabled by GPUs, and further progress in deep learning is largely gated by GPU performance. Making deep learning successful has required three ingredients: algorithms, datasets, and hardware. The core algorithms for deep learning: deep neural networks, convolutional networks, training with backpropagation, stochastic gradient descent, and others, had all been developed in the 1980s or earlier. Large datasets such as PASCAL VOC and ImageNet were available by 2005. The missing ingredient was hardware fast enough to train

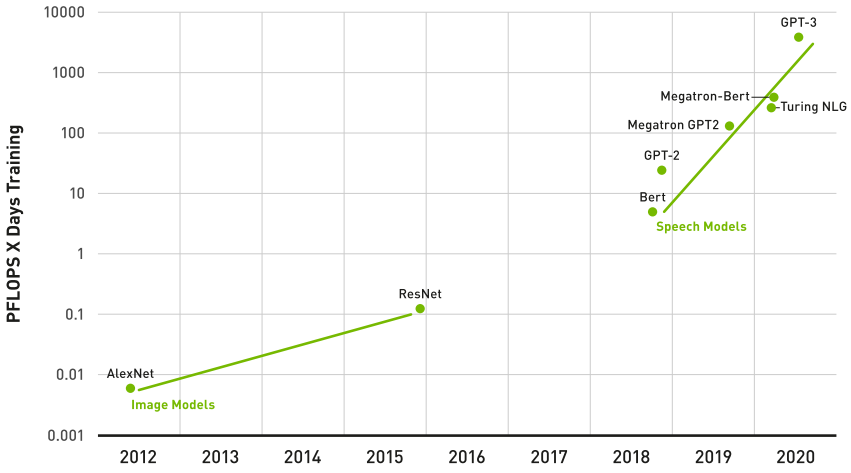

FIGURE 2. Scaling of deep learning: operations required to train vision and NLP models versus time.

a powerful neural network on a large dataset, in a reasonable amount of time.

This ingredient was provided by easily programmable GPUs with high floating-point performance. The field of computer vision, for example, was transformed by the development of AlexNet in 2012, a convolutional neural network trained on a pair of GeForce 580 (Fermi) GPUs in two weeks. AlexNet won the ImageNet competition in 2012 with such a large accuracy improvement that the computer vision community largely abandoned manually designed feature detectors in favor of neural networks. Similar results followed in other fields such as speech recognition, natural language processing, and recommender systems.

GPUs democratized deep learning, making the technology accessible to anyone with a GPU-equipped PC. In 2010, researchers at Google Brain built a deep neural network to find cats on the Internet using 16,000 CPUs. This experiment was repeated in 2012 on 48 GPUs[10] using software that ultimately became cuDNN, a CUDA library for deep learning. Just as CUDA simplified the task of scientific programming on GPUs, cuDNN simplified the task of implementing deep learning on GPUs, facilitating widespread use by deep learning researchers and practitioners.

As shown in Figure 2, the complexity of deep learning models has increased dramatically over time. The complexity of natural language models, for example, has increased by 1,000× in only two years from BERT in 2018 to GPT-3 in 2020. These models have significantly better fidelity, but the ability to train successively more complex models has depended on increasing hardware performance.

GPUs have evolved to meet this rapidly growing demand for deep learning training performance in two ways: scale-up and scale out. To facilitate scale-out, in 2015, the Pascal GPU introduced NVLink, a high-speed GPU–GPU communication channel. Then, in 2017, the Volta GPU introduced NVSwitch to enable high-performance networks of GPUs. Nodes of GPUs connected by NVSwitch share memory and communicate with other nodes via Infiniband networking. While AlexNet was originally trained on two GPUs, modern language models are trained on clusters with many thousands of GPUs. For example, Selene, which is #6 on the June 2021 Top 500 list, is a deep learning (DL) training cluster consisting of 560 DGX-A100 boxes connected by Infiniband. Each DGX box contains eight A100 GPUs connected by NVSwitch for a total of 4,480 GPUs.

Figure 3 shows how single GPU performance has scaled up to meet the demands of deep learning. GPU inference performance has improved by 317×, more than doubling each year, from 2012 (Kepler) to 2020 (Ampere), a trend far exceeding what Moore's law scaling would predict. Dubbed Huang's Law, this increase has been largely due to the addition of DL-specific data types and instructions.[11] The inference arithmetic performance available in Kepler was 4 TFLOPS of 32-bit floating-point (FP32) operations. This data representation exceeds the precision required for DL (particularly for inference). In addition, as the most complex instruction available in Kepler was a fused multiply–add (FMA), Kepler could only amortize the overhead of instruction fetch, decode, and operand fetch over two math operations per lane.[a] In 2016, the Pascal architecture increased FP32 performance to 10.6 TFLOPS and also added 21.3 TFLOPS

[a]The GPU's SIMT architecture amortizes instruction overhead over the 32-lanes of a *warp*.

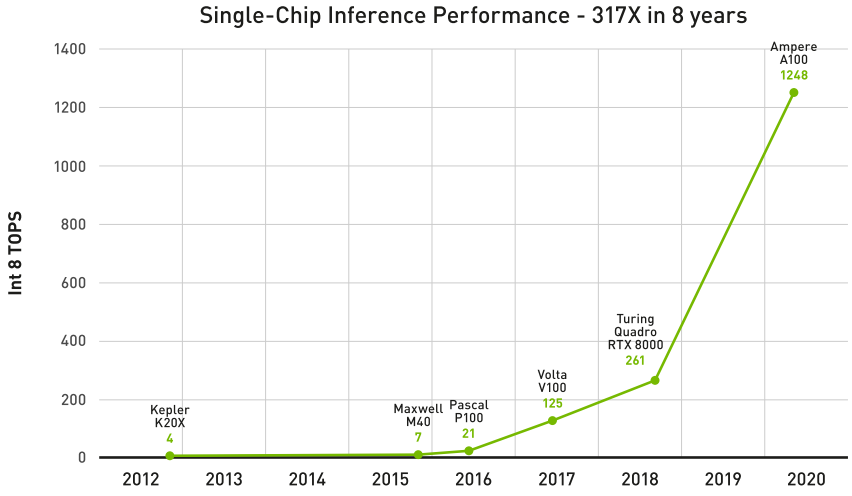

FIGURE 3. Huang's Law: GPU inference performance is more than doubling every year.

of FP16 performance. Pascal not only delivered higher DL performance but did so more efficiently because: 1) the accuracy and dynamic range of FP16 is adequate for most DL applications; and 2) Pascal's dot-product instructions, like FDP4, amortize the instruction overhead over additional arithmetic operations.

In 2017, Volta introduced a major advance in DL performance with the *Tensor Core* architecture. The Volta Tensor Cores implement a 4×4 FP16 matrix multiply and accumulate instruction, HMMA, which forms the inner loop of most DL algorithms. This single instruction implements 128 floating-point operations (64 FP16 multiplies and 64 FP32 adds), which amortizes instruction overheads and makes them negligible. As a result, the GPU Tensor Cores deliver the flexibility of a programmable architecture with the efficiency of a specialized DL chip. The FP16 tensor cores in Volta provide an aggregate 125 TFLOPS of FP16 performance. In 2018, Turing introduced integer tensor cores, which provide 260 TOPS of INT8 performance through an IMMA instruction that performs an 8×8 INT8 matrix multiply and accumulate. This support in Turing was ideal for most inference tasks for which INT8 has sufficient accuracy and dynamic range.

In 2020, Ampere introduced tensor core support for computation on sparse data. These sparse MMA instructions double performance for weight matrices where two out of every four elements are zero, called *structured sparsity*. The sparse instructions bring Ampere's total inference performance to 1,248 TOPS (INT8). Ampere is described in more detail in in the next section.

As with HPC, software is a key component of the success of GPUs in deep learning. Starting with the CuDNN library in 2012, a large ecosystem of deep learning software has been developed to aid in the creation and deployment of machine learning applications. NVIDIA's Tensor RT (TRT) optimizes inference code and its Triton Inference Server manages scheduling of multiple models. Third-party frameworks including TensorFlow, PyTorch, and Caffe2 are supported. The DALI library manages the parts of training pipelines auxiliary to deep learning, such as staging and decompressing the data. A template library called CUTLASS simplifies the process of developing highly optimized matrix multiply kernels for each new generation of GPUs. Libraries supporting applications such as medical imaging and video analytics are available. The evolution of software stacks is also critical to DL performance. In MLPerf Training v1.0 submissions, the performance of benchmarks improved by up to 2.1× on the same hardware, due to software enhancements alone.

Because of their high performance and rich software ecosystem, today's GPUs are the dominant platform for deep learning applications. The most recent MLPerf results demonstrate that NVIDIA GPU systems are the fastest commercially available platforms for both datacenter inference and training.[12–14]

BECAUSE OF THEIR HIGH PERFORMANCE AND RICH SOFTWARE ECOSYSTEM, TODAY'S GPUs ARE THE DOMINANT PLATFORM FOR DEEP LEARNING APPLICATIONS.

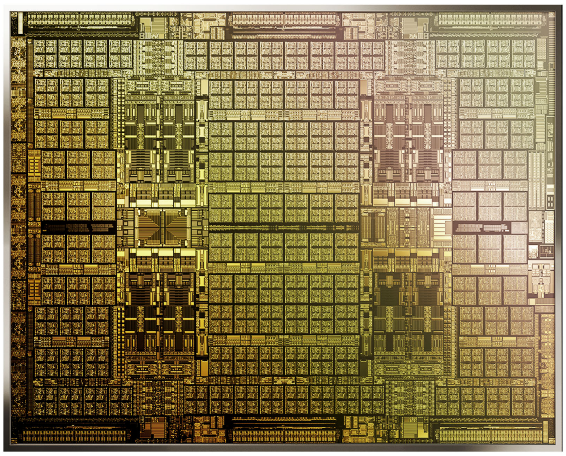

FIGURE 4. NVIDIA A100 GPU, launched November, 2020 (image artistically rendered).

TABLE 1. V100 and A100 specifications.

	Volta V100	Ampere A100	Increase
FP64	7.8 TFLOPS	19.5 TFLOPS	2.5×
FP16	125 TFLOPS	624[b] TFLOPS	5×
DRAM Bandwidth	900 GB/s	2,000 GB/s	2.2×
NVLink Bandwidth	300 GB/s	600 GB/s	2×
L2 Capacity	6 MB	40 MB	6.7×
DRAM Capacity	32 GB	80 GB	2.5×

A100: TODAY'S GPU

The Ampere A100, launched in 2020 and shown in Figure 4, is the most recent NVIDIA GPU.[15] Ampere is built using a 7-nm technology and includes 54 billion transistors on a reticle-limited 826-mm^2 die. Table 1 compares key attributes of Ampere A100 (2020) to its predecessor in the datacenter, the Volta V100 (2017) GPU. Ampere dramatically increases each of these key hardware features, including 5× greater FP16 throughput, 2.2× more DRAM bandwidth, and 6.7× more on-chip L2 cache. In addition to massive parallel computing throughput and memory bandwidth, the Ampere architecture includes hardware support for accelerating machine learning and HPC applications along with features for scalability to large datacenters and supercomputers.

Ampere also provides support for advances in the CUDA programming model. For example, Ampere accelerates CUDA task graphs by automatically launching dependent compute tasks within complex task graphs. Ampere also provides hardware support for arrive–wait barriers that facilitate coordination of asynchronous tasks in the GPU. Finally, Ampere provides additional hardware support for CUDA's Cooperative Groups, such as a warp-wide reduction operation that accelerates group-wide collective operations.

Machine Learning Features

Each Ampere Tensor Core employs its 256 FP16 floating-point units to perform an 8×4×8 mixed-precision matrix multiply every cycle. The Tensor Core architecture also supports high-throughput computation for different numerical representations, including binary (INT1), INT4, INT8, FP16, and BFloat16 (8-bit exponent, 7-bit mantissa or E8:M7). A100 also introduces a new TensorFloat-32 (TF32) format (E8:M10) that provides the same exponent dynamic range as FP32 but reduces the precision of the mantissa to 10 bits. A100 provides hardware for this 19-bit format that runs at 8× the throughput of FP32 but with sufficient precision to mimic the accuracy of DNNs trained with FP32.

Mentioned earlier, A100 also provides hardware support for structured sparsity in the Tensor Cores that can be used to dramatically reduce DNN weight storage and boost throughput by eliminating unnecessary arithmetic. Structured sparsity can be applied to compress the weight matrix (typically used as matrix A in the C = A×B GEMM formulation in the Tensor Cores) by requiring two of the four values in each subvector to be zero. This feature transforms the computation into a smaller matrix that requires half the operations of a dense formulation, producing a 2× speedup. While structured sparsity requires an additional neural network training refinement step, it produces no loss in accuracy on a wide range of networks including those for vision, object detection, segmentation, natural language modeling, and language translation.

In the memory system, A100 provides a range of features to provide better control over data movement and placement. To improve efficiency, A100 supports data transfers directly from the memory hierarchy into shared memory, without requiring the data to transit through the register file. A100 also provides a new set of L2 cache control operations that allow the programmer

[b]624 TFLOPS for sparse matrices with 2:1 structured sparsity; dense FP16 performance is 312 TFLOPS.

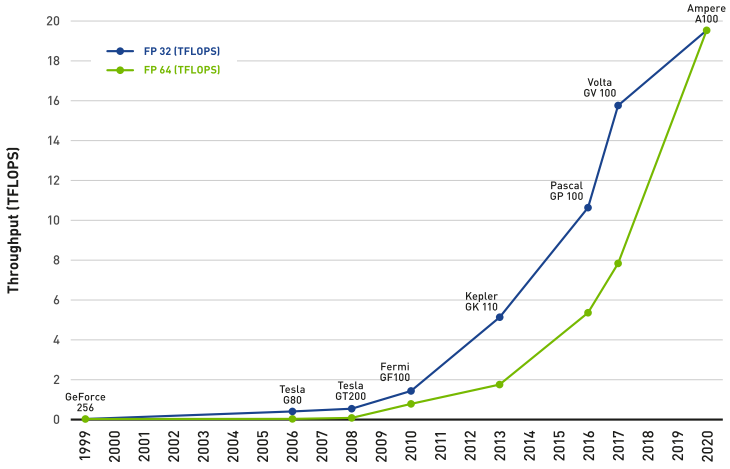

FIGURE 5. Single GPU performance scaling.

to influence the cache replacement policies and effectively dictate which data structures stay resident in the cache and which can be streamed from DRAM without polluting the cache. Finally, the L2 cache includes hardware-supported data compression in which data are kept compressed in both DRAM and L2 (saving bandwidth and capacity) and is then uncompressed/compressed when they are transferred to and from an SM.

HPC Features

Traditional HPC workloads continue to demand more double-precision compute performance, memory bandwidth, and interconnect bandwidth. Leveraging architecture concepts for GEMM acceleration, the A100 incorporates Tensor Core support for double-precision FP64 data types, boosting peak GPU performance to 19.5 TFLOPS. In addition, A100 deploys the third generation of NVLink interconnect, delivering 600 GB/s of total off-chip bandwidth through a combination of 50 Gb/s pin data rates and 12 NVLink channels. This high-speed interconnect enables tightly coupled systems such as the DGX A100 to include eight fully connected GPUs that can sustain 600 GB/s between all pairs of GPUs and deliver 150 TFLOPS of double-precision performance. When extended with Infiniband, the 560 DGX-A100 systems of Selene deliver 64 PFLOPS in just 2.6 MW; nine of the ten most energy-efficient supercomputers on the June 2021 Green 500 list are based on A100 GPUs.

Graphics Features

In addition to traditional raster graphics, the Ampere architecture includes improved hardware support for ray tracing. Introduced in the Turing GPU generation, the NVIDIA RT cores are integrated with the GPU's SMs to accelerate bounding-volume hierarchy data structure traversal and ray-triangle intersection tests.[16] Parallel shader code running on the SM casts millions of rays and passes them to the RT core, which then indicates whether each ray has hit a triangle. Ampere doubles the RT core throughput of Turing, enabling 2–3× faster frame rendering than software-only ray tracing on GPUs. Emphasizing the continuing convergence between graphics and compute, Turing and now Ampere enable graphics applications to employ machine-learning-based graphics algorithms such as deep learning supersampling (DLSS), using the Tensor Cores to further accelerate ray tracing pipelines.

THE ROAD AHEAD

GPUs have evolved into powerful parallel computing platforms with hardware support to accelerate computer graphics, HPC, and deep learning. In the coming years, GPUs will continue to evolve in performance, in application areas, in scalability, and in programmability.

As largely predicted in our article ten years ago, GPU FP64 performance increased by 20× over the decade from Fermi in 2010 to Ampere in 2020, an annual growth rate of 35% (see Figure 5).[17] We expect this trend to roughly continue, with some slowing, in future GPU generations. While the end of Dennard scaling in 2005 halted traditional voltage scaling, semiconductor feature sizes are still being made smaller. As GPUs move down the technology curve from the 7 nm of Ampere to 5 nm, 3 nm, and beyond, we will fit more devices on each die, and the capacitance, and

hence energy, of a given function will be reduced, allowing more performance per unit area and power. Deep learning inference performance has been increasing at a much faster rate (see Figure 3) of over 100% per year. We expect this trend to continue for the next several generations but then to converge to the 35% per year rate as architecture matures and gains become driven by technology.

AS GPUs MOVE DOWN THE TECHNOLOGY CURVE FROM THE 7 NM OF AMPERE TO 5 NM, 3 NM, AND BEYOND, WE WILL FIT MORE DEVICES ON EACH DIE, AND THE CAPACITANCE, AND HENCE ENERGY, OF A GIVEN FUNCTION WILL BE REDUCED, ALLOWING MORE PERFORMANCE PER UNIT AREA AND POWER.

A major challenge to the continued scaling of GPU performance is main memory bandwidth. While benchmarks like Linpack (HPL) make good use of caches and can saturate the arithmetic units of an A100, most HPC and ray-traced graphics applications are memory bandwidth limited. A typical HPC application needs 1–4 bytes from main memory per FP64 arithmetic operation. For example, the HPCG benchmark, which is representative of codes that use conjugate-gradient solvers, has a Byte/Flop ratio of 4:1. A major jump in memory bandwidth came in the Pascal generation with the move to high-bandwidth memory (HBM) mounted next to the GPU on a shared silicon interposer. However, Ampere's FP64 Tensor Cores and HBM2E DRAM still deliver a Byte/Flop ratio of only 0.1. We expect additional innovations in packaging and memory architecture will be needed to continue scaling memory bandwidth and HPC application performance.

GPUs are ideal platforms for integrating new domain-specific hardware acceleration. They provide a programmable platform with a high-bandwidth on-chip and off-chip memory system and off-chip networking. Domain-specific hardware can be added as new instructions in the SM (as with the tensor core instructions for deep learning) or as separate memory clients with operations launched from the SM (as with the RT cores). In coming years, we expect that new instructions and memory-based accelerators will be added to GPUs to support application areas such as databases, sparse linear algebra, and bioinformatics.

In addition to scaling up the performance of individual GPUs, we expect GPUs to be scaled out to larger clusters for both DL training and HPC applications. Today's NVLink-connected Ampere systems enable groups of up to 8 GPUs to share memory at per-GPU bandwidths of 600 GB/s, as if they were a single GPU. In coming generations, we expect the off-chip communication bandwidth to continue to scale and the size of both NVLink-connected nodes and Infiniband-connected clusters to grow. We also anticipate closer integration between GPUs, CPUs, and DPUs (smart NICs) to enable higher bandwidth communication and I/O with less overhead.

The use of CUDA and associated libraries has simplified the programming of GPUs to the point where over 600 HPC applications are GPU-accelerated. We expect that future GPUs will be even easier to program as better tools and abstractions for programming them become available. For example, the Legate programming system[18] allows Numpy programs to be run on any number of GPUs, from a single one to a 4,480 GPU cluster like Selene. Legate is built on top of Legion,[19] a runtime system that manages both the task-graph of a multi-GPU application and its data model. Legion schedules tasks and manages the communication, replication, and migration of data, greatly simplifying the task of programming a multi-GPU system.

One challenge in programming future GPUs is the inherent nonuniform cost of accessing different memory locations from different SMs. Historically, GPUs have provided a uniform memory access (UMA) model in which any memory location is equally accessible from any SM. With increases in GPU size and multi-GPU system scale, the UMA model becomes impossible to maintain, effectively requiring all memory to appear equally far away. We expect that a shift to a nonuniform memory access (NUMA) model will be required to enable SMs to be close to some portion of memory so that they can exploit that locality for increased performance and efficiency. This shift will require future GPU programs to express location and affinity so threads and the data they operate on can be co-located. While programmers already do this for multi-GPU systems, we expect the NUMA model to become useful within a single GPU as well.

GPUs have come a long way in the 20 years since the GeForce 256. They have increased in performance by 390×, have become highly programmable, and have added domain-specific hardware to support

HPC, deep learning, and ray-traced graphics. We expect comparable gains in performance and functionality over the next two decades.

ACKNOWLEDGMENTS

The shared effort of thousands of NVIDIA hardware and software engineers has resulted in the rapid evolution of GPUs over the last two decades. We all look forward to building the future together.

REFERENCES

1. B. Khailany *et al.*, "Imagine: Media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, Mar./Apr. 2001.
2. W. J. Dally *et al.*, "Merrimac: Supercomputing with streams," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2003, pp. 35–35.
3. I. Buck *et al.*, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, Aug. 2004.
4. J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for?," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008.
5. E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar./Apr. 2008.
6. J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010.
7. D. B. Kirk and W.-M. W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2009.
8. J. Choquette, O. Giroux, and D. Foley, "Volta: Performance and programmability," *IEEE Micro*, vol. 38, no. 2, pp. 42–52, Mar./Apr. 2018.
9. "GPUs now accelerate almost 600 HPC apps," Nov. 2018. [Online]. Available: https://www.hpcwire.com/off-the-wire/gpus-now-accelerate-almost-600-hpc-apps/
10. A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.
11. J.-H. Huang, NVIDIA GPU Technology Conference, Keynote Lecture, Mar. 28, 2018.
12. "ML Commons," 2021. [Online]. Available: https://mlcommons.org/en/
13. "Extending NVIDIA performance leadership with MLPerf inference 1.0 results," Apr. 2021. [Online]. Available: https://developer.nvidia.com/blog/extending-nvidia-performance-leadership-with-mlperf-inference-1-0-results/
14. "Global computer makers deliver breakthrough MLPerf results with NVIDIA AI," Jun. 2021. [Online]. Available: https://blogs.nvidia.com/blog/2021/06/30/mlperf-ai-training-partners/
15. J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar./Apr. 2021.
16. J. Burgess, "RTX on-the NVIDIA turing GPU," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, Mar./Apr. 2020.
17. S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Oct. 2011.
18. M. Bauer and M. Garland, "Legate NumPy: Accelerated and distributed array computing," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2019, pp. 1–23.
19. M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, pp. 1–11.

WILLIAM J. DALLY is the Chief Scientist and Senior Vice President of research at NVIDIA Corporation, Santa Clara, CA, USA, and an Adjunct Professor and former Chair of computer science at Stanford University, Stanford, CA, USA. His research interests include domain-specific accelerators, parallel computer architectures, interconnection networks, and high-speed signaling circuits. Dally received a Ph.D. degree in computer science in 1986 from the California Institute of Technology, Pasadena, CA, USA. He is a Member of the National Academy of Engineering, and a Fellow of IEEE, ACM, and the American Academy of Arts and Sciences. Contact him at bdally@nvidia.com.

STEPHEN W. KECKLER is the Vice President of architecture research at NVIDIA Corporation, Santa Clara, CA, USA, and an Adjunct Professor of computer science at the University of Texas at Austin, Austin, TX, USA. His research interests include parallel computer architectures, high-performance computing, energy-efficient architectures, and embedded computing. Keckler received a Ph.D. degree in computer science in 1998 from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is a Fellow of IEEE and ACM. Contact him at skeckler@nvidia.com.

DAVID B. KIRK is the former Chief Scientist, Vice President of architecture and research, and Founder of NVIDIA Research at NVIDIA Corporation, Santa Clara, CA, USA. He is currently an independent consultant. He has made significant contributions to graphics hardware and graphics algorithm research. His current research interests include computer science education, advancing parallel programming, robotics, and artificial intelligence. Kirk received a Ph.D. degree in computer science in 1993 from the California Institute of Technology, Pasadena, CA, USA. He is a Member of the National Academy of Engineering. Contact him at dkxxxx@hotmail.com.