

**EXHIBIT 7 – U.S. PATENT NO. 8,648,867 VS NVIDIA ACCELERATED COMPUTING PRODUCTS & SERVICES –
MAXWELL ARCHITECTURE**

I. INTRODUCTION

The chart below demonstrates how Defendant Nvidia Corporation (“Nvidia” or “Defendant”), as well as Defendant’s partners, customers, and end users, directly infringe, either literally or under the doctrine of equivalents, claims 16-19 of U.S. Patent No. 8,648,867 (“the ’867 Patent”).

Nvidia has committed acts of infringement within this District. Nvidia uses the Accused Products in this District in manners that practice the ’867 Patent, including by testing the Accused Products and by using the Accused Products at its offices and premises in this District. Defendant makes, uses, advertises, offers for sale, and/or sells hardware for accelerated computing, including GPUs, CPUs, and SoCs; computers for accelerated computing (e.g., supercomputers, servers, and data centers for high performance computing); and computer platform software-as-a-service (“SaaS”) that implements accelerated computing (including the Accused Products) in the State of Texas and in this District directly and/or through its partnerships

Defendant also provides data center and HPC services that practice the ’867 Patent in the State of Texas and in this District directly and/or through its partnerships with businesses in the State of Texas and in this District.

Nvidia sells, offers for sale, advertises, makes, installs, and/or otherwise provides hardware, software, firmware, and/or computer platforms for accelerated computing and data center and HPC services, including the Accused Products, the use of which infringes the ’867 Patent in this District and the State of Texas. (See <https://www.nvidia.com/en-us/data-center/solutions/accelerated-computing/>.) Nvidia performs these acts directly and/or through its partnerships with other entities. (See *id.* (“NVIDIA has defined a range of accelerated platforms that each consist of hardware systems designed according to the needs of the use case as well as the software stack that enables the operation and management of the business applications. These hardware systems and software are available from NVIDIA and our partners.”).)

Nvidia also uses a network of partners, which comprise re-sellers, managed service providers, and product and solution experts, to provide the Accused Products and implementation services for the Accused Products to customers in this District. Each of these partners sells, offers for sale, installs, and/or implements Nvidia’s accelerated computing hardware, software, and/or computer platform services. (See <https://www.nvidia.com/en-us/about-nvidia/partners/>.)

Nvidia’s partners include “Data Center Provider[s].” (See <https://www.nvidia.com/en-us/about-nvidia/partners/>.) Nvidia’s Data Center Provider partners “offer colocation services such as high-density data center facilities, interconnected infrastructure, and state-of-art

cooling technologies for hosting NVIDIA DGX™ servers globally.” (See *id.*) Nvidia’s Data Center Provider partners in the “NVIDIA DGX-Ready Data Center program, built on the NVIDIA DGX™ platform and delivered by NVIDIA partners,” help “accelerate the scaling (See <https://www.nvidia.com/en-us/data-center/colocation-partners/#aligned-energy>.)

As further detailed below, Nvidia engages in activities that directly infringe the ’867 Patent within this District. For example, Nvidia’s operation and use of its accelerated computing hardware, software, and/or computer platform services, including its data center-scale accelerated computing platforms, within this District infringe the ’867 Patent.

Nvidia also infringes (directly or indirectly) the ’867 Patent by providing services in connection with the Accused Products including installing, maintaining, supporting, operating, providing instructions, and/or advertising Nvidia’s computer platform, data center, and HPC services within this District. For example, under Nvidia’s cloud and data center line of products and services, the Nvidia DGX platform is a “a fully integrated hardware and software AI platform” and “combines the best of NVIDIA software, infrastructure, and expertise in a modern, unified AI development solution.” (See <https://www.nvidia.com/en-us/data-center/dgx-platform/>.) Indeed, “DGX infrastructure is a complete AI solution, and includes NVIDIA AI Enterprise software to accelerate data science pipelines and streamline development and deployment of production-grade AI applications.” (See *id.*) Nvidia platform user and partner customers infringe the ’867 Patent by installing and operating Nvidia’s computer platform software, which performs the claimed methods in the ’867 Patent within this District. (See also, *e.g.*, <https://www.nvidia.com/en-us/data-center/products/ai-enterprise/> (Nvidia AI Enterprise); <https://developer.nvidia.com/cuda-zone> (Nvidia CUDA Toolkit); <https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/> (GPU Cloud Computing).)

Defendant encourages and induces its customers of the Accused Products to perform the methods claimed in the ’867 Patent. For example, Nvidia makes its accelerated platforms and services available on its website, widely advertises those platforms and services, provides applications that allow partners and users to access those platforms and services, provides instructions for installing, and maintaining those platforms and services and supporting software and/or firmware, and provides technical support to users. (See <https://www.nvidia.com/en-us/data-center/dgx-support/>.) Nvidia further encourages and induces its customers to operate Nvidia’s hardware and software in an infringing manner, and to use Nvidia’s infringing computer platforms, by providing directions for and encouraging customers to install software, such as software for NVIDIA AI Enterprise and CUDA, (see <https://docs.nvidia.com/ai-enterprise/deployment-guide-vmware/0.1.0/software.html>; <https://developer.nvidia.com/cuda-downloads>), which offers evaluation, installation, configuration, customization, and development of Nvidia’s infringing software products and services. Defendant also contributes to the infringement of its customers and end users of the Accused Products by offering within the United States or importing into the United States the Accused Products, which are for use in practicing, and under normal operation practice, one or more of the methods claimed in the ’867 Patent, constituting a material part of the inventions claimed, and not a staple article or commodity of

commerce suitable for substantial non-infringing uses. Indeed, as shown herein, the Accused Products and the example functionality described below have no substantial non-infringing uses and are specifically designed to practice the methods claimed in the '867 Patent.

Nvidia offers, sells, and uses several products that provide and implement GPU-acceleration hardware, software, platforms, and services for individuals and enterprises and incorporate Plaintiff's patented technologies. (See <https://www.nvidia.com/en-us/solutions/ai/inference/>; <https://marketplace.nvidia.com/en-us/data-center/?page=4>; <https://marketplace.nvidia.com/en-us/laptops-workstations/?page=9>; <https://marketplace.nvidia.com/en-us/software/?page=3>.)

The Accused Products, as described *infra*, include Nvidia's hardware, software, and services designed, implemented, and used for hardware acceleration, including Nvidia's GPU accelerators and superchips; Nvidia's computers, supercomputers, data centers, servers, workstations that implement its GPU accelerators and superchips; and Nvidia's software, platforms, and services for accelerated computing.

The Accused Products include Nvidia's GPU accelerators and superchips. (See <https://resources.nvidia.com/en-us-gpu>.) Nvidia's GPU accelerators include Nvidia's GPUs with Nvidia's "Hopper," "Ada Lovelace," "Ampere," "Turing," "Volta," "Pascal," and "Maxwell" GPU architectures. (See <https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-896/support-matrix/index.html>.) These GPUs are specifically designed to run and implement GPU-based hardware acceleration using Nvidia's proprietary CUDA (Compute Unified Device Architecture) platform and CUDA libraries for GPU acceleration. (See *id.* (Nvidia GPU architectures implementing Nvidia's cuDNN (CUDA Deep Neural Network) library for GPU acceleration.); <https://developer.nvidia.com/cuda-gpus>.)

Nvidia's Hopper GPUs include the H100 and H200 GPUs. (See <https://www.nvidia.com/en-us/data-center/technologies/hopper-architecture/> (Hopper architecture); <https://www.nvidia.com/en-us/data-center/h100/> (H100); <https://www.nvidia.com/en-us/data-center/h200/> (H200).) In addition, Nvidia's superchips that implement GPU accelerators include the GH200, or Grace Hopper Superchip, which implements the Hopper-GPU architecture. (See <https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/> (GH200).)

Nvidia's Ada Lovelace (or Lovelace) GPUs include Nvidia Data Center GPUs, including L40, L40S, and L4 GPUs; Nvidia Workstation and Professional Laptop GPUs, including RTX Ada Generations series GPUs and Laptop GPUs (including RTX 6000, RTX 6000 Ada, RTX 5000 Ada, RTX 4500 Ada, RTX 4000 Ada, RTX 4000 SFF, RTX 3500, RTX 3000, RTX 2000, RTX 1000, RTX 500); and GeForce RTX 40 series GPUs and Laptop GPUs (RTX 4090, RTX 4080 SUPER, RTX 4070 Ti SUPER, RTX 4070 SUPER, RTX 4070, RTX 4060 Ti, and RTX 4060; GeForce RTX 4090 Laptop GPU, GeForce RTX 4080 Laptop GPU, GeForce RTX 4070 Laptop GPU, GeForce RTX 4060 Laptop GPU, GeForce RTX 4050 Laptop GPU). (See <https://www.nvidia.com/en-us/technologies/ada-architecture/> (Ada Lovelace architecture). See <https://www.nvidia.com/en-us/data-center/l40/> (L40); <https://www.nvidia.com/en-us/data-center/l40s/>

(L40S); <https://www.nvidia.com/en-us/data-center/l4/> (L4). See <https://resources.nvidia.com/en-us-design-viz-stories-ep/l40-linecard> (Nvidia Professional GPUs); <https://www.nvidia.com/en-us/ai-on-rtx/> (RTX GPUs featuring “Accelerated Development”); <https://www.nvidia.com/en-us/design-visualization/desktop-graphics/> (RTX Ada Generation GPUs); <https://www.nvidia.com/en-us/design-visualization/rtx-professional-laptops/compare-table/> (RTX Ada Generation Laptop GPUs). See <https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/> (GeForce RTX 40 GPUs); <https://www.nvidia.com/en-us/geforce/graphics-cards/compare/> (GeForce RTX 40 GPUs); <https://www.nvidia.com/en-us/geforce/laptops/compare/> (GeForce RTX 40 Laptop GPUs).)

Nvidia’s Ampere GPUs include Nvidia Data Center GPUs, including A100, A40, A30, A16, A10, and A2 GPUs; Nvidia Workstation and Professional Laptop GPUs, including RTX A series GPUs and Laptop GPUs (A800 40GB Active, RTX A6000, RTX A5500, RTX A5000, RTX A4500, RTX A4000, RTX A2000, RTX A2000 12GB, RTX A1000, RTX A400, RTX A5500, RTX A4500, RTX A3000 12GB, RTX A2000 8GB, RTX A1000 6GB, RTX A500); GeForce RTX 30 series GPUs and Laptop GPUs (GeForce RTX 3090 Ti, GeForce RTX 3090, GeForce RTX 3080 Ti, GeForce RTX 3080, GeForce RTX 3070 Ti, GeForce RTX 3070, GeForce RTX 3060 Ti, GeForce RTX 3060, GeForce RTX 3050 (8 GB), GeForce RTX 3050 (6 GB), GeForce RTX 3080 Ti Laptop GPU, GeForce RTX 3080 Laptop GPU, GeForce RTX 3070 Ti Laptop GPU, GeForce RTX 3070 Laptop GPU, GeForce RTX 3060 Laptop GPU, GeForce RTX 3050 Ti Laptop GPU, GeForce RTX 3050 Laptop GPU); and GeForce MX570 Laptop GPU. (See <https://www.nvidia.com/en-us/data-center/ampere-architecture/> (Ampere architecture). See <https://www.nvidia.com/en-us/data-center/a100/> (A100); <https://www.nvidia.com/en-us/data-center/a40/> (A40); <https://www.nvidia.com/en-us/data-center/a30/> (A30); <https://www.nvidia.com/en-us/data-center/a16/> (A16); <https://www.nvidia.com/en-us/data-center/a10/> (A10); <https://www.nvidia.com/en-us/data-center/a2/> (A2). See <https://www.nvidia.com/en-us/design-visualization/desktop-graphics/> (RTX A GPUs); <https://www.nvidia.com/en-us/design-visualization/rtx-professional-laptops/compare-table/> (RTX A Laptop GPUs). See <https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/>; (GeForce RTX 30 GPUs) <https://www.nvidia.com/en-us/geforce/graphics-cards/compare/> (GeForce RTX 30 GPUs); <https://www.nvidia.com/en-us/geforce/laptops/compare/30-series/> (GeForce RTX 30 Laptop GPUs); <https://www.nvidia.com/en-us/geforce/gaming-laptops/mx-570/> (GeForce MX570 Laptop GPU).)

Nvidia’s Turing GPUs include Nvidia Data Center GPUs, including Tesla T4 GPUs and Quadro RTX 8000 (passive) and Quadro RTX 6000 (passive) GPUs; Nvidia Workstation and Professional Laptop GPUs, including T series GPUs and Laptop GPUs, Quadro T series Laptop GPUs, and Quadro RTX series GPUs and Laptop GPUs (Quadro RTX 8000, Quadro RTX 6000, Quadro RTX 5000, Quadro RTX 4000, Quadro RTX 3000, Quadro T2000, T1000 8GB, T1200, Quadro T1000, T1000 (4GB), T600, T550, T500 T400, T400 4GB); Titan series Titan RTX GPU; GeForce RTX 20 series GPUs and Laptop GPUs (GeForce RTX 2080 Ti, GeForce RTX 2080 Super, GeForce RTX 2080, GeForce RTX 2070 Super, GeForce RTX 2070, GeForce RTX 2060 Super, GeForce RTX 2060, GeForce RTX 2500); GeForce GTX 16 series GPUs and Laptop GPUs (GeForce GTX 1660 Ti, GeForce GTX 1660 Super, GeForce GTX 1660, GeForce GTX 1650 Ti, GeForce GTX 1650 Super, GeForce GTX 1650 (G5), GeForce GTX 1650 (G6), GeForce GTX 1650, GeForce GTX 1630); and GeForce MX550, MX450, and MX430 Laptop GPUs. (See <https://www.nvidia.com/en-us/geforce/turing/> (Turing

architecture). See <https://www.nvidia.com/en-us/data-center/tesla-t4/> (Tesla T4); <https://www.nvidia.com/en-gb/design-visualization/quadro-data-center/> (Quadro RTX 8000 (passive) and Quadro RTX 6000 (passive)). See <https://www.nvidia.com/en-us/design-visualization/quadro/> (T series GPUs/Laptop GPUs, Quadro T series Laptop GPUs, and Quadro RTX GPUs/Laptop GPUs); <https://www.nvidia.com/en-us/design-visualization/desktop-graphics> (T series GPUs/Laptop GPUs); <https://www.nvidia.com/content/dam/en-zz/Solutions/titan/documents/titan-rtx-for-creators-us-nvidia-1011126-r6-web.pdf> (Titan RTX); <https://www.nvidia.com/en-us/geforce/20-series/> (GeForce RTX 20 GPUs); <https://www.nvidia.com/en-us/geforce/graphics-cards/compare/> (GeForce RTX 20 GPUs and GeForce GTX 16 GPUs); <https://www.nvidia.com/en-us/geforce/gaming-laptops/compare-20-series/> (GeForce RTX 20 Laptop GPUs); <https://www.nvidia.com/en-us/geforce/gaming-laptops/compare-16-series/> (GeForce GTX 16 Laptop GPUs); <https://www.nvidia.com/en-us/geforce/gaming-laptops/mx-550/> (GeForce MX550 Laptop GPU); <https://www.nvidia.com/en-us/geforce/gaming-laptops/mx-450/> (GeForce MX450 Laptop GPU); <https://wccftech.com/nvidia-geforce-mx450-turing-discrete-notebook-gpu-gddr6-pcie-4/> (GeForce M Laptop GPUs).)

Nvidia's Volta GPUs include Nvidia Data Center GPUs, including the Tesla V100 GPU; Nvidia Workstation GPUs, including Quadro GV100; and Titan series Titan V GPU. (See <https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/> (Volta architecture); <https://www.nvidia.com/en-us/data-center/v100/> (Tesla V100); <https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/quadro-volta-gv100-data-sheet-us-nvidia-704619-r3-web.pdf> (Quadro GV100); <https://nvidianews.nvidia.com/news/nvidia-titan-v-transforms-the-pc-into-ai-supercomputer> (Titan V).)

Nvidia's Pascal GPUs include Nvidia Data Center GPUs, including Tesla P100, P40, and P4 GPUs; Nvidia Workstation and Professional Laptop GPUs, including the Quadro GP100 GPU and Quadro P series GPUs and Laptop GPUs (Quadro P6000, Quadro P5200, Quadro P5000, Quadro P4200, Quadro P4000, Quadro P3200, Quadro P3000, Quadro P2200, Quadro P2000, Quadro P1000, Quadro P620, Quadro P600, Quadro P520, Quadro P500, Quadro P400); Titan series Titan Xp and Titan X GPUs; GeForce GTX 10 series GPUs and Laptop GPUs (GeForce GTX 1080 Ti, GeForce GTX 1080, GeForce GTX 1070 Ti, GeForce GTX 1070, GeForce GTX 1060, GeForce GTX 1050 Ti, GeForce GTX 1050); and GeForce MX300 series, MX200 series, and MX150 Laptop GPUs. (See <https://developer.nvidia.com/pascal>; <https://www.nvidia.com/en-us/data-center/pascal-gpu-architecture/> (Pascal architecture). See <https://www.nvidia.com/en-us/data-center/tesla-p100> (Tesla P100); <https://developer.nvidia.com/cuda-gpus> (Tesla P40 and P4); <https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/quadro-pascal-gp100-data-sheet-us-nv-704562-r1.pdf> (Quadro GP100); <https://www.nvidia.com/en-us/design-visualization/quadro/> (Quadro P series GPUs/Laptop GPUs). See https://www.nvidia.com/content/geforce-gtx/NVIDIA_TITAN_X_USER_GUIDE_v02.pdf (Titan X); https://www.nvidia.com/content/geforce-gtx/NVIDIA_TITAN_Xp_USER_GUIDE_v02.pdf (Titan Xp); <https://www.nvidia.com/en-us/geforce/10-series/> (GeForce GTX 10); <https://www.nvidia.com/en-us/geforce/graphics-cards/compare/> (GeForce GTX 10 GPUs); <https://www.nvidia.com/en-us/geforce/news/gfecnt/nvidia-geforce-gtx-10-series-laptops/> (GeForce GTX 10 Laptop GPUs); <https://www.nvidia.com/en-us/geforce/gaming-laptops/mx-350/> (GeForce MX350 Laptop GPU); [5](https://www.nvidia.com/en-</p>
</div>
<div data-bbox=)

us/geforce/gaming-laptops/mx-330/ (GeForce MX330 Laptop GPU); <https://wccfttech.com/nvidia-geforce-mx450-turing-discrete-notebook-gpu-gddr6-pcie-4/> (GeForce M Laptop GPUs).)

Nvidia's Maxwell GPUs include Nvidia Data Center GPUs, including Tesla M60, M40, and M10 GPUs; Nvidia Workstation and Professional Laptop GPUs, including Quadro M series GPUs and Laptop GPUs (Quadro M6000 24GB, Quadro M6000 (12GB), Quadro M5000, Quadro M5000M, Quadro M5500, Quadro M4000, Quadro M4000M, Quadro M3000M, Quadro M2200, Quadro M2000, Quadro M2000M, Quadro M1200, Quadro M1000M, Quadro M620, Quadro M600M, Quadro M520, Quadro M500M), the NVS 810 GPU, and Tesla M6 series Laptop GPUs; Titan series GTX Titan X GPU; GeForce GTX 900 series GPUs and Laptop GPUs (GeForce GTX 980Ti, GeForce GTX 980, GeForce GTX 970, GeForce GTX 960, GeForce GTX 980M, GeForce GTX 970M, GeForce GTX 965M, GeForce GTX 960M, GeForce GTX 950M); GeForce GTX 700 series GPUs and Laptop GPUs (GeForce GTX 750 Ti, GeForce GTX 750); and GeForce MX130 series and MX110 Laptop GPUs. (See <https://developer.nvidia.com/blog/maxwell-most-advanced-cuda-gpu-ever-made/> (Maxwell architecture); <https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/solutions/resources/documents1/nvidia-m60-datasheet.pdf> (M60); https://images.nvidia.com/content/tesla/pdf/78071_Tesla_M40_24GB_Print_Datasheet_LR.PDF (M40); <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-m10/pdf/188359-Tesla-M10-DS-NV-Aug19-A4-fnl-Web.pdf> (M10); <https://www.nvidia.com/en-us/design-visualization/quadro/> (Quadro M GPUs/Laptop GPUs); <https://www.nvidia.com/docs/IO/146527/nvs-810-datasheet.pdf> (NVS 810); <https://images.nvidia.com/content/tesla/pdf/188300-Tesla-M6-DS-Aug19-A4-fnl-Web.pdf> (Tesla M6); https://www.nvidia.com/content/geforce-gtx/GTX_TITAN_X_User_Guide.pdf (GTX Titan X); <https://developer.nvidia.com/maxwell-compute-architecture> (GeForce GTX 900 and 700 GPUs/Laptop GPUs); <https://wccfttech.com/nvidia-geforce-mx450-turing-discrete-notebook-gpu-gddr6-pcie-4/> (GeForce M Laptop GPUs).

These GPUs and superchips implement, and are specifically designed for, GPU-acceleration for artificial intelligence and neural networks. Nvidia's proprietary CUDA platform for parallel computing, which includes GPU-acceleration libraries such as cuDNN (CUDA Deep Neural Network), is implemented in the Nvidia Hopper, Ada Lovelace, Ampere, Turing, Volta, Pascal, and Maxwell GPU architectures.

The Accused Products further include Nvidia's computers, supercomputers, data centers, servers, workstations that implement its GPU accelerators and superchips. These computer hardware systems include: the DGX line of supercomputers, the HGX line of supercomputers, the OVX line of supercomputers, and the EGX line of servers for data centers and edge devices. (See <https://www.nvidia.com/en-us/data-center/solutions/accelerated-computing/>.)

Nvidia's DGX supercomputers include the DGX H200, DGX BasePOD, and DGX SuperPOD with DGX GB200. (See <https://www.nvidia.com/en-us/data-center/dgx-platform/>; see also <https://www.nvidia.com/en-us/data-center/base-command/>;

<https://resources.nvidia.com/en-us-dgx-software/nvidia-base-command> (DGX Base Command operating system for DGX data centers.) Nvidia's HGX "AI supercomputing platform brings together the full power of NVIDIA GPUs, NVIDIA NVLink™, NVIDIA networking, and fully optimized AI and high-performance computing (HPC) software stacks." (See <https://www.nvidia.com/en-us/data-center/hgx/>; <https://nvdam.widen.net/s/5kgbjq2v2t/hpc-hgx-h100-datasheet-nvidia-web>.) One example configuration includes "four or eight H200 or H100 GPUs." (See *id.*; see <https://nvdam.widen.net/s/5kgbjq2v2t/hpc-hgx-h100-datasheet-nvidia-web>.) Nvidia's OVX supercomputers implement "L40S GPUs . . . for both complex AI and graphics-intensive workloads." (See <https://www.nvidia.com/en-us/data-center/products/ovx/>; see <https://resources.nvidia.com/en-us-ovx/ovx-datasheet>.) And Nvidia's "EGX hardware portfolio" includes "accelerators [that] combine the performance of NVIDIA Ampere GPUs." (See <https://www.nvidia.com/en-us/data-center/products/egx/>; see <https://www.nvidia.com/en-us/design-visualization/egx-graphics/>.)

The Accused Products further include Nvidia's software, platforms, and services for accelerated computing. These includes CUDA, Nvidia AI Enterprise, the DGX Platform, Nvidia Omniverse, Nvidia Drive, Nvidia Isaac Sim, Nvidia Clara, Nvidia AI Foundation models, and Nvidia NGC.

CUDA is Nvidia's proprietary "parallel computing platform and programming model." (See <https://developer.nvidia.com/cuda-zone>.) CUDA is designed to support Nvidia's GPU accelerators and superchips and includes software specifically for GPU-acceleration such as the cuDNN "GPU-accelerated library." (See *id.*; <https://developer.nvidia.com/cudnn>.) In addition, Nvidia's CUDA-X, built on top of CUDA, is a collection of "GPU-accelerated microservices and libraries for AI." (See <https://www.nvidia.com/en-us/technologies/cuda-x/>.) Nvidia also offers the CUDA Toolkit and SDK Manager for developing GPU-accelerated applications. (See <https://developer.nvidia.com/cuda-toolkit>; <https://developer.nvidia.com/sdk-manager>.)

In addition, Nvidia AI Enterprise is Nvidia's "end-to-end, cloud-native software platform" for "accelerat[ing] data science pipelines . . . and other generative AI applications." (See <https://www.nvidia.com/en-us/data-center/products/ai-enterprise/>.) It is Nvidia's "'operating system' for enterprise AI." (See *id.*)

In addition, Nvidia's DGX platform is "is a complete AI solution and includes NVIDIA AI Enterprise software." (See <https://www.nvidia.com/en-us/data-center/dgx-platform/>.) Nvidia DGX Cloud is "an AI-training-as-a-service platform which includes cloud-based infrastructure and software for AI, customizable pretrained AI models, and access to NVIDIA experts." (See <https://d18rn0p25nwr6d.cloudfront.net/CIK-0001045810/1cbe8fe7-e08a-46e3-8dcc-b429fc06c1a4.pdf>, Nvidia U.S. Securities and Exchange Commission Form 10-K for Fiscal Year Ended January 28, 2024 at 6.)

In addition, Nvidia Omniverse is "a development platform and operating system for building virtual world simulation applications, available as a software subscription." (See <https://d18rn0p25nwr6d.cloudfront.net/CIK-0001045810/1cbe8fe7-e08a-46e3-8dcc>

b429fc06c1a4.pdf, Nvidia U.S. Securities and Exchange Commission Form 10-K for Fiscal Year Ended January 28, 2024 at 6.) Nvidia Omniverse implements software and services “into existing software tools and simulation workflows for building AI systems.” (See <https://www.nvidia.com/en-us/omniverse/>.)

In addition, Nvidia Drive is a platform that “consists of both the AI infrastructure and in-vehicle hardware and software” for autonomous vehicles. (See <https://www.nvidia.com/en-us/self-driving-cars/>.) “NVIDIA DRIVE Infrastructure encompasses data center hardware, software, and workflows—both on premises and in NVIDIA DGX Cloud & Omniverse.” (See *id.*)

In addition, Nvidia Isaac Sim is a platform that enables “developers to design, simulate, test, and train AI-based robots and autonomous machines in a physically-based virtual environment.” (See <https://developer.nvidia.com/isaac/sim/>.) It is built on Nvidia Omniverse. (See *id.*)

In addition, Nvidia Clara is “a suite of computing platforms, software, and services that powers AI solutions for healthcare and life sciences, from imaging and instruments to genomics and drug discovery” that provides “AI-Powered Solutions for Healthcare.” (See <https://www.nvidia.com/en-us/clara/>.)

In addition, Nvidia AI Foundation models are “community and NVIDIA-built models” that “are NVIDIA-optimized to deliver the best performance on NVIDIA accelerated infrastructure.” (See <https://www.nvidia.com/en-us/ai-data-science/foundation-models/>.)

In addition, Nvidia NGC is a collection of software services and tools that support “end-to-end AI and digital twin workflows” that runs on “NVIDIA GPU-accelerated platforms.” (See <https://www.nvidia.com/en-us/gpu-cloud/>.) NGC “offers a collection of cloud services . . . for generative AI, drug discovery, and speech AI solutions, and the NGC Private Registry for securely sharing proprietary AI software.” (See *id.*)

II. CLAIM CHARTS

Key Features	Accused Products
[16.Pre] A method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator, the method comprising:	<p>The Accused Products perform each step of the method of claim 16 of the '867 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform <i>a method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator</i>, as further explained below.</p> <p>The Accused Products perform <i>a method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator</i> by implementing Nvidia's CUDA (Compute Unified Device Architecture) parallel computing platform, including CUDA toolkits and drivers, and deploying that platform on a computer system to perform <i>numerical simulations</i>.</p> <p>The Accused Products include Nvidia GPUs with the Maxwell GPU architecture. All of these GPUs are used to perform <i>a method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator</i>. The aforementioned Nvidia GPU architectures are compatible with Nvidia's proprietary CUDA platform for parallel computing, which is "a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs)." CUDA further includes the CUDA Toolkit, which "includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime." An exemplary CUDA library specialized for acceleration is cuDNN (CUDA Deep Neural Network), "a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, attention, matmul, pooling, and normalization."</p>

Key Features	Accused Products
	<p data-bbox="1050 248 1409 305" style="text-align: center;">CUDA Zone</p> <p data-bbox="590 347 1866 407"><u>CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.</u></p> <p data-bbox="590 443 1866 570">In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance – while the compute intensive portion of the application runs on thousands of GPU cores in parallel. When using CUDA, developers program in popular languages such as C, C++, Fortran, Python and MATLAB and express parallelism through extensions in the form of a few basic keywords.</p> <p data-bbox="581 609 1875 669">The <u>CUDA Toolkit</u> from NVIDIA provides everything you need to develop GPU-accelerated applications. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime.</p> <p data-bbox="569 688 1386 721">(See https://developer.nvidia.com/cuda-zone (emphasis added).)</p> <p data-bbox="987 776 1465 833" style="text-align: center;">NVIDIA cuDNN</p> <p data-bbox="581 881 1875 1078"><u>The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks.</u> cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, attention, matmul, pooling, and normalization.</p> <p data-bbox="569 1088 1335 1120">(See https://developer.nvidia.com/cudnn (emphasis added).)</p> <p data-bbox="569 1161 1894 1411">As illustrated in the table below, Nvidia’s GPUs implementing Maxwell architecture (“Supported NVIDIA Hardware”) support Nvidia’s CUDA platform for parallel computing and CUDA libraries such as cuDNN for GPU accelerated. As shown in the table below, columns for “cuDNN Package” and “CUDA Compute Capability” (represented by a version number or SM version, which “identifies the features supported by the GPU hardware and is used by applications at runtime to determine which hardware features and/or instructions are available on the present GPU”) identify the supported functionality the Accused Product GPUs support via their Nvidia GPU architecture.</p>

Key Features	Accused Products
	<p>For cuDNN, as an example, package 9.5.1 for CUDA 12.x (e.g., CUDA 12.0 and later) supports Nvidia GPUs with CUDA Compute Capability numbers 9.0, 8.9, 8.6, 8.0, 7.5, 7.0, 6.1, 6.0, and 5.0 including the Accused Products with the Maxwell GPU architecture. Nvidia notes that “the recommended configuration for GPUs Volta or later is cuDNN 9.5.1 with CUDA 12.6. For GPUs prior to Volta (that is, Pascal and Maxwell), the recommended configuration is cuDNN 9.5.1 with CUDA 11.8.” As shown below, Maxwell GPUs have a CUDA compute Capability of 5.2 or 5.0, which supports CUDA 11 for cuDNN 9.5.1. In addition, because “CUDA is backward compatible, existing CUDA applications can continue to be used with newer CUDA versions.”</p> <p>Support Matrix</p> <p>GPU, CUDA Toolkit, and CUDA Driver Requirements</p> <p>The following sections highlight the compatibility of NVIDIA cuDNN versions with the various supported NVIDIA CUDA Toolkit, CUDA driver, and NVIDIA hardware versions.</p>

Key Features	Accused Products					
	Supported NVIDIA Hardware and CUDA Version					
	cuDNN Package [1]	CUDA Toolkit Version	Supports static linking? [2]	NVIDIA Driver Version for Linux	NVIDIA Driver Version for Windows	CUDA Compute Capability
	cuDNN 9.5.1 for CUDA 12.x	› 12.6	Yes	›=525.60.13	›=527.41	› 9.0 [3]
		› 12.5				› 8.9 [3]
		› 12.4				› 8.6
		› 12.3				› 8.0
		› 12.2				› 7.5
		› 12.1				› 7.0
		› 12.0				› 6.1
						› 6.0
						› 5.0
						› NVIDIA Hopper [3]
						› NVIDIA Ada Lovelace architecture [3]
						› NVIDIA Ampere architecture
						› NVIDIA Turing
						› NVIDIA Volta
						› NVIDIA Pascal
						› NVIDIA Maxwell

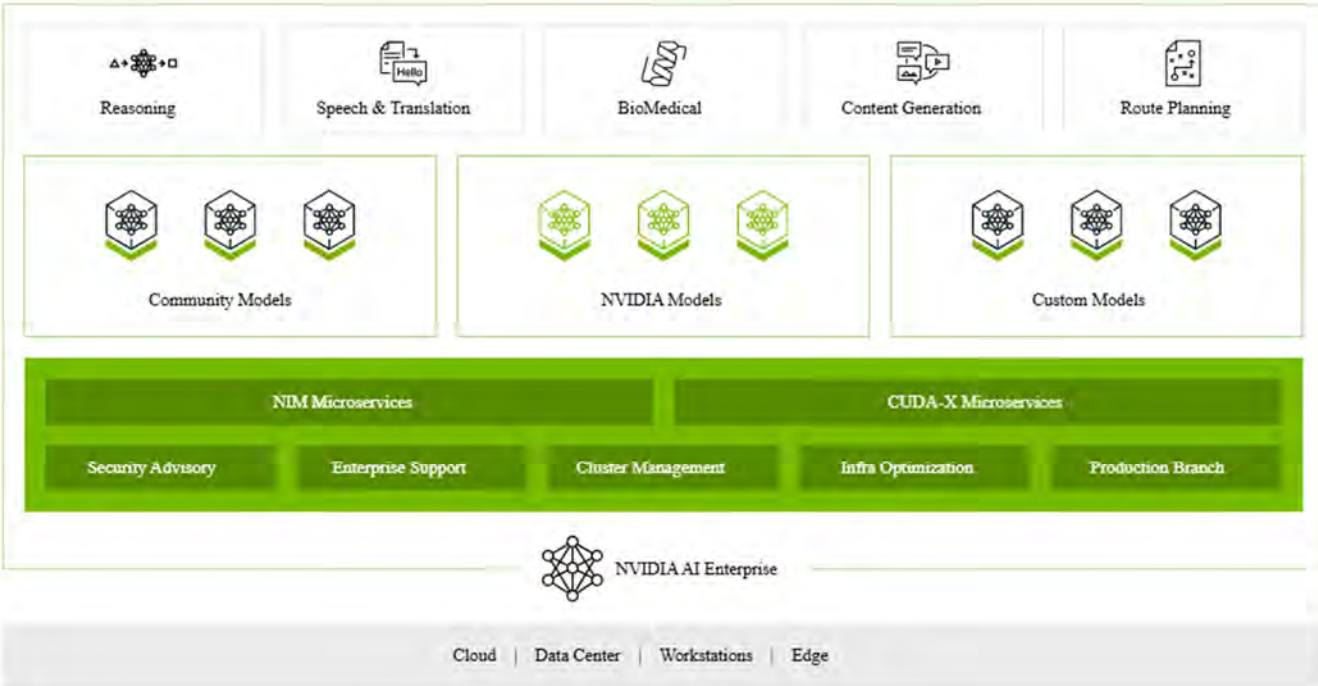
Key Features	Accused Products						
cuDNN 9.5.1 for CUDA 11.x	11.8	Yes	>= 450.80.02	>=452.39	› 9.0 [3]	› NVIDIA Hopper [3]	
					› 8.9 [3]		
					› 8.6	› NVIDIA Ada Lovelace architecture [3]	
					› 8.0		
					› 7.5		
					› 7.0	› NVIDIA Ampere architecture	
					› 6.1		
					› 6.0	› NVIDIA Turing	
					› 5.0	› NVIDIA Volta	
							› NVIDIA Pascal
		› NVIDIA Maxwell					
<p>(See https://docs.nvidia.com/deeplearning/cudnn/latest/reference/support-matrix.html#gpu-cuda-toolkit-and-cuda-driver-requirements (emphasis added).)</p> <p>› If we build our CUDA application using CUDA 11.0, can it continue to be used with newer NVIDIA drivers (such as CUDA 11.1/R455, 11.x etc.)? Or is it only the other way around?</p> <p>Drivers have always been backwards compatible with CUDA. This means that a CUDA 11.0 application will be compatible with R450 (11.0), R455 (11.1) and beyond. CUDA applications typically statically include all the libraries (for example cudart, CUDA math libraries such as cuBLAS, cuFFT) they need, so they should work on new drivers or CUDA Toolkit installations.</p> <p>In other words, since CUDA is backward compatible, existing CUDA applications can continue to be used with newer CUDA versions.</p> <p>(See https://docs.nvidia.com/deploy/cuda-compatibility/ (emphasis added).)</p>							

Key Features	Accused Products
	<p>As shown below, the GPU and “Compute Capability” are displayed for the Nvidia GPU products. As an example, the Nvidia Tesla M60 GPU based on the Nvidia Maxwell architecture has a compute capability of 5.2. As previously stated, the Accused Products include all Nvidia GPUs implementing the Maxwell architecture. All of the accused Nvidia GPUs are compatible with CUDA and GPU-acceleration libraries such as cuDNN, that implement infringing functionality as shown throughout this claim chart, <i>infra</i>. The accused Nvidia GPUs and their respective CUDA compute capability for compatibility with Nvidia’s CUDA and CUDA-based GPU-acceleration libraries, such as cuDNN, include the following:</p> <p>Maxwell: Tesla M60 (5.2), Tesla M40 (5.2), Tesla M10 (5.0), Quadro M series GPUs and Laptop GPUs (5.2); Quadro M6000 24GB, Quadro M6000 (12GB), Quadro M5000, Quadro M5500, Quadro M4000, Quadro M2000, Quadro M2200) (5.0); Quadro M5000M, Quadro M4000M, Quadro M3000M, Quadro M2000M, Quadro M1200, Quadro M1000M, Quadro M620, Quadro M600M, Quadro M520, Quadro M500M), NVS 810 GPU (5.0), Tesla M6 (5.2), GTX Titan X GPU (5.2), GeForce GTX 900 series GPUs and Laptop GPUs (5.2); GeForce GTX 980Ti, GeForce GTX 980, GeForce GTX 970, GeForce GTX 960, GeForce GTX 980M, GeForce GTX 970M, GeForce GTX 965M) (5.0); GeForce GTX 960M, GeForce GTX 950M), GeForce GTX 700 series GPUs and Laptop GPUs (5.0); GeForce GTX 750 Ti, GeForce GTX 750), GeForce MX130 and MX110 Laptop GPUs (5.0).</p> <p>(See https://developer.nvidia.com/cuda-gpus; https://arnon.dk/matching-sm-architectures-arch-and-encode-for-various-nvidia-cards/; https://www.nvidia.com/en-us/design-visualization/quadro/; https://developer.nvidia.com/maxwell-compute-architecture; https://www.nvidia.com/en-us/geforce/gaming-laptops/mx-330/; https://wccftch.com/nvidia-geforce-mx450-turing-discrete-notebook-gpu-gddr6-pcie-4/; https://developer.nvidia.com/cuda-gpus.)</p> <p>In addition to Nvidia’s GPUs, the Accused Products include Nvidia’s supercomputers, data centers, servers, workstations, and cloud services that embody computer systems specifically designed for GPU acceleration using the Maxwell architecture.</p> <p>Nvidia AI Enterprise is also deployed on Nvidia-Certified Systems “servers, workstations and laptop certified to accelerate AI workloads” utilizing Nvidia GPUs; and on cloud platforms utilizing Nvidia GPUs. (See https://www.nvidia.com/en-us/data-center/products/ai-enterprise/.) Nvidia’s NVIDIA-</p>

Key Features	Accused Products																																													
	<p>Certified Systems “are rigorously tested and validated to deliver enterprise-grade performance, manageability, scalability, and security for AI and accelerated computing.” (See https://www.nvidia.com/en-us/data-center/products/certified-systems/ (emphasis added).)</p> <p>An exemplary list of “NVIDIA-Certified Systems - Data Center Servers” is shown below, comprising Maxwell-architecture GPUs Tesla M6 and M60.</p> <div><h2>Supported Servers</h2><p>The following tables contain the supported servers and models using NVIDIA GRID K1 and NVIDIA GRID K2 (Table 1), Tesla M6 (Table 2), Tesla P100 (Table 3) Tesla M60 (Table 4), and Tesla P4 (Table 5).</p><p>Table 2. Tesla M6 Supported Servers</p><table><tr><th>Manufacturer</th><th>Model</th><th>Rack Units</th><th>Node per Chassis</th><th>Tesla M6</th></tr><tr><td>Cisco</td><td>UCS B200 M4</td><td>6</td><td>8</td><td>1</td></tr><tr><td>HPE</td><td>ProLiant WS460c Gen9</td><td>10</td><td>16</td><td>1 or 4**</td></tr><tr><td>HPE</td><td>ProLiant WS460c Gen8</td><td>10</td><td>16</td><td>1 or 4**</td></tr><tr><td>HPE</td><td>Synergy 480 Gen10</td><td>10</td><td>12</td><td>1 or 7**</td></tr><tr><td>HPE</td><td>Synergy 480 Gen9</td><td>10</td><td>12</td><td>1 or 7**</td></tr><tr><td>Amulet Hotkey</td><td>CoreStation VM630</td><td>10</td><td>16</td><td>1</td></tr><tr><td>Amulet Hotkey</td><td>Corestation VM640</td><td>10</td><td>16</td><td>1</td></tr></table><p>Note: **With expansion chassis</p><p>Table 4. Tesla M60 Supported Servers</p><table><tr><th>Manufacturer</th><th>Model</th><th>Rack Units</th><th>Node per Chassis</th><th>Tesla M60</th></tr></table></div>	Manufacturer	Model	Rack Units	Node per Chassis	Tesla M6	Cisco	UCS B200 M4	6	8	1	HPE	ProLiant WS460c Gen9	10	16	1 or 4**	HPE	ProLiant WS460c Gen8	10	16	1 or 4**	HPE	Synergy 480 Gen10	10	12	1 or 7**	HPE	Synergy 480 Gen9	10	12	1 or 7**	Amulet Hotkey	CoreStation VM630	10	16	1	Amulet Hotkey	Corestation VM640	10	16	1	Manufacturer	Model	Rack Units	Node per Chassis	Tesla M60
Manufacturer	Model	Rack Units	Node per Chassis	Tesla M6																																										
Cisco	UCS B200 M4	6	8	1																																										
HPE	ProLiant WS460c Gen9	10	16	1 or 4**																																										
HPE	ProLiant WS460c Gen8	10	16	1 or 4**																																										
HPE	Synergy 480 Gen10	10	12	1 or 7**																																										
HPE	Synergy 480 Gen9	10	12	1 or 7**																																										
Amulet Hotkey	CoreStation VM630	10	16	1																																										
Amulet Hotkey	Corestation VM640	10	16	1																																										
Manufacturer	Model	Rack Units	Node per Chassis	Tesla M60																																										

Key Features	Accused Products				
	Cisco	UCS C240 M5	2	1	2
	Cisco	UCS C240 M4	2	1	2
	Cisco	HyperFlex HX240c M5	2	1	2
	Cisco	HyperFlex HX240c M4	2	1	1
	Cisco	UCS C480 M5	4	1	6
	Cisco	UCS C460 M4	4	1	2
	Dell	PowerEdge C4130	1	1	4
	Dell	PowerEdge R740	2	1	3
	Dell	PowerEdge R740xd	2	1	3
	Dell	PowerEdge R730	2	1	2
	Dell	XC740xd	2	1	3
	Dell	XC730-16G	2	1	2
	Dell	Dell EMC VxRail V470	2	1	2
	Dell	Dell EMC VxRail V470F	2	1	2
	Dell	Dell EMC VxRail V570	2	1	3
	Dell	Dell EMC VxRail V570F	2	1	3
	Dell	EMC VxRail V570	2	1	3
	Dell	Precision Appliance for Wyse	2	1	2
	Dell	PowerEdge T640	5	1	4
	Dell	PowerEdge T630	5	1	4
	HPE	Apollo XL190r Gen9	2	2	2
	HPE	ProLiant DL380 Gen10	2	1	3
	HPE	ProLiant DL380 Gen9	2	1	2
	HPE	Hyper Converged 380	2	1	2
	HPE	ProLiant DL580 Gen9	4	1	4
	HPE	Apollo XL250a Gen9	5	5	2
	HPE	ProLiant WS460c Gen9	10	16	1**
	HPE	Synergy 480 Gen10	10	12	2**
	HPE	Synergy 480 Gen9	10	12	1
	Lenovo	ThinkSystem SR650	2	1	2
	Lenovo	ThinkSystem SD530/D2	2	2	2
	Lenovo	ThinkSystem SR860	4	1	2
	Lenovo	System x3650 M5	2	1	2

Key Features	Accused Products																																																																																																							
		<table><tr><th>Manufacturer</th><th>Model</th><th>Rack Units</th><th>Node per Chassis</th><th>Tesla M60</th></tr><tr><td>Lenovo</td><td>Converged HX3510-G</td><td>2</td><td>1</td><td>2</td></tr><tr><td>Lenovo</td><td>ThinkAgile HX3520-G</td><td>2</td><td>1</td><td>2</td></tr><tr><td>Lenovo</td><td>ThinkAgile HX3521-G</td><td>2</td><td>1</td><td>2</td></tr><tr><td>Lenovo</td><td>System x3850 X6</td><td>4</td><td>1</td><td>2</td></tr><tr><td>Lenovo</td><td>ThinkStation P710</td><td>4</td><td>1</td><td>1</td></tr><tr><td>Lenovo</td><td>System x3500 M5</td><td>5</td><td>1</td><td>2</td></tr><tr><td>Lenovo</td><td>ThinkStation P910</td><td>5</td><td>1</td><td>1</td></tr><tr><td>Lenovo</td><td>NeXtScale nx360 M5</td><td>6</td><td>4</td><td>4</td></tr><tr><td>Lenovo</td><td>System x3950 X6</td><td>8</td><td>1</td><td>4</td></tr><tr><td>Supermicro</td><td>SYS-1018R-WR</td><td>1</td><td>1</td><td>1</td></tr><tr><td>Supermicro</td><td>SYS-1018GR / 5018GR</td><td>1</td><td>1</td><td>2</td></tr><tr><td>Supermicro</td><td>SYS-1028GQ</td><td>1</td><td>1</td><td>4</td></tr><tr><td>Supermicro</td><td>SYS-1028GR</td><td>1</td><td>1</td><td>3</td></tr><tr><td>Supermicro</td><td>SYS-1028U / 6018U</td><td>1</td><td>1</td><td>1</td></tr><tr><td>Supermicro</td><td>SYS-1029U / 6019U</td><td>1</td><td>1</td><td>1</td></tr><tr><td>Supermicro</td><td>SYS-2028GR</td><td>2</td><td>1</td><td>4</td></tr><tr><td>Supermicro</td><td>SYS-2028TP-DC1FR</td><td>2</td><td>2</td><td>1</td></tr><tr><td>Supermicro</td><td>SYS-2028U / 6028U</td><td>2</td><td>1</td><td>2</td></tr><tr><td>Supermicro</td><td>SYS-F628R3</td><td>4</td><td>4</td><td>3</td></tr></table>	Manufacturer	Model	Rack Units	Node per Chassis	Tesla M60	Lenovo	Converged HX3510-G	2	1	2	Lenovo	ThinkAgile HX3520-G	2	1	2	Lenovo	ThinkAgile HX3521-G	2	1	2	Lenovo	System x3850 X6	4	1	2	Lenovo	ThinkStation P710	4	1	1	Lenovo	System x3500 M5	5	1	2	Lenovo	ThinkStation P910	5	1	1	Lenovo	NeXtScale nx360 M5	6	4	4	Lenovo	System x3950 X6	8	1	4	Supermicro	SYS-1018R-WR	1	1	1	Supermicro	SYS-1018GR / 5018GR	1	1	2	Supermicro	SYS-1028GQ	1	1	4	Supermicro	SYS-1028GR	1	1	3	Supermicro	SYS-1028U / 6018U	1	1	1	Supermicro	SYS-1029U / 6019U	1	1	1	Supermicro	SYS-2028GR	2	1	4	Supermicro	SYS-2028TP-DC1FR	2	2	1	Supermicro	SYS-2028U / 6028U	2	1	2	Supermicro	SYS-F628R3	4	4	3		
Manufacturer	Model	Rack Units	Node per Chassis	Tesla M60																																																																																																				
Lenovo	Converged HX3510-G	2	1	2																																																																																																				
Lenovo	ThinkAgile HX3520-G	2	1	2																																																																																																				
Lenovo	ThinkAgile HX3521-G	2	1	2																																																																																																				
Lenovo	System x3850 X6	4	1	2																																																																																																				
Lenovo	ThinkStation P710	4	1	1																																																																																																				
Lenovo	System x3500 M5	5	1	2																																																																																																				
Lenovo	ThinkStation P910	5	1	1																																																																																																				
Lenovo	NeXtScale nx360 M5	6	4	4																																																																																																				
Lenovo	System x3950 X6	8	1	4																																																																																																				
Supermicro	SYS-1018R-WR	1	1	1																																																																																																				
Supermicro	SYS-1018GR / 5018GR	1	1	2																																																																																																				
Supermicro	SYS-1028GQ	1	1	4																																																																																																				
Supermicro	SYS-1028GR	1	1	3																																																																																																				
Supermicro	SYS-1028U / 6018U	1	1	1																																																																																																				
Supermicro	SYS-1029U / 6019U	1	1	1																																																																																																				
Supermicro	SYS-2028GR	2	1	4																																																																																																				
Supermicro	SYS-2028TP-DC1FR	2	2	1																																																																																																				
Supermicro	SYS-2028U / 6028U	2	1	2																																																																																																				
Supermicro	SYS-F628R3	4	4	3																																																																																																				
	<p>(See https://images.nvidia.com/content/grid/pdf/DA-09018-001_v04.pdf (emphasis added).)</p> <p>In addition to the aforementioned Nvidia GPUs, supercomputers, datacenters, servers, and hardware, the Accused Products further include Nvidia’s software, platforms, and services for accelerated computing, including Nvidia’s proprietary CUDA platform for parallel computing and Nvidia AI Enterprise. Nvidia AI Enterprise is an operating system for enterprise AI applications that provides accelerated computing services based on Nvidia GPUs and using the CUDA platform, including Nvidia NIM Microservices (which includes Nvidia GPUs and utilizes the CUDA platform and, e.g., libraries cuDNN, cuBLAS, DALI, TensorRT and TensorRT-LLM) and the CUDA-X Microservices.</p>																																																																																																							

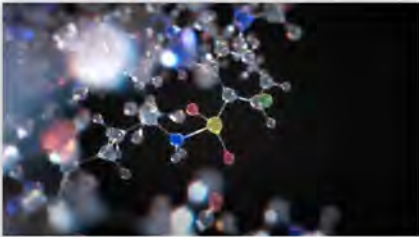
Key Features	Accused Products
	 <p>The diagram illustrates the NVIDIA AI Enterprise architecture. At the top, five application areas are shown: Reasoning, Speech & Translation, BioMedical, Content Generation, and Route Planning. Below these are three model categories: Community Models, NVIDIA Models, and Custom Models. The core of the architecture consists of NIM Microservices and CUDA-X Microservices. Below these are five support and management components: Security Advisory, Enterprise Support, Cluster Management, Infra Optimization, and Production Branch. The entire system is branded as NVIDIA AI Enterprise and is designed to run across four deployment environments: Cloud, Data Center, Workstations, and Edge.</p> <p>(See https://www.nvidia.com/en-us/data-center/products/ai-enterprise/; see also https://docs.nvidia.com/ai-enterprise/index.html#overview.)</p>

Key Features	Accused Products
	<div data-bbox="585 233 1705 293"> <h2>NVIDIA NIM for optimized AI inference</h2> </div> <div data-bbox="585 329 1730 422"> <p>NVIDIA NIM is designed to bridge the gap between the complex world of AI development and the operational needs of enterprise environments, enabling 10-100X more enterprise application developers to contribute to AI transformations of their companies.</p> </div> <div data-bbox="774 456 1539 777"> <p>The diagram illustrates the NVIDIA NIM architecture as a multi-layered stack. At the top, a green hexagonal block represents the core inference engine, composed of smaller green cubes. Below this, a blue square block with a white circular logo represents the containerization layer. At the base is a grey and green circuit board representing the hardware. Various components are labeled with lines pointing to specific parts of the stack:</p> <ul style="list-style-type: none"> Industry Standard APIs: Text, Speech, Image, Video, 3D, Biology Triton Inference Server: cuDPP, cuXCLUDA, DALI, NCCL, Post-Processing Decoder Cloud Native Stack: GPU Operator, Network Operator Enterprise Management: Health Check, Identity, Metrics, Monitoring, Secrets Management Kubernetes TensorRT and TensorRT-LLM: cuBLAS, cuDNN, in-Flight Batching, Memory Optimization, FP8 Quantization Optimized Model: Single GPU, Multi-GPU, Multi-Node Customization Cache: P-Tuning, LoRA, Model Weights NVIDIA CUDA </div> <div data-bbox="606 805 1709 850"> <p>Figure 1. NVIDIA NIM is a containerized inference microservice including industry-standard APIs, domain-specific code, optimized inference engines, and enterprise runtime</p> </div> <div data-bbox="564 873 1894 943"> <p>(See https://developer.nvidia.com/blog/nvidia-nim-offers-optimized-inference-microservices-for-deploying-ai-models-at-scale/.)</p> </div> <div data-bbox="564 982 1894 1198"> <p>The Nvidia AI Enterprise cloud-based “End-to-End Software Platform for Production AI” includes “the full stack of NVIDIA AI software,” including Nvidia RAPID Accelerator for Apache Spark for speed data processing; Nvidia NeMo (Neural Modules) for building, customizing, and deploying generative AI; NVIDIA TensorRT, TensorRT-LLM, and NVIDIA Triton Inference Server for AI “inference” (e.g., making AI predictions or solutions) at scale; and Base Command Manager Essentials for managing AI clusters across edge devices and data centers.</p> </div>

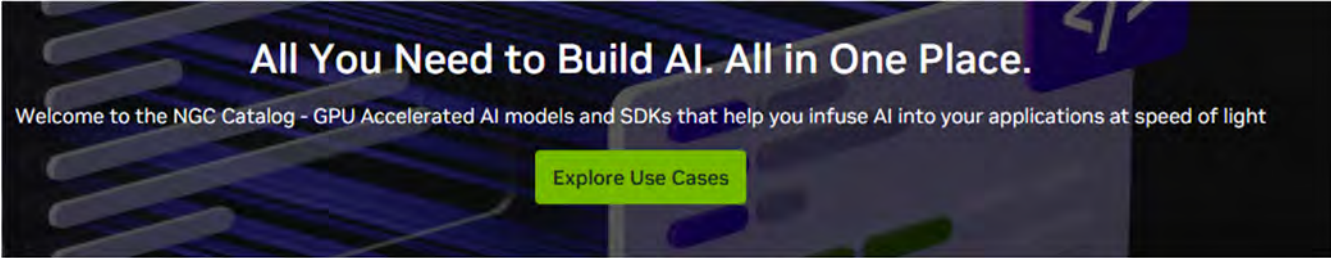
Key Features	Accused Products
	<p data-bbox="772 250 1690 285" style="text-align: center;">The End-to-End Software Platform for Production AI</p> <p data-bbox="583 315 1879 358" style="text-align: center;">NVIDIA AI Enterprise offers best-in-class development tools, frameworks, and pretrained models for AI practitioners and reliable management and orchestration for IT professionals to ensure performance, high availability, and security.</p> <div data-bbox="583 412 1879 688" style="display: flex; align-items: center;"> <div style="flex: 1;">  </div> <div style="flex: 1; padding-left: 20px;"> <p>Accelerate your AI pipeline with the full stack of NVIDIA AI software.</p> <ul style="list-style-type: none"> > Speed data processing with <u>NVIDIA RAPIDS™ Accelerator for Apache Spark</u>, > Develop custom generative AI with <u>NVIDIA NeMo</u>, an end-to-end platform that delivers enterprise-ready models with precise data curation, cutting-edge customization, RAG, and accelerated performance. > Achieve inference at scale with NIMs built on <u>NVIDIA TensorRT™</u>, <u>TensorRT-LLM</u>, and <u>NVIDIA Triton™ Inference Server</u>. > Manage AI clusters at scale, across the edge and data center, with <u>NVIDIA Base Command™ Manager Essentials</u>. </div> </div> <p data-bbox="567 699 1848 915">(See https://www.nvidia.com/en-us/data-center/products/ai-enterprise/; see also https://www.nvidia.com/en-us/deep-learning-ai/solutions/data-science/apache-spark-3/; https://www.nvidia.com/en-us/ai-data-science/generative-ai/nemo-framework/; https://developer.nvidia.com/tensorrt; https://www.nvidia.com/en-us/ai-data-science/products/triton-inference-server/; https://www.nvidia.com/en-us/ai-data-science/products/base-command-manager-essentials/.)</p> <p data-bbox="567 956 1894 1062">The Accused Products further Nvidia software, platforms, and services that implement and exploit GPU acceleration systems and provide GPU acceleration service, including Nvidia Omniverse, Nvidia Drive, Nvidia Isaac Sim, Clara, AI Foundation models, and Nvidia NGC.</p> <p data-bbox="567 1102 1894 1321">Nvidia Omniverse is a cloud-based “industrial digitalization and physical AI simulation” platform and operating system for building virtual world simulation applications, available as a software subscription for enterprise use. Using Nvidia RTX GPUs, Omniverse helps industries building physical AI and industrial applications with applications for “Autonomous Vehicle Simulation,” “Configurator Development,” “Reinforcement Learning” (e.g., training robots in virtual environments) “Synthetic Data Generation” (e.g., virtual model training), and “Virtual Facility Integration.”</p>

Key Features	Accused Products
	<p data-bbox="1171 245 1276 269">Overview</p> <h2 data-bbox="600 297 1848 391">Design, Develop, and Deploy the Next Era of 3D Applications and Services</h2> <p data-bbox="581 431 1871 553">NVIDIA Omniverse™ is a platform of APIs, SDKs, and services that enable developers to easily integrate <u>Universal Scene Description (OpenUSD)</u> and NVIDIA RTX™ rendering technologies into existing software tools and simulation workflows for building AI systems.</p> <p data-bbox="569 573 1186 605">(See https://www.nvidia.com/en-us/omniverse/.)</p> <p data-bbox="569 646 1894 716">Nvidia Isaac Sim, also based off Nvidia Omniverse, is a robotics simulation platform that implements Nvidia’s GPU acceleration for training and verifying AI models (e.g., neural networks) virtually.</p> <h2 data-bbox="583 776 1098 833">NVIDIA Isaac Sim</h2> <p data-bbox="581 914 1835 1044">The NVIDIA Isaac Sim™ robotics developer simulation platform and reference application is designed to help developers design, simulate, test, and train AI-based robots and autonomous machines in a physically based virtual environment.</p> <p data-bbox="581 1114 1820 1243">Isaac Sim, built on <u>NVIDIA Omniverse™</u>, is fully extensible. This means you can build your own <u>Universal Scene Description (OpenUSD)</u>-based custom simulators or integrate core Isaac Sim technologies into your existing testing and validation pipelines.</p> <p data-bbox="581 1317 1730 1398">Ready to get started? Download the SDK through the Omniverse Launcher, pulled as a container from NGC, or install it using a command line with PIP.</p>

Key Features	Accused Products
	<p>(See https://developer.nvidia.com/isaac/sim.)</p> <p>In addition, Nvidia Clara is “is a suite of computing platforms, software, and services that powers AI solutions for healthcare and life sciences, from imaging and instruments to genomics and drug discovery.” AI-powered solution applications provided by Clara include “Biopharma,” “Digital Health,” “Medical Devices,” “Medical Imaging,” and “Genomics.”</p>  <p>The image shows the NVIDIA Clara logo in large white letters on a black background. Below the logo, a white text box contains the following description: 'NVIDIA Clara™ is a suite of computing platforms, software, and services that powers AI solutions for healthcare and life sciences, from imaging and instruments to genomics and drug discovery.'</p>

Key Features	Accused Products
	<div data-bbox="919 248 1554 289"> <h3>AI-Powered Solutions for Healthcare</h3> </div> <div data-bbox="581 313 997 548">  </div> <div data-bbox="600 576 707 600"> <p>Biopharma</p> </div> <div data-bbox="600 636 972 730"> <p>NVIDIA Clara for Biopharma is a collection of AI and accelerated computing frameworks, applications, <u>generative AI platforms and services</u>, and pretrained models for accelerating drug discovery pipelines.</p> </div> <div data-bbox="600 813 930 836"> <p>Explore NVIDIA Clara for Biopharma ></p> </div> <div data-bbox="1024 313 1440 548">  </div> <div data-bbox="1047 576 1182 600"> <p>Digital Health</p> </div> <div data-bbox="1047 636 1417 756"> <p>NVIDIA Clara Digital Health is a comprehensive AI toolkit for developing personalized healthcare applications that enhance clinician-patient interactions, streamline documentation, and extract valuable insights from real-world data.</p> </div> <div data-bbox="1047 813 1400 836"> <p>Explore NVIDIA Clara for Digital Health ></p> </div> <div data-bbox="1470 313 1885 548">  </div> <div data-bbox="1491 576 1648 600"> <p>Medical Devices</p> </div> <div data-bbox="1491 636 1864 756"> <p>NVIDIA Clara for Medical Devices is a domain-specific AI computing platform that delivers the full-stack infrastructure needed for building scalable, software-defined medical devices that can process streaming data at the edge in real time.</p> </div> <div data-bbox="1491 790 1850 836"> <p>Explore NVIDIA Clara for Medical Devices ></p> </div> <div data-bbox="569 876 1117 912"> <p>(See https://www.nvidia.com/en-us/clara/.)</p> </div> <div data-bbox="569 950 1894 1166"> <p>In addition, Nvidia Foundation models is a “curated collection of enterprise-grade pretrained models” that “give[s] developers a running start for bringing custom generative AI to their enterprise applications.” For example, Nvidia provides “leading community models such as Llama 2, Stable Diffusion XL and Mistral,” and “models [that] have been optimized with NVIDIA TensorRT-LLM to deliver the highest throughput and lowest latency and to run at scale on any NVIDIA GPU-accelerated stack.”</p> </div>

Key Features	Accused Products
	<p>Today's landscape of free, open-source <u>large language models</u> (LLMs) is like an all-you-can-eat buffet for enterprises. This abundance can be overwhelming for developers building custom generative AI applications, as they need to navigate unique project and business requirements, including compatibility, security and the data used to train the models.</p> <p><u>NVIDIA AI Foundation Models</u> — a curated collection of enterprise-grade pretrained models — give developers a running start for bringing custom generative AI to their enterprise applications.</p> <p>NVIDIA-Optimized Foundation Models Speed Up Innovation</p> <p>NVIDIA AI Foundation Models can be experienced through a simple user interface or API, directly from a browser. Additionally, these models can be accessed from NVIDIA AI Foundation Endpoints to test model performance from within their enterprise applications.</p> <p>Available models include leading community models such as Llama 2, Stable Diffusion XL and Mistral, which are formatted to help developers streamline customization with proprietary data. Additionally, models have been optimized with <u>NVIDIA TensorRT-LLM</u> to deliver the highest throughput and lowest latency and to run at scale on any NVIDIA GPU-accelerated stack. For instance, the Llama 2 model optimized with TensorRT-LLM runs nearly <u>2x faster</u> on NVIDIA H100.</p> <p>The new NVIDIA family of <u>Nemotron-3 8B foundation models</u> supports the creation of today's most advanced enterprise chat and Q&A applications for a broad range of industries, including healthcare, telecommunications and financial services.</p> <p>(See https://blogs.nvidia.com/blog/custom-generative-ai-model-development/ (emphasis added); see also https://www.nvidia.com/en-us/ai-data-science/foundation-models/.)</p> <p>Nvidia NGC is Nvidia's catalog of "GPU Accelerated AI models and SDKs." As shown below, examples include "LLMs [<u>large language models</u>, a type of artificial intelligence program that can recognize and</p>

Key Features	Accused Products
	<p>generate text, among other tasks] optimized for RTX PCs” which includes a “collection of TensorRT-LLM accelerated Windows RTX PC LLM models.” Furthermore, Nvidia provides “Documentation,” “AI Enterprise Documentation,” “Enterprise Support,” and a “Licensing Portal.”</p>  <p>(See https://catalog.ngc.nvidia.com/.)</p>
<p>[16.1] receiving, by an accelerator, first input data from the central processing unit;</p>	<p>The Accused Products perform a method that includes <i>receiving, by an accelerator, first input data from the central processing unit</i>.</p> <p>For instance, the “CUDA programming model” includes programmatic functions, primitives, and executable libraries for CPUs and GPUs that are used by the Accused Products to perform numerical simulations. “The host is the CPU [(e.g., <i>central processing unit</i>)] available in the system” and “system memory associated with the CPU is called host memory.” “The GPU is called a device and GPU memory likewise called device memory” (e.g., <i>accelerator</i>). As an example, the first main CUDA program execution step is “[c]opy[ing] the input data from host [CPU] memory to device [GPU] memory, also known as host-to-device transfer” (e.g., <i>receiving, by an accelerator, first input data from the central processing unit</i>).</p>

Key Features	Accused Products
	<p>Let me introduce two keywords widely used in CUDA programming model: <u>host and device</u>.</p> <p><u>The host is the CPU available in the system</u>. The system memory associated with the CPU is called host memory. <u>The GPU is called a device</u> and GPU memory likewise called device memory.</p> <p>To execute any CUDA program, there are three main steps:</p> <ul style="list-style-type: none"> • <u>Copy the input data from host memory to device memory, also known as host-to-device transfer.</u> • Load the GPU program and execute, caching data on-chip for performance. • Copy the results from device memory to host memory, also called device-to-host transfer. <p>(See https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/ (emphasis added).)</p> <p>Examples of publicly available CUDA toolkit documentation for “the memory management functions of the CUDA runtime application programming interface” are shown below. Functions including “[c]op[y]ing data between host [CPU] and device [GPU]” (see “cudaMemcpy”; “cudaMemcpy2D”) and “[a]llocat[ing] an array on the device [GPU]” (see “cudaMallocArray”).</p> <p>6.11. Memory Management</p> <p>This section describes the memory management functions of the CUDA runtime application programming interface.</p> <p>Some functions have overloaded C++ API template versions documented separately in the C++ API Routines module.</p> <p>Functions</p> <p style="text-align: center;">* * * * *</p> <p><code>__host__ cudaError_t cudaMemcpy(void* dst , const void* src , size_t count , cudaMemcpyKind kind)</code> <u>Copies data between host and device.</u></p>

Key Features	Accused Products
	<pre> * * * * * __host__ cudaError_t cudaMallocArray (cudaArray_t* array , const cudaChannelFormatDesc* desc , size_t width , size_t height = 0, unsigned int flags = 0) Allocate an array on the device. * * * * * __host__ cudaError_t cudaMemcpy2D (void* dst , size_t dpitch , const void* src , size_t spitch , size_t width , size_t height , cudaMemoryKind kind) Copies data between host and device. </pre> <p>(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p>
<p>[16.2] transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation;</p>	<p>The Accused Products practice a method that includes <i>transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation.</i></p> <p>For instance, the GPU architecture of the Accused Products implement <i>memory</i>. As an example, Maxwell-architecture GPUs (listed, <i>supra</i>), such as the “GM204” GPU (Geforce GTX 980), include “memory controllers” (e.g., <i>accelerator controller</i>) and “DRAM” (e.g., <i>accelerator memory</i>).</p>

Key Features	Accused Products
	<p data-bbox="590 245 1188 277">GM204 Hardware Architecture In-Depth</p>  <p data-bbox="911 1179 1218 1198"><i>Figure 2: GM204 Full-chip block diagram</i></p>

Key Features	Accused Products
	<p>Like Fermi and Kepler, GM204 is composed of an array of Graphics Processing Clusters (GPCs), Streaming Multiprocessors (SMs), and memory controllers. GM204 consists of four GPCs, 16 Maxwell SMs (SMM), and four memory controllers. GeForce GTX 980 uses the full complement of these architectural components (if you are not well versed in these structures, we suggest you first read the Kepler and Fermi whitepapers).</p> <p>Another version of the chip, with 13 SMs, will ship concurrently and be called GeForce GTX 970. In the future we plan to offer additional products based on GM204 that will ship with different combinations of GPCs, SMs, and memory controllers to address various segments of the graphics market.</p> <p style="text-align: center;">* * * * *</p> <p>The memory subsystem has also been significantly revamped. GTX 980's memory clock is over 15% higher than GTX 680, and GM204's cache is larger and more efficient than Kepler's design, reducing the number of memory requests that have to be made to DRAM. Improvements in our implementation of memory compression provide a further benefit in reducing DRAM traffic—effectively amplifying the raw DRAM bandwidth in the system.</p>

Key Features	Accused Products
	<p data-bbox="577 240 940 264">Maxwell Streaming Multiprocessor</p> <div data-bbox="577 293 1041 1166"> </div> <p data-bbox="1058 289 1402 475">The SM is the heart of our GPUs. Almost every operation flows through the SM at some point in the rendering pipeline. Maxwell GPUs feature a new SM that's been designed to provide dramatically improved performance per watt than prior GeForce GPUs.</p> <p data-bbox="1058 500 1409 1125">Compared to GPUs based on our Kepler architecture, Maxwell's new SMM design has been reconfigured to improve efficiency. Each SMM contains four warp schedulers, and each warp scheduler is capable of dispatching two instructions per warp every clock. Compared to Kepler's scheduling logic, we've integrated a number of improvements in the scheduler to further reduce redundant re-computation of scheduling decisions, improving energy efficiency. We've also integrated a completely new datapath organization. Whereas Kepler's SM shipped with 192 CUDA Cores—a non-power-of-two organization—the Maxwell SMM is partitioned into four distinct 32-CUDA core processing blocks (128 CUDA cores total per SM), each with its own dedicated resources for scheduling and instruction buffering. This new configuration in Maxwell aligns with warp size, making it easier to utilize efficiently and saving area</p> <p data-bbox="1396 1149 1409 1170">8</p> <p data-bbox="583 1182 1037 1219"><i>Figure 3: GM204 SMM Diagram (GM204 also features 4 DP units per SMM, which are not depicted on this diagram)</i></p> <p data-bbox="569 1230 1726 1263">(See https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf (emphasis added).)</p> <p data-bbox="569 1304 1894 1406">As previously stated, the Accused Products implement CUDA, Nvidia's parallel computing platform. CUDA enables NVIDIA GPUs to be used for general purpose computing tasks and includes specialized GPU-acceleration libraries such as cuDNN. CUDA implements numerous “memory management</p>

Key Features	Accused Products
	<p>functions” that “[c]op[y] data between host [CPU] and device [GPU]” (e.g., <i>transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory</i>), including “cudaMemcpy,” “cudaMemcpy2D,” “cudaMemcpy2DArrayToArray,” “cudaMemcpy2DAsync,” “cudaMemcpy2DFromArray,” “cudaMemcpy2DFromArrayAsync,” “cudaMemcpy2DToArray,” “cudaMemcpy2DToArrayAsync,” and “cudaMemcpyAsync.”</p> <p>6.11. Memory Management</p> <p>This section describes the memory management functions of the CUDA runtime application programming interface.</p> <p>Some functions have overloaded C++ API template versions documented separately in the C++ API Routines module.</p> <p>Functions</p> <p style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	<pre> __host__ <u>cudaError_t</u> <u>cudaMemcpy</u> (void* dst , const void* src Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2D</u> (void* dst , size_t dpitch Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2DFromArray</u> (<u>cudaArray_t</u> d Copies data between host and device. __host__ __device__ <u>cudaError_t</u> <u>cudaMemcpy2DAsync</u> (void* dst , size_t d Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2DFromArray</u> (void* dst , size Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2DFromArrayAsync</u> (void* dst , Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2DToArray</u> (<u>cudaArray_t</u> dst , Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy2DToArrayAsync</u> (<u>cudaArray_t</u> d Copies data between host and device. __host__ <u>cudaError_t</u> <u>cudaMemcpy3D</u> (const <u>cudaMemcpy3DParms</u>* Copies data between 3D objects. __host__ __device__ <u>cudaError_t</u> <u>cudaMemcpy3DAsync</u> (const <u>cudaMemcpy3DPa Copies data between 3D objects. __host__ <u>cudaError_t</u> <u>cudaMemcpy3DPeer</u> (const <u>cudaMemcpy3DPee Copies memory between devices. __host__ <u>cudaError_t</u> <u>cudaMemcpy3DPeerAsync</u> (const <u>cudaMemcpy Copies memory between devices asynchronously. __host__ __device__ <u>cudaError_t</u> <u>cudaMemcpyAsync</u> (void* dst , const void Copies data between host and device. </u></u></u></pre> <p>(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p>Parameters used in CUDA include pointers “dst” (“Destination memory address”) and “src” (“Source memory address”). For example, CUDA function “cudaMemcpy” copies “bytes [data] from the memory area pointed to by src [source memory address pointer] to the memory area pointed to by dst [destination memory address pointer], where kind [type of transfer] specifies the direction of the copy” (e.g.,</p>

Key Features	Accused Products
	<p><i>referenced by first pointer). One of the destinations is “cudaMemcpyHostToDevice,” or host (CPU) to device (GPU) (e.g., transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory).</i></p> <pre data-bbox="583 394 1812 427">__host__ <u>cudaError_t</u> cudaMemcpy (void* dst , const void* src , size_t count , <u>cudaMemcpyKind</u> kind)</pre> <p data-bbox="611 446 1003 479"><u>Copies data between host and device.</u></p> <div data-bbox="606 508 1008 823" style="border: 2px solid red; padding: 5px;"> <p>Parameters</p> <p><u>dst</u> - Destination memory address</p> <p><u>src</u> - Source memory address</p> <p><u>count</u> - Size in bytes to copy</p> <p><u>kind</u> - Type of transfer</p> </div> <p>Returns <u>cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection</u></p> <p>Description Copies <u>count</u> bytes from the memory area pointed to by <u>src</u> to the memory area pointed to by <u>dst</u>, where <u>kind</u> specifies the <u>direction of the copy</u>, and must be one of <u>cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice, or cudaMemcpyDefault</u>. Passing <u>cudaMemcpyDefault</u> is recommended, in which case the type of transfer is inferred from the <u>pointer values</u>. However, <u>cudaMemcpyDefault</u> is only allowed on systems that support unified virtual addressing. Calling <u>cudaMemcpy()</u> with <u>dst</u> and <u>src</u> pointers that do not match the direction of the copy results in an undefined behavior.</p> <p>(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p>In a related example, CUDA function “cudaMemcpy2DArrayToArray” also “[c]opies data between host [CPU] and device [GPU]” and includes pointers “dst” and “src.” cudaMemcpy2DArrayToArray “[c]opies a matrix (height rows of width bytes each) from the CUDA array src [source memory address pointer] starting at hOffsetSrc [source starting Y offset (rows)] rows and wOffsetSrc [source starting X</p>

Key Features	Accused Products
	<p>or width offset (columns in bytes)] bytes from the upper left corner to the CUDA array dst [destination memory address pointer] starting at hOffsetDst [destination starting Y or height offset (rows)] rows and wOffsetDst [destination starting X offset (columns in bytes)] bytes from the upper left corner, where kind specifies the direction of the copy.” One of the destinations is, again, “cudaMemcpyHostToDevice,” or host (CPU) to device (GPU).</p> <pre data-bbox="577 456 1812 548">__host__ cudaError_t cudaMemcpy2DArrayToArray (cudaArray_t dst , size_t wOffsetDst , size_t hOffsetDst , cudaArray_const_t src , size_t wOffsetSrc , size_t hOffsetSrc , size_t width , size_t height , cudaMemcpyKind kind = cudaMemcpyDeviceToDevice)</pre> <p><u>Copies data between host and device.</u></p> <div data-bbox="594 641 1215 1325" style="border: 2px solid red; padding: 10px;"> <p>Parameters</p> <p>dst - Destination memory address</p> <p>wOffsetDst - Destination starting X offset (columns in bytes)</p> <p>hOffsetDst - Destination starting Y offset (rows)</p> <p>src - Source memory address</p> <p>wOffsetSrc - Source starting X offset (columns in bytes)</p> <p>hOffsetSrc - Source starting Y offset (rows)</p> <p>width - Width of matrix transfer (columns in bytes)</p> <p>height - Height of matrix transfer (rows)</p> <p>kind - Type of transfer</p> </div> <p>Returns cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection</p>

Key Features	Accused Products
	<p>Description</p> <p>Copies a matrix (<u>height</u> rows of <u>width</u> bytes each) from the CUDA array <u>src</u> starting at <u>hOffsetSrc</u> rows and <u>wOffsetSrc</u> bytes from the upper left corner to the CUDA array <u>dst</u> starting at <u>hOffsetDst</u> rows and <u>wOffsetDst</u> bytes from the upper left corner, where <u>kind</u> specifies the direction of the copy, and must be one of <u>cudaMemcpyHostToHost</u>, <u>cudaMemcpyHostToDevice</u>, <u>cudaMemcpyDeviceToHost</u>, <u>cudaMemcpyDeviceToDevice</u>, or <u>cudaMemcpyDefault</u>. Passing <u>cudaMemcpyDefault</u> is recommended, in which case the type of transfer is inferred from the <u>pointer values</u>. However, <u>cudaMemcpyDefault</u> is only allowed on systems that support unified virtual addressing. <u>wOffsetDst + width</u> must not exceed the width of the CUDA array <u>dst</u>. <u>wOffsetSrc + width</u> must not exceed the width of the CUDA array <u>src</u>.</p> <p>(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p>As a further example, CUDA function “cudaMalloc” is used for “[a]llocat[ing] memory on the device [GPU].” Parameters include pointer “devPtr” (“Pointer to allocated device memory”). The cudaMalloc function “[a]llocates size bytes [data] of linear memory on the device [GPU] and returns in *devPtr a pointer to the allocated memory [of the GPU]” (e.g., <i>the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation</i>).</p>

Key Features	Accused Products
	<pre data-bbox="590 245 1812 277">__host__ __device__ <u>cudaError_t</u> cudaMalloc (void** devPtr , size_t size)</pre> <p data-bbox="590 297 915 326"><u>Allocate memory on the device.</u></p> <div data-bbox="590 363 1066 553" style="border: 2px solid red; padding: 5px;"> <p data-bbox="590 375 722 404">Parameters</p> <p data-bbox="590 423 680 453">devPtr</p> <p data-bbox="680 453 1045 482">- Pointer to allocated device memory</p> <p data-bbox="590 488 659 518">size</p> <p data-bbox="680 518 1037 547">- Requested allocation size in bytes</p> </div> <p data-bbox="590 602 684 631">Returns</p> <p data-bbox="590 634 1260 664"><u>cudaSuccess</u>, <u>cudaErrorInvalidValue</u>, <u>cudaErrorMemoryAllocation</u></p> <p data-bbox="590 709 722 738">Description</p> <p data-bbox="590 742 1770 823">Allocates <u>size</u> bytes of linear memory on the device and returns in <u>*devPtr</u> a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. <u>cudaMalloc()</u> returns <u>cudaErrorMemoryAllocation</u> in case of failure.</p> <p data-bbox="590 846 1692 875">The device version of <u>cudaFree</u> cannot be used with a <u>*devPtr</u> allocated using the host API, and vice versa.</p> <p data-bbox="569 885 1894 956">(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p data-bbox="569 995 1894 1248">In a related example, the CUDA function “cudaMallocArray” is used for “[a]llocat[ing] an array [pointer] on the device [GPU].” Parameters include pointer “array” (“Pointer to allocated array in device memory”), “desc” (“Requested channel format”), “width” (“Requested array allocation width”), “height” (“Requested array allocation height”), and “flags” (“Requested properties of allocated array”). The cudaMallocArray function “[a]llocates a CUDA array according to the cudaChannelFormatDesc structure desc [requested channel format] and returns a handle to the new CUDA array in *array [pointer to allocated array in device or GPU memory].”</p>

Key Features	Accused Products
	<pre data-bbox="583 240 1812 297">__host__ <u>cudaError_t</u> cudaMallocArray (<u>cudaArray_t</u>* array , const <u>cudaChannelFormatDesc</u>* desc , size_t width , size_t height = 0, unsigned int flags = 0)</pre> <p data-bbox="604 318 909 342"><u>Allocate an array on the device.</u></p> <div data-bbox="604 386 1115 764" style="border: 2px solid red; padding: 10px;"> <p>Parameters</p> <p><code>array</code> - Pointer to allocated array in device memory</p> <p><code>desc</code> - Requested channel format</p> <p><code>width</code> - Requested array allocation width</p> <p><code>height</code> - Requested array allocation height</p> <p><code>flags</code> - Requested properties of allocated array</p> </div> <p>Returns <u><code>cudaSuccess</code>, <code>cudaErrorInvalidValue</code>, <code>cudaErrorMemoryAllocation</code></u></p> <p>Description <u>Allocates a CUDA array according to the <code>cudaChannelFormatDesc</code> structure <code>desc</code> and returns a handle to the new CUDA array in <code>*array</code>.</u></p> <p>(See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p>As a further example, device “shared memory” is partitioned into “memory modules (banks).”</p> <h2 data-bbox="590 1190 1486 1247" style="color: green;">CUDA C++ Best Practices Guide</h2> <p data-bbox="590 1300 1881 1333">The <u>programming guide to using the CUDA Toolkit to obtain the best performance from <u>NVIDIA GPUs.</u></u></p> <p data-bbox="1173 1360 1285 1377" style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	<p>9.2.3. Shared Memory</p> <p>Because it is on-chip, shared memory has much higher bandwidth and lower latency than local and global memory - provided there are no bank conflicts between the threads, as detailed in the following section.</p> <p>9.2.3.1. Shared Memory and Memory Banks</p> <p>To achieve high memory bandwidth for concurrent accesses, shared memory is divided into equally sized memory modules (<u>banks</u>) that can be accessed simultaneously. Therefore, any memory load or store of n addresses that spans n distinct memory banks can be serviced simultaneously, yielding an effective bandwidth that is n times as high as the bandwidth of a single bank.</p> <p>However, if multiple addresses of a memory request map to the same memory bank, the accesses are serialized. The hardware splits a memory request that has bank conflicts into as many separate conflict-free requests as necessary, decreasing the effective bandwidth by a factor equal to the number of separate memory requests. The one exception here is when multiple threads in a warp address the same shared memory location, resulting in a broadcast. In this case, multiple broadcasts from different banks are coalesced into a single multicast from the requested shared memory locations to the threads.</p> <p>To minimize bank conflicts, it is important to understand how memory addresses map to memory banks and how to optimally schedule memory requests.</p> <p>On devices of compute capability 5.x or newer, each bank has a bandwidth of 32 bits every clock cycle, and successive 32-bit words are assigned to successive banks. The warp size is 32 threads and the number of banks is also 32, so bank conflicts can occur between any threads in the warp. See Compute Capability 5.x in the CUDA C++ Programming Guide for further details.</p> <p>(See https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#shared-memory/ (emphasis added).)</p>
<p>[16.3] performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data;</p>	<p>The Accused Products practice a method that includes <i>performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data.</i></p> <p>For instance, CUDA uses “streams” to execute a sequence of commands in order. As shown below, an exemplary CUDA function “cudaMemcpyAsync” is used to copy data between a host (CPU) and a device (GPU). “Each stream copies its portion of input array hostPtr [pointer for CPU] to array inputDevPtr in device [GPU] memory” (e.g., <i>performing, by at least one graphics processing unit</i>). The stream then “processes inputDevPtr on the device [GPU] by calling MyKernel(), and copies the result outputDevPtr back to the same portion of hostPtr” (e.g., <i>performing, by at least one graphics processing unit during</i></p>

Key Features	Accused Products
	<p><i>the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data).</i></p> <p>3.2.8.5.1. Creation and Destruction of Streams</p> <p>A stream is defined by creating a stream object and specifying it as the stream parameter to a sequence of kernel launches and host <-> device memory copies. The following code sample creates two streams and allocates an array <code>hostPtr</code> of <code>float</code> in page-locked memory.</p> <pre> cudaStream_t stream[2]; for (int i = 0; i < 2; ++i) cudaStreamCreate(&stream[i]); float* hostPtr; cudaMallocHost(&hostPtr, 2 * size); </pre> <p>Each of these streams is defined by the following code sample as a sequence of one memory copy from host to device, one kernel launch, and one memory copy from device to host:</p> <pre> for (int i = 0; i < 2; ++i) { cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i * size, size, cudaMemcpyHostToDevice, stream[i]); MyKernel <<<100, 512, 0, stream[i]>>> (outputDevPtr + i * size, inputDevPtr + i * size, size); cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i * size, size, cudaMemcpyDeviceToHost, stream[i]); } </pre> <p>Each stream copies its portion of input array <code>hostPtr</code> to array <code>inputDevPtr</code> in device memory, processes <code>inputDevPtr</code> on the device by calling <code>MyKernel()</code>, and copies the result <code>outputDevPtr</code> back to the same portion of <code>hostPtr</code>. Overlapping Behavior describes how the streams overlap in this example depending on the capability of the device. Note that <code>hostPtr</code> must point to page-locked host memory for any overlap to occur.</p> <p>(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#creation-and-destruction-of-streams (emphasis added).)</p>

Key Features	Accused Products
	<p>Additional examples of CUDA code are shown below for an “implementation of matrix multiplication that does take advantage of shared memory.” As shown, matrix multiplication function “MatMul” calculates “square sub-matrix Csub of C” from sub-matrix A and sub-matrix B. By implementing the matrix computation in this manner, CUDA “take[s] advantage of fast shared memory and save[s] a lot of global memory bandwidth.” Example code shown below demonstrates “Load A and B to device [GPU] memory” (e.g., <i>performing, by at least one graphics processing unit</i>), “Invoke kernel,” “Matrix multiplication kernel called by MatMul(),” and “Write [square sub-matrix] Csub to device [GPU] memory” (e.g., <i>performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data</i>).</p> 

Key Features	Accused Products
	<p>The following code sample is an implementation of matrix multiplication that does take advantage of shared memory. In this implementation, <u>each thread block is responsible for computing one square sub-matrix C_{sub} of C and each thread within the block is responsible for computing one element of C_{sub}.</u> As illustrated in Figure 9, <u>C_{sub} is equal to the product of two rectangular matrices: the sub-matrix of A of dimension $(A.width, block_size)$ that has the same row indices as C_{sub}, and the sub-matrix of B of dimension $(block_size, A.width)$ that has the same column indices as C_{sub}.</u> In order to fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension $block_size$ as necessary and C_{sub} is computed as the sum of the products of these square matrices. Each of these products is performed by first loading the two corresponding square matrices from global memory to shared memory with one thread loading one element of each matrix, and then by having each thread compute one element of the product. Each thread accumulates the result of each of these products into a register and once done writes the result to global memory.</p> <p><u>By blocking the computation this way, we take advantage of fast shared memory and save a lot of global memory bandwidth since A is only read $(B.width / block_size)$ times from global memory and B is read $(A.height / block_size)$ times.</u></p> <p>The <i>Matrix</i> type from the previous code sample is augmented with a <i>stride</i> field, so that sub-matrices can be efficiently represented with the same type. <code>__device__</code> functions are used to get and set elements and build any sub-matrix from a matrix.</p> <p style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	<pre> // Thread block size #define BLOCK_SIZE 16 // Forward declaration of the matrix multiplication kernel __global__ void MatMulKernel(const Matrix, const Matrix, Matrix); // Matrix multiplication - Host code // Matrix dimensions are assumed to be multiples of BLOCK_SIZE void MatMul(const Matrix A, const Matrix B, Matrix C) { // Load A and B to device memory Matrix d_A; d_A.width = d_A.stride = A.width; d_A.height = A.height; size_t size = A.width * A.height * sizeof(float); cudaMalloc(&d_A.elements, size); cudaMemcpy(d_A.elements, A.elements, size, cudaMemcpyHostToDevice); Matrix d_B; d_B.width = d_B.stride = B.width; d_B.height = B.height; size = B.width * B.height * sizeof(float); cudaMalloc(&d_B.elements, size); cudaMemcpy(d_B.elements, B.elements, size, cudaMemcpyHostToDevice); // Allocate C in device memory Matrix d_C; d_C.width = d_C.stride = C.width; d_C.height = C.height; size = C.width * C.height * sizeof(float); cudaMalloc(&d_C.elements, size); // Invoke kernel dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE); dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y); MatMulKernel<<<dimGrid, dimBlock>>>(d_A, d_B, d_C); // Read C from device memory cudaMemcpy(C.elements, d_C.elements, size, cudaMemcpyDeviceToHost); // Free device memory cudaFree(d_A.elements); cudaFree(d_B.elements); cudaFree(d_C.elements); } </pre>

```

// Matrix multiplication kernel called by MatMul()
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    // Block row and column
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;
    // Each thread block computes one sub-matrix Csub of C
    Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
    // Each thread computes one element of Csub
    // by accumulating results into Cvalue
    float Cvalue = 0;
    // Thread row and column within Csub
    int row = threadIdx.y;
    int col = threadIdx.x;
    // Loop over all the sub-matrices of A and B that are
    // required to compute Csub
    // Multiply each pair of sub-matrices together
    // and accumulate the results
    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {
        // Get sub-matrix Asub of A
        Matrix Asub = GetSubMatrix(A, blockRow, m);
        // Get sub-matrix Bsub of B
        Matrix Bsub = GetSubMatrix(B, m, blockCol);
        // Shared memory used to store Asub and Bsub respectively
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
        // Load Asub and Bsub from device memory to shared memory
        // Each thread loads one element of each sub-matrix
        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);
        // Synchronize to make sure the sub-matrices are loaded
        // before starting the computation
        __syncthreads();
        // Multiply Asub and Bsub together
        for (int e = 0; e < BLOCK_SIZE; ++e)
            Cvalue += As[row][e] * Bs[e][col];
        // Synchronize to make sure that the preceding
        // computation is done before loading two new
        // sub-matrices of A and B in the next iteration
        __syncthreads();
    }
    // Write Csub to device memory
    // Each thread writes one element
    SetElement(Csub, row, col, Cvalue);
}

```


Key Features	Accused Products								
	<p>In addition, cuDNN, the GPU-acceleration library for deep neural networks, implements a cuDNN backend MatMul (matrix multiplication) operation descriptor using “cudaBackendCreateDescriptor.” It “specifies an operation node for matmul to compute the matrix product C [output] by multiplying matrix A and matrix B [inputs]” (e.g., <i>at least one calculation on the first portion of the input data as to generate first output data</i>).</p> <p>9.3.16. CUDNN_BACKEND_OPERATION_MATMUL_DESCRIPTOR</p> <p>Created with <code>cudaBackendCreateDescriptor(CUDNN_BACKEND_OPERATION_MATMUL_DESCRIPTOR, &desc)</code>; the cuDNN backend <code>matmul</code> operation descriptor specifies an operation node for <code>matmul</code> to compute the matrix product C by multiplying matrix A and matrix B, as shown in the following equation: $C = AB$</p> <p>When using the <code>matmul</code> operation, the matrices are expected to be at least rank-2 tensors. The last two dimensions are expected to correspond to either M, K or N. All the preceding dimensions are interpreted as batch dimensions. If there are zero batch dimensions then the requirements are as follows:</p> <p>Table 53. CUDNN_BACKEND_OPERATION_MATMUL_DESCRIPTOR for Zero Batch Dimensions</p> <table><tr><th>Case</th><th>Matrix A</th><th>Matrix B</th><th>Matrix C</th></tr><tr><td>Single matmul</td><td>M x K</td><td>K x N</td><td>M x N</td></tr></table> <p>(See https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-893/api/index.html#CUDNN_BACKEND_OPERATION_MATMUL_DESCRIPTOR (emphasis added).)</p> <p>As another example, the “<code>cuda_adv_infer.so</code>” library for cuDNN includes a directional mode “<code>cudaDirectionMode_t</code>” for unidirectional and bidirectional values for recurrence patterns in recurrent neural network (RNN) iterations. The networks can “iterate[] recurrently from the first input to the last” and “from the first input to the last and separately from the last input to the first.” For the bidirectional mode, “[t]he outputs . . . are concatenated at each iteration giving the output of the layer.”</p> <p>7. cuda_adv_infer.so Library</p> <p>This entity contains all other features and algorithms. This includes RNNs, CTC loss, and multi-head attention. The <code>cuda_adv_infer</code> library depends on <code>cuda_ops_infer</code>.</p> <p style="text-align: center;">* * * * *</p>	Case	Matrix A	Matrix B	Matrix C	Single matmul	M x K	K x N	M x N
Case	Matrix A	Matrix B	Matrix C						
Single matmul	M x K	K x N	M x N						

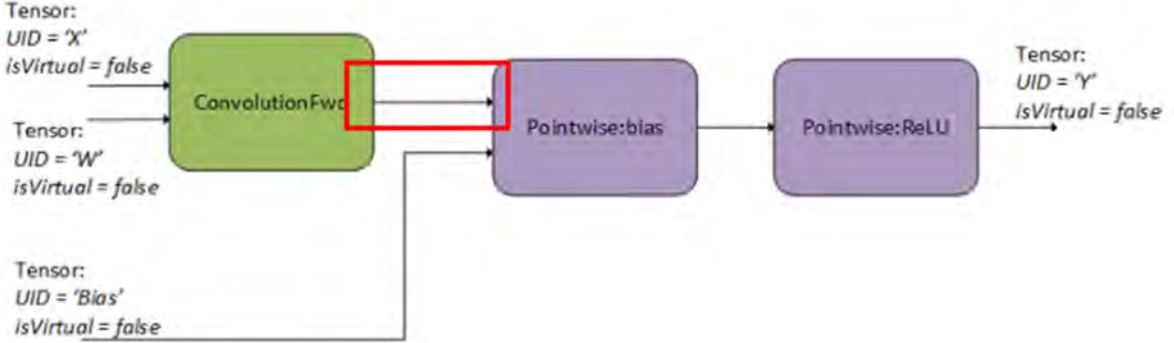
Key Features	Accused Products
	<p><u>7.1.2. Enumeration Types</u> These are the enumeration types in the <code>cuda_adv_infer.so</code> library.</p> <p><u>7.1.2.1. cudnnDirectionMode_t</u> <code>cudnnDirectionMode_t</code> is an enumerated type used to specify the recurrence pattern in the <code>cudnnRNNForwardInference()</code>, <code>cudnnRNNForwardTraining()</code>, <code>cudnnRNNBackwardData()</code> and <code>cudnnRNNBackwardWeights()</code> routines.</p> <p>Values</p> <div style="border: 1px solid red; padding: 5px;"> <p>CUDNN_UNIDIRECTIONAL The network iterates recurrently from the first input to the last.</p> <p>CUDNN_BIDIRECTIONAL Each layer of the network iterates recurrently from the first input to the last and separately from the last input to the first. The outputs of the two are concatenated at each iteration giving the output of the layer.</p> </div> <p>(See https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-893/api/index.html#cudnnDirectionMode_t (emphasis added).)</p>
<p>[16.4] storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory; and</p>	<p>The Accused Products practice a method that includes <i>storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory.</i></p> <p>See [16.2] (<i>transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation</i>), <i>supra</i>, regarding the memory features of the Accused Products with the Maxwell GPU architecture.</p> <p>(See https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf (Maxwell-architecture GPUs (listed, <i>supra</i>), such as the “GM204” GPU (Geforce GTX 980), include “memory controllers” and “DRAM” (dynamic random-access memory) (e.g., <i>storing, by the accelerator controller</i>) (emphasis added).)</p> <p>Exemplary excerpts of CUDA code shown below demonstrate CUDA being used to calculate a square sub-matrix Csub of matrix C using the function MatMul. An exemplary CUDA stream “allocate[s] [matrix] C in device memory.” After Matrices A and B are synchronized and multiplied, the exemplary CUDA stream “[w]rite[s] Csub to device [GPU] memory” (e.g., <i>the first output data into a second partition, referenced by a second pointer, of the accelerator memory</i>).</p>

Key Features	Accused Products
	<pre> // Load A and B to device memory Matrix d_A; d_A.width = d_A.stride = A.width; d_A.height = A.height; size_t size = A.width * A.height * sizeof(float); cudaMalloc(&d_A.elements, size); cudaMemcpy(d_A.elements, A.elements, size, cudaMemcpyHostToDevice); Matrix d_B; d_B.width = d_B.stride = B.width; d_B.height = B.height; size = B.width * B.height * sizeof(float); cudaMalloc(&d_B.elements, size); cudaMemcpy(d_B.elements, B.elements, size, cudaMemcpyHostToDevice); // <u>Allocate C in device memory</u> Matrix d_C; d_C.width = d_C.stride = C.width; d_C.height = C.height; size = C.width * C.height * sizeof(float); cudaMalloc(&d_C.elements, size); ***** // Synchronize to make sure the sub-matrices are loaded // before starting the computation __syncthreads(); // Multiply Asub and Bsub together for (int e = 0; e < BLOCK_SIZE; ++e) Cvalue += As[row][e] * Bs[e][col]; // Synchronize to make sure that the preceding // computation is done before loading two new // sub-matrices of A and B in the next iteration __syncthreads(); } // <u>Write Csub to device memory</u> // Each thread writes one element SetElement(Csub, row, col, Cvalue); } </pre> <p>(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#shared-memory (emphasis added)).</p>

Key Features	Accused Products
	<p>Exemplary CUDA function “cudaMalloc” is used to “allocate weight, work, and reserve space buffer sizes in the GPU memory.” “The work-space buffer is used for temporary storage” and the “content can be discarded or modified after all GPU kernels launched by the corresponding API complete.” The “reserve-space buffer” used to transfer intermediate results is used for transferring “intermediate results” as used in the cuDNN GPU-acceleration library for CUDA (e.g., <i>storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory</i>).</p> <p><u>7.2.15. cudnnGetMultiHeadAttnBuffers()</u></p> <p>This function computes weight, work, and reserve space buffer sizes used by the following functions:</p> <ul style="list-style-type: none"> • <u>cudnnMultiHeadAttnForward()</u> • <u>cudnnMultiHeadAttnBackwardData()</u> • <u>cudnnMultiHeadAttnBackwardWeights()</u> <pre> cudnnStatus_t cudnnGetMultiHeadAttnBuffers(cudnnHandle_t handle, const cudnnAttnDescriptor_t attnDesc, size_t *weightSizeInBytes, size_t *workSpaceSizeInBytes, size_t *reserveSpaceSizeInBytes); </pre> <p>Assigning NULL to the <code>reserveSpaceSizeInBytes</code> argument indicates that the user does not plan to invoke multi-head attention gradient functions: <u>cudnnMultiHeadAttnBackwardData()</u> and <u>cudnnMultiHeadAttnBackwardWeights()</u>. This situation occurs in the inference mode.</p> <p>Note: NULL cannot be assigned to <code>weightSizeInBytes</code> and <code>workSpaceSizeInBytes</code> pointers.</p> <p>The user must allocate weight, work, and reserve space buffer sizes in the GPU memory using <u>cudaMalloc()</u> with the reported buffer sizes. The buffers can be also carved out from a larger chunk of allocated memory but the buffer addresses must be at least 16B aligned.</p> <p>The work-space buffer is used for temporary storage. Its content can be discarded or modified after all GPU kernels launched by the corresponding API complete. The reserve-space buffer is used to transfer intermediate results from <u>cudnnMultiHeadAttnForward()</u> to <u>cudnnMultiHeadAttnBackwardData()</u>, and from <u>cudnnMultiHeadAttnBackwardData()</u> to <u>cudnnMultiHeadAttnBackwardWeights()</u>. The content of the reserve-space buffer cannot be modified until all GPU kernels launched by the above three multi-head attention API functions finish.</p> <p>All multi-head attention weight and bias tensors are stored in a single weight buffer. For speed optimizations, the cuDNN API may change tensor layouts and their relative locations in the weight buffer based on the provided attention parameters. Use the <u>cudnnGetMultiHeadAttnWeights()</u> function to obtain the start address and the shape of each weight or bias tensor.</p> <p>(See https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-893/api/index.html#cudnnGetMultiHeadAttnBuffers/ (emphasis added).)</p>

Key Features	Accused Products
<p>[16.5] swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation.</p>	<p>The Accused Products practice a method that includes <i>swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation.</i></p> <p>For example, the cuDNN GPU-acceleration library of the Accused Products implement operations that “take tensors as input and produce tensors as output” (e.g., <i>first output data becomes an input for a second computational cycle of the numerical simulation</i>).</p> <p>2.2. Tensors and Layouts</p> <p>Whether using the graph API or the legacy API, <u>cuDNN operations take tensors as input and produce tensors as output.</u></p> <p>(See https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-881/developer-guide/index.html#tensors-layouts (emphasis added).)</p> <p>As shown below for a “forward convolution operation” using cuDNN, “backend tensor <i>Tmp0</i> [is] both the output of the convolution operation and the input of the bias operation.” From this, “cuDNN infers that the dataflow runs from the convolution into the bias.” When a neural network output becomes an input for another operation or layer, the first pointer is swapped with the second pointer (e.g., <i>swapping the first pointer with the second pointer at the end of the first computational cycle</i>). A finalized operation graph in which “cuDNN performs the dataflow analysis to establish the dependency relationship between operations and connect the edges” illustrates the output of “ConvolutionFwd” (<i>Tmp0</i>) as the input for “Pointwise:bias” (e.g., <i>such that the first output data becomes an input for a second computational cycle of the numerical simulation</i>).</p>

Key Features	Accused Products
	<p>3.2. Graph API Example with Operation Fusion</p> <p>The following example implements a fusion of convolution, bias, and activation.</p> <p>3.2.1. Creating Operation and Tensor Descriptors to Specify the Graph Dataflow</p> <p>First, create three cuDNN backend operation descriptors.</p> <p>As can be seen in Figure 6, the user specified one forward convolution operation (using <code>CUDNN_BACKEND_OPERATION_CONVOLUTION_FORWARD_DESCRIPTOR</code>), a pointwise operation for the bias addition (using <code>CUDNN_BACKEND_OPERATION_POINTWISE_DESCRIPTOR</code> with mode <code>CUDNN_POINTWISE_ADD</code>), and a pointwise operation for the ReLU activation (using <code>CUDNN_BACKEND_OPERATION_POINTWISE_DESCRIPTOR</code> with mode <code>CUDNN_POINTWISE_RELU_FWD</code>). Refer to the NVIDIA cuDNN Backend API for more details on setting the attributes of these descriptors. For an example of how a forward convolution can be set up, refer to the Setting Up An Operation Graph For A Grouped Convolution use case in the cuDNN backend API.</p> <p>You should also create tensor descriptors for the inputs and outputs of all of the operations in the graph. <u>The graph dataflow is implied by the assignment of tensors (refer to Figure 6), for example, by specifying the backend tensor <i>Tmp0</i> as both the output of the convolution operation and the input of the bias operation, cuDNN infers that the dataflow runs from the convolution into the bias. The same applies to tensor <i>Tmp1</i>. If the user doesn't need the intermediate results <i>Tmp0</i> and <i>Tmp1</i> for any other use, then the user can specify them to be virtual tensors, so the memory I/Os can later be optimized out.</u></p> <p style="text-align: center;">* * * * *</p> <p>Figure 6. A set of operation descriptors the user passes to the operation graph</p> <pre> graph LR X[Tensor: UID = 'X' isVirtual = false] --> Conv[ConvolutionFwd] W[Tensor: UID = 'W' isVirtual = false] --> Conv Conv --> Tmp0_1[Tensor: UID = 'Tmp0' isVirtual = true] Tmp0_1 --> ReLU[Pointwise:ReLU] ReLU --> Y[Tensor: UID = 'Y' isVirtual = false] Tmp0_1 --> Tmp0_2[Tensor: UID = 'Tmp0' isVirtual = true] Bias[Tensor: UID = 'Bias' isVirtual = false] --> BiasOp[Pointwise:bias] Tmp0_2 --> BiasOp BiasOp --> Tmp1[Tensor: UID = 'Tmp1' isVirtual = true] </pre>

Key Features	Accused Products
	<p>3.2.2. Finalizing The Operation Graph</p> <p>Second, the user finalizes the operation graph. As part of finalization, cuDNN performs the dataflow analysis to establish the <u>dependency relationship between operations and connect the edges</u>, as illustrated in the following figure. In this step, cuDNN performs various checks to confirm the validity of the graph.</p> <p>Figure 7. The operation graph after finalization</p>  <p>(See https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn-881/developer-guide/index.html#tensors-layouts (emphasis added); see also https://docs.nvidia.com/deeplearning/cudnn/latest/developer/graph-api.html (“ConvolutionFwd computes the convolution of X with filter data W. In addition, it uses scaling factors α and β to blend this result with the previous output. This graph operation is similar to cudnnConvolutionForward()”) (emphasis added) (e.g., <i>the first output data becomes an input for a second computational cycle of the numerical simulation</i>).)</p> <p>Nvidia has further confirmed that CUDA implements pointer swapping for device (GPU) pointers (e.g., <i>swapping the first pointer with the second pointer at the end of the first computational cycle</i>).</p>

Key Features	Accused Products
	<div data-bbox="594 256 1264 337"> <h3>Swap device Pointers</h3> <p>Home > Accelerated Computing > CUDA > CUDA Programming and Performance</p> </div> <div data-bbox="594 354 1766 922"> <div>  <div> <p>SamWhite</p> <p>Aug 01 '15</p> </div> </div> <p>Hi guys :)</p> <p>I'm really sorry for the very basic question but I cannot find a straight answer on any of the online resources I have checked.</p> <p>I am writing some code in cuda that 'ping pongs' on two sets of data, reading from one and writing to the other.</p> <p>What I need to know is <u>does the following piece of code actually work on DEVICE pointers in cuda</u>. This is memory that has already been allocated on the device. I call the below in a method at the start of each frame.</p> <pre>cData* temp = d_data; d_data= d_dataOld; d_dataOld= temp;</pre> <p>If not can you suggest alternatives to simply changing which pointers I send to the kernels - I prefer to avoid that method in order to keep the code easy to read!</p> <p>Thanks :)</p> <p style="text-align: center;">* * * * *</p> <p>3.7k views</p> </div> <div data-bbox="573 946 1766 1092"> <div>  <div> <p>Robert_Crovella Moderator</p> <p>Aug 01 '15</p> </div> </div> <p>It works. A pointer in C is just a number.</p> <p><u>A DEVICE pointer in cuda is just a C pointer</u>, after all. There's no magic here.</p> </div> <p>(See https://forums.developer.nvidia.com/t/swap-device-pointers/38964 (emphasis added).)</p>

3.2.8.5. Streams

Applications manage the concurrent operations described above through *streams*. A stream is a sequence of commands (possibly issued by different host threads) that execute in order. Different streams, on the other hand, may execute their commands out of order with respect to one another or concurrently; this behavior is not guaranteed and should therefore not be relied upon for correctness (for example, inter-kernel communication is undefined). The commands issued on a stream may execute when all the dependencies of the command are met. The dependencies could be previously launched commands on same stream or dependencies from other streams. The successful completion of synchronize call guarantees that all the commands launched are completed.

3.2.8.5.1. Creation and Destruction of Streams

A stream is defined by creating a stream object and specifying it as the stream parameter to a sequence of kernel launches and host `<->` device memory copies. The following code sample creates two streams and allocates an array `hostPtr` of `float` in page-locked memory.

```
cudaStream_t stream[2];
for (int i = 0; i < 2; ++i)
    cudaStreamCreate(&stream[i]);
float* hostPtr;
cudaMallocHost(&hostPtr, 2 * size);
```

Each of these streams is defined by the following code sample as a sequence of one memory copy from host to device, one kernel launch, and one memory copy from device to host:

```
for (int i = 0; i < 2; ++i) {
    cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i * size,
                    size, cudaMemcpyHostToDevice, stream[i]);
    MyKernel <<<100, 512, 0, stream[i]>>>
    (outputDevPtr + i * size, inputDevPtr + i * size, size);
    cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i * size,
                    size, cudaMemcpyDeviceToHost, stream[i]);
}
```

Each stream copies its portion of input array `hostPtr` to array `inputDevPtr` in device memory, processes `inputDevPtr` on the device by calling `MyKernel()`, and copies the result `outputDevPtr` back to the same portion of `hostPtr`.

Overlapping Behavior describes how the streams overlap in this example depending on the capability of the device. Note that `hostPtr` must point to page-locked host memory for any overlap to occur.

Streams are released by calling `cudaStreamDestroy()`.

Key Features	Accused Products
	(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/#creation-and-destruction-of-streams (emphasis added).)
<p>[17] The method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.</p>	<p>The Accused Products perform <i>the method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.</i></p> <p>See [16.2] (<i>transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation</i>), <i>supra</i>, regarding the memory features of the Accused Products with the Maxwell GPU architecture.</p> <p>(See https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf (Maxwell-architecture GPUs (listed, <i>supra</i>), such as the “GM204” GPU (Geforce GTX 980), include “memory controllers” and “DRAM” (dynamic random-access memory) (e.g., <i>sending, by the accelerator controller</i>) (emphasis added).)</p> <p>For instance, the first main CUDA program execution step is “[c]opy[ing] the input data from host [CPU] memory to device [GPU] memory, also known as host-to-device transfer” (e.g., <i>sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit</i>).</p>

Key Features	Accused Products
	<p>Let me introduce two keywords widely used in <u>CUDA programming model: <i>host</i> and <i>device</i>.</u></p> <p><u>The host is the CPU available in the system.</u> The system memory associated with the CPU is called host memory. <u>The GPU is called a device</u> and GPU memory likewise called device memory.</p> <p>To execute any CUDA program, there are three main steps:</p> <ul style="list-style-type: none"> • <u>Copy the input data from host memory to device memory, also known as host-to-device transfer.</u> • Load the GPU program and execute, caching data on-chip for performance. • Copy the results from device memory to host memory, also called device-to-host transfer. <p>(See https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/ (emphasis added).)</p> <p>Indeed, the Maxwell architecture Tesla M4 GPU is described as “the world’s first accelerator designed for hyperscale servers, enabling customers to keep up with ever-growing amount of data. It’s engineered to accelerate application throughput in a small, low-power design, slashing data center costs by half and deliver up to 5x more power-efficient processing than CPUs for deep learning inference and video workloads.” And the Maxwell architecture Tesla M60 GPU is “designed for single precision GPU compute tasks as well as to accelerate graphics in virtual remote workstation and virtual desktop environments.”</p>

Key Features	Accused Products																						
	<p>The World's First Accelerator Designed for Hyperscale Servers</p> <p>Exploding volumes of user-generated data are redefining what's required for hyperscale data centers. Today's cloud applications harness valuable data to deliver smarter, real-time experiences using modern video and image processing and deep learning techniques. These applications can benefit greatly from GPU acceleration in the data center.</p> <p>The NVIDIA Tesla M4 is the world's first accelerator designed for hyperscale servers, enabling customers to keep up with ever-growing amount of data. It's engineered to accelerate application throughput in a small, low-power design, slashing data center costs by half and deliver up to 5x more power-efficient processing than CPUs for deep learning inference and video workloads.</p>  <p>FEATURES</p> <ul style="list-style-type: none"> NVIDIA GPU Boost™, which delivers up to 2.2 Teraflops of single-precision performance Small, low-power design for hyperscale servers Server qualification to deliver maximum uptime in the data center <p>SPECIFICATIONS</p> <table> <tr> <td>GPU Architecture</td><td>NVIDIA Maxwell™</td></tr> <tr> <td>NVIDIA CUDA® Cores</td><td>1024</td></tr> <tr> <td>Single-Precision Performance</td><td>2.2 Teraflops with NVIDIA GPU Boost</td></tr> <tr> <td>Double-Precision Performance</td><td>.07 Teraflops with NVIDIA GPU Boost</td></tr> <tr> <td>GPU Memory</td><td>4 GB GDDR5</td></tr> <tr> <td>Memory Bandwidth</td><td>88 GB/s</td></tr> <tr> <td>System Interface</td><td>PCIe Gen3</td></tr> <tr> <td>Max Power Consumption</td><td>50W-75W</td></tr> <tr> <td>Thermal Solution</td><td>Passive</td></tr> <tr> <td>Form Factor</td><td>Low Profile</td></tr> <tr> <td>Compute APIs</td><td>NVIDIA CUDA, DirectCompute, OpenCL, OpenACC</td></tr> </table>	GPU Architecture	NVIDIA Maxwell™	NVIDIA CUDA® Cores	1024	Single-Precision Performance	2.2 Teraflops with NVIDIA GPU Boost	Double-Precision Performance	.07 Teraflops with NVIDIA GPU Boost	GPU Memory	4 GB GDDR5	Memory Bandwidth	88 GB/s	System Interface	PCIe Gen3	Max Power Consumption	50W-75W	Thermal Solution	Passive	Form Factor	Low Profile	Compute APIs	NVIDIA CUDA, DirectCompute, OpenCL, OpenACC
GPU Architecture	NVIDIA Maxwell™																						
NVIDIA CUDA® Cores	1024																						
Single-Precision Performance	2.2 Teraflops with NVIDIA GPU Boost																						
Double-Precision Performance	.07 Teraflops with NVIDIA GPU Boost																						
GPU Memory	4 GB GDDR5																						
Memory Bandwidth	88 GB/s																						
System Interface	PCIe Gen3																						
Max Power Consumption	50W-75W																						
Thermal Solution	Passive																						
Form Factor	Low Profile																						
Compute APIs	NVIDIA CUDA, DirectCompute, OpenCL, OpenACC																						

Key Features	Accused Products
	<p data-bbox="569 240 1896 310">(See https://images.nvidia.com/content/tesla/pdf/NVIDIATesla-Print-M4-Datasheet_LR.PDF (emphasis added).)</p> <p data-bbox="569 362 1896 626">The NVIDIA® Tesla® M60 is a dual-slot 10.5 inch PCI Express Gen3 graphics card with two high-end NVIDIA Maxwell™ graphics processing units (GPUs). The Tesla M60 has 16 GB GDDR5 memory (8 GB per GPU) and a 300 W maximum power limit. The board is offered in a 300 W passively cooled variant that requires system airflow to properly operate the card within its thermal limits or in a 240 W actively cooled version. It is designed for single precision GPU compute tasks as well as to accelerate graphics in virtual remote workstation and virtual desktop environments.</p> <p data-bbox="569 678 1896 716">(See https://images.nvidia.com/content/pdf/tesla/tesla-m60-product-brief.pdf (emphasis added).)</p> <p data-bbox="569 751 1896 927">Relatedly, the PowerEdge C4130 with Nvidia Tesla M60 GPU “provides supercomputing agility and performance in an ultra-dense platform purpose-built for high-performance computing (HPC) and virtual desktop infrastructure (VDI) workloads” including “complex research, simulation and visualization problems in medicine, finance, energy exploration and related fields without compromising on versatility or data center space.”</p>

Key Features	Accused Products
	<p>The PowerEdge C4130 provides supercomputing agility and performance in an ultra-dense platform purpose-built for high-performance computing (HPC) and virtual desktop infrastructure (VDI) workloads. Speed through complex research, simulation and visualization problems in medicine, finance, energy exploration and related fields without compromising on versatility or data center space.</p> <p>Deliver peak performance</p> <p>Get results faster with greater precision by combining up to two Intel® Xeon® E5-2600 v4 processors and up to four 300W dual-width PCIe accelerators in each C4130 server. Support for NVIDIA® Tesla® GPUs and Intel Xeon Phi™ coprocessors along with up to 1TB of DDR4 memory gives you ultimate control in matching your server architecture to your specific performance requirements.</p> <p>(See https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-PowerEdge-C4130-Spec-Sheet.pdf/.)</p>
<p>[18] The method as claimed in claim 16, further comprising: partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular</p>	<p>The Accused Products perform <i>the method as claimed in claim 16, further comprising: partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation.</i></p> <p><i>See [16.2] (transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation), supra, regarding the memory features of the Accused Products with the Maxwell GPU architecture.</i></p> <p>(See https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf (Maxwell-architecture GPUs (listed, <i>supra</i>), such as the “GM204” GPU (Geforce GTX 980), include “memory controllers” and</p>

Key Features	Accused Products
<p>computation cycle of the numerical simulation.</p>	<p>“DRAM” (dynamic random-access memory) (e.g., <i>partitioning the accelerator memory</i>) (emphasis added).)</p> <p>As previously discussed, the Accused Products and CUDA and CUDA libraries such as cuDNN partition memory when implementing and performing calculations for accelerated computing.</p> <p>For instance, “shared memory” of the Accused Products is partitioned into “memory modules (banks)” (e.g., <i>partitioning the accelerator memory into the first partition, the second partition, a third partition . . . , and a fourth partition</i>).</p> <div data-bbox="583 609 1411 662" data-label="Section-Header"> <h2>CUDA C++ Best Practices Guide</h2> </div> <div data-bbox="583 706 1774 738" data-label="Text"> <p>The <u>programming guide to using the CUDA Toolkit</u> to obtain the best performance from <u>NVIDIA GPUs</u>.</p> </div> <div data-bbox="1171 763 1285 787" data-label="Text"> <p>* * * * *</p> </div>

Key Features	Accused Products
	<p data-bbox="575 240 856 272">9.2.3. Shared Memory</p> <p data-bbox="575 293 1791 350">Because it is on-chip, shared memory has much higher bandwidth and lower latency than local and global memory - provided there are no bank conflicts between the threads, as detailed in the following section.</p> <p data-bbox="575 380 1050 412">9.2.3.1. Shared Memory and Memory Banks</p> <p data-bbox="575 431 1816 521">To achieve high memory bandwidth for concurrent accesses, shared memory is divided into equally sized memory modules (<u>banks</u>) that can be accessed simultaneously. Therefore, any memory load or store of n addresses that spans n distinct memory banks can be serviced simultaneously, yielding an effective bandwidth that is n times as high as the bandwidth of a single bank.</p> <p data-bbox="575 545 1806 695">However, if multiple addresses of a memory request map to the same memory bank, the accesses are serialized. The hardware splits a memory request that has bank conflicts into as many separate conflict-free requests as necessary, decreasing the effective bandwidth by a factor equal to the number of separate memory requests. The one exception here is when multiple threads in a warp address the same shared memory location, resulting in a broadcast. In this case, multiple broadcasts from different banks are coalesced into a single multicast from the requested shared memory locations to the threads.</p> <p data-bbox="575 719 1797 776">To minimize bank conflicts, it is important to understand how memory addresses map to memory banks and how to optimally schedule memory requests.</p> <p data-bbox="575 800 1801 889">On devices of compute capability 5.x or newer, each bank has a bandwidth of 32 bits every clock cycle, and successive 32-bit words are assigned to successive banks. The warp size is 32 threads and the number of banks is also 32, so bank conflicts can occur between any threads in the warp. See Compute Capability 5.x in the CUDA C++ Programming Guide for further details.</p> <p data-bbox="569 906 1896 979">(See https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#shared-memory/ (emphasis added).)</p> <p data-bbox="569 1016 1896 1377">An example below illustrates an exemplary neural network the Accused Products are designed to accelerate using parallel computations. “Input” (four) and “Output” (eight) neurons are depicted below in a full-connected or linear layer structure in which all of the input neurons depicted in a first layer are connected to all of the output neurons depicted in a second layer. Computations for the neural network are performed using, for example, “NVIDIA Matrix Multiplication.” Examples of inputs and outputs for forward propagation, activation gradient computation, and weight gradient computation (as matrix by matrix multiplications) for GEMMs (General Matrix Multiplications) are shown below. For example, for “(a) forward propagation . . . of a fully-connected layer” (the process of feeding input data through a neural network to generate an output), “K = # of inputs” and “M = # of outputs” and in “N = batch size” with results for “Input Activations,” “Output Activations,” and “Weights” (e.g., <i>a third partition to store</i></p>

Key Features	Accused Products
	<p><i>internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation).</i></p> <ul style="list-style-type: none"> As a rough guideline, choose batch sizes and neuron counts greater than 128 to avoid being limited by memory bandwidth (NVIDIA® A100-SXM4-80GB; this threshold is similar for other A100 and V100 GPUs); see <u>Batch Size</u>. <h2><u>2. Fully-Connected Layer</u></h2> <p><u>Fully-connected layers, also known as linear layers, connect every input neuron to every output neuron and are commonly used in neural networks.</u></p> <p><i>Figure 1. Example of a small fully-connected layer with <u>four input and eight output neurons.</u></i></p>

Key Features	Accused Products
	<div data-bbox="884 245 1652 889" data-label="Diagram"> <p>The diagram illustrates a fully-connected layer within a neural network. It consists of two vertical columns of light blue circular nodes. The left column is labeled 'Input Neurons' and contains five nodes. The right column is labeled 'Output Neurons' and contains six nodes. Every node in the input column is connected to every node in the output column by a thin black line, representing a complete bipartite graph. The entire diagram is enclosed within a red rectangular border.</p> </div> <p data-bbox="575 906 1890 1409">Three parameters define a fully-connected layer: batch size, number of inputs, and number of outputs. <u>Forward propagation, activation gradient computation, and weight gradient computation are directly expressed as matrix-matrix multiplications.</u> How the three parameters map to GEMM dimensions (General Matrix Multiplication, background in the <u>NVIDIA Matrix Multiplication Background User's Guide</u>) varies among frameworks, but the underlying principles are the same. For the purposes of the discussion, we adopt the convention used by PyTorch and Caffe where A contains the weights and B the activations. In TensorFlow, matrices take the opposite roles, but the performance principles are the same.</p>

Key Features	Accused Products			
	Table 1. Mapping of inputs, outputs, and batch size to GEMM parameters M, N, K.			
	Computation Phase	M	N	K
	Forward Propagation	Number of outputs	Batch size	Number of inputs
	Activation Gradient	Number of inputs	Batch size	Number of outputs
	Weight Gradient	Number of inputs	Number of outputs	Batch size
* * * * *				

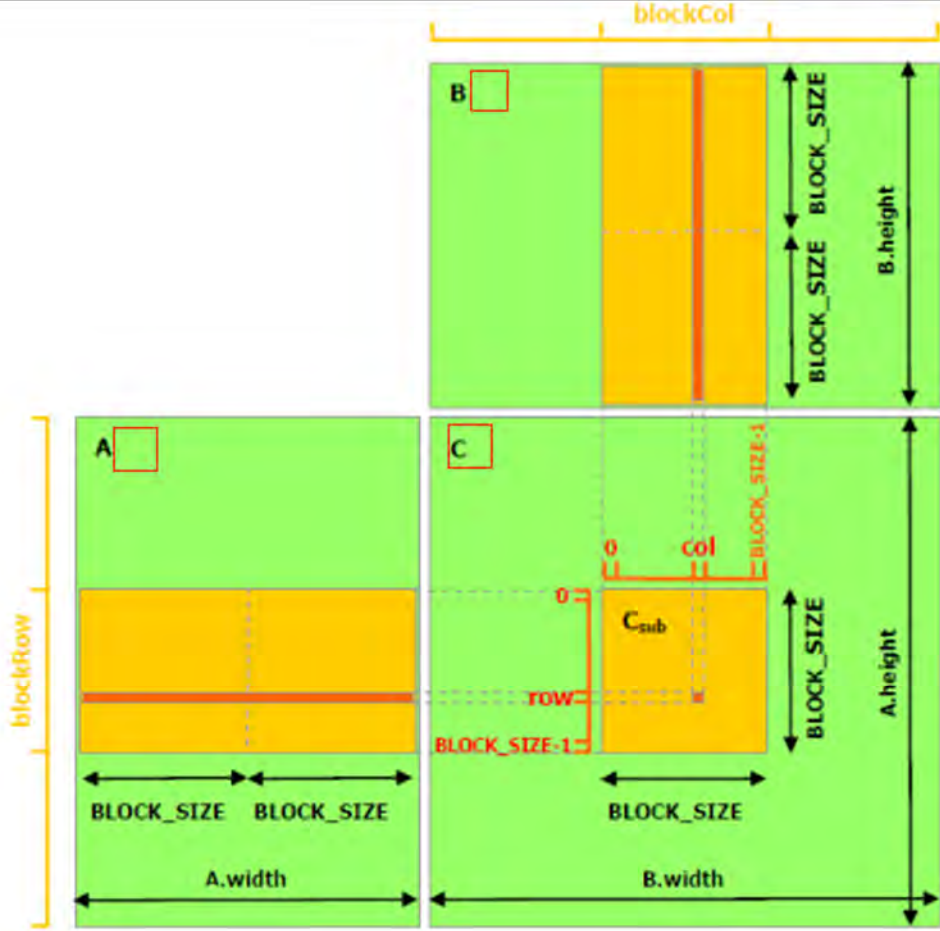
Key Features	Accused Products
	<p data-bbox="583 245 1808 378"><i>Figure 2. Dimensions of equivalent GEMMs for (a) forward propagation, (b) activation gradient, and (c) weight gradient computations of a fully-connected layer.</i></p> <div data-bbox="583 602 1879 1089"> <p>Figure 2 illustrates the dimensions of equivalent GEMMs for three types of computations in a fully-connected layer:</p> <ul style="list-style-type: none"> (a) Forward propagation: A matrix multiplication between $Weights^T$ (dimensions $M \times K$, where M is the number of outputs and K is the number of inputs) and $Input\ Activations$ (dimensions $K \times N$, where N is the batch size). The result is $Output\ Activations$ (dimensions $M \times N$). (b) Activation gradient: A matrix multiplication between $Gradients\ (Input)$ (dimensions $M \times K$, where M is the number of inputs and K is the number of outputs) and $Gradients\ (Output)$ (dimensions $K \times N$, where N is the batch size). The result is $Gradients\ (Weights)$ (dimensions $M \times N$). (c) Weight gradient: A matrix multiplication between $Input\ Activations$ (dimensions $M \times K$, where M is the number of inputs and K is the batch size) and $Gradient^T\ (Output)$ (dimensions $K \times N$, where N is the number of outputs). The result is $Gradient\ (Weights)$ (dimensions $M \times N$). </div> <p data-bbox="569 1105 1560 1175">(See https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#performance (emphasis added).)</p> <p data-bbox="569 1214 1896 1391">As previously discussed, CUDA code can be implemented for “matrix multiplication that does take advantage of shared memory.” As shown, matrix multiplication function “MatMul” calculates “square sub-matrix Csub of C” from sub-matrix A and sub-matrix B. By implementing the matrix computation in this manner, CUDA “take[s] advantage of fast shared memory and save[s] a lot of global memory bandwidth.” Example code shown below demonstrates “Load A and B to device [GPU] memory,”</p>

Key Features	Accused Products
	<p>“Invoke kernel,” “Matrix multiplication kernel called by MatMul(),” and “Write [square sub-matrix] Csub to device [GPU] memory.” Integers (“int”) are defined for “width,” “height,” and “stride” as well as floating-point numbers (“float*”) for “elements.” In this example, sub-matrix A is of dimension “A.width, block_size” (“d_A.width = d_A.stride = A.width; d_A.height = A.height;” “size_t size = A.width * A.height * sizeof(float);”), sub-matrix B is of dimension “block_size, A.width” (“d_B.width = d_B.stride = B.width; d_B.height = B.height;” “size = B.width * B.height * sizeof(float);”), and sub-matrix Csub of C “is equal to the product of [the] two rectangular matrices” with CUDA code “[w]rit[ing] Csub to device memory” and each CUDA threat writing an element (“SetElement(Csub, row, col, Cvalue)”) (e.g., <i>partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation</i>).</p> <p>CUDA C++ Programming Guide</p> <p><u>The programming guide to the CUDA model and interface.</u></p> <p>* * * * *</p>

Key Features	Accused Products
	<p>The following code sample is an implementation of matrix multiplication that does take advantage of shared memory. In this implementation, <u>each thread block is responsible for computing one square sub-matrix C_{sub} of C and each thread within the block is responsible for computing one element of C_{sub}.</u> As illustrated in Figure 9, <u>C_{sub} is equal to the product of two rectangular matrices: the sub-matrix of A of dimension $(A.width, block_size)$ that has the same row indices as C_{sub}, and the sub-matrix of B of dimension $(block_size, A.width)$ that has the same column indices as C_{sub}.</u> In order to fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension $block_size$ as necessary and C_{sub} is computed as the sum of the products of these square matrices. Each of these products is performed by first loading the two corresponding square matrices from global memory to shared memory with one thread loading one element of each matrix, and then by having each thread compute one element of the product. Each thread accumulates the result of each of these products into a register and once done writes the result to global memory.</p> <p><u>By blocking the computation this way, we take advantage of fast shared memory and save a lot of global memory bandwidth since A is only read $(B.width / block_size)$ times from global memory and B is read $(A.height / block_size)$ times.</u></p> <p>The <i>Matrix</i> type from the previous code sample is augmented with a <i>stride</i> field, so that sub-matrices can be efficiently represented with the same type. <code>__device__</code> functions are used to get and set elements and build any sub-matrix from a matrix.</p> <p style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	<pre> // Thread block size #define BLOCK_SIZE 16 // Forward declaration of the matrix multiplication kernel __global__ void MatMulKernel(const Matrix, const Matrix, Matrix); // Matrix multiplication - Host code // Matrix dimensions are assumed to be multiples of BLOCK_SIZE void MatMul(const Matrix A, const Matrix B, Matrix C) { // Load A and B to device memory Matrix d_A; d_A.width = d_A.stride = A.width; d_A.height = A.height; size_t size = A.width * A.height * sizeof(float); cudaMalloc(&d_A.elements, size); cudaMemcpy(d_A.elements, A.elements, size, cudaMemcpyHostToDevice); Matrix d_B; d_B.width = d_B.stride = B.width; d_B.height = B.height; size = B.width * B.height * sizeof(float); cudaMalloc(&d_B.elements, size); cudaMemcpy(d_B.elements, B.elements, size, cudaMemcpyHostToDevice); // Allocate C in device memory Matrix d_C; d_C.width = d_C.stride = C.width; d_C.height = C.height; size = C.width * C.height * sizeof(float); cudaMalloc(&d_C.elements, size); // Invoke kernel dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE); dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y); MatMulKernel<<<dimGrid, dimBlock>>>(d_A, d_B, d_C); // Read C from device memory cudaMemcpy(C.elements, d_C.elements, size, cudaMemcpyDeviceToHost); // Free device memory cudaFree(d_A.elements); cudaFree(d_B.elements); cudaFree(d_C.elements); } </pre> <p style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	<pre> // Synchronize to make sure the sub-matrices are loaded // before starting the computation __syncthreads(); // Multiply Asub and Bsub together for (int e = 0; e < BLOCK_SIZE; ++e) Cvalue += As[row][e] * Bs[e][col]; // Synchronize to make sure that the preceding // computation is done before loading two new // sub-matrices of A and B in the next iteration __syncthreads(); } // Write Csub to device memory // Each thread writes one element SetElement(Csub, row, col, Cvalue); } </pre> <p style="text-align: center;">* * * * *</p>

Key Features	Accused Products
	 <p data-bbox="588 1209 1228 1242">Figure 9: <i>Matrix Multiplication with Shared Memory</i></p> <p data-bbox="567 1258 1890 1331">(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#shared-memory (emphasis added).)</p>

Key Features	Accused Products
<p>[19] The method of claim 16, further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.</p>	<p>The Accused Products perform <i>the method of claim 16 further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.</i></p> <p><i>See [16.2] (transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation), supra, regarding the memory features of the Accused Products with the Maxwell GPU architecture.</i></p> <p><i>See [16.4] (“storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory”), supra.</i></p> <p><i>See [16.5] (“swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation”), supra.</i></p> <p>(<i>See https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf (Maxwell-architecture GPUs (listed, <i>supra</i>), such as the “GM204” GPU (Geforce GTX 980), include “memory controllers” and “DRAM” (dynamic random-access memory) (e.g., <i>transferring, by the accelerator controller</i>) (emphasis added).</i>)</p> <p>In addition, CUDA enables asynchronous transfer of data during computations. Exemplary CUDA memory management function “cudaMemcpyAsync” “[c]opies count bytes [data] from the memory area pointed to by src [source memory address pointer] to the memory area pointed to by dst [destination memory address pointer], where kind [type of transfer] specifies the direction of the copy.” Destinations includes “cudaMemcpyHostToDevice [CPU to device GPU], cudaMemcpyDeviceToHost [GPU to CPU] (e.g., <i>the first output data to the main memory</i>), cudaMemcpyDeviceToDevice [GPU to GPU]. Because the function “cudaMemcpyAsync() is asynchronous with respect to the host, [] the call may return before the copy is complete. The copy can optionally be associated to a stream [identified stream] by passing a non-zero stream argument” (e.g., <i>the first output data to the main memory during the second computational cycle</i>).</p>

Key Features	Accused Products
	<pre data-bbox="577 240 1885 297">__host__ __device__ cudaError_t cudaMemcpyAsync(void* dst, const void* src, size_t count, cudaMemcpyKind kind, cudaStream_t stream = 0)</pre> <p data-bbox="598 310 945 334"><u>Copies data between host and device.</u></p> <div data-bbox="588 358 963 711" style="border: 1px solid red; padding: 5px;"> <p>Parameters</p> <p><code>dst</code> - Destination memory address</p> <p><code>src</code> - Source memory address</p> <p><code>count</code> - Size in bytes to copy</p> <p><code>kind</code> - Type of transfer</p> <p><code>stream</code> - Stream identifier</p> </div> <p>Returns <code>cudaSuccess</code>, <code>cudaErrorInvalidValue</code>, <code>cudaErrorInvalidMemcpyDirection</code></p> <p>Description Copies <code>count</code> bytes from the memory area pointed to by <code>src</code> to the memory area pointed to by <code>dst</code>, where <code>kind</code> specifies the direction of the copy, and must be one of <code>cudaMemcpyHostToHost</code>, <code>cudaMemcpyHostToDevice</code>, <code>cudaMemcpyDeviceToHost</code>, <code>cudaMemcpyDeviceToDevice</code>, or <code>cudaMemcpyDefault</code>. Passing <code>cudaMemcpyDefault</code> is recommended, in which case the type of transfer is inferred from the pointer values. However, <code>cudaMemcpyDefault</code> is only allowed on systems that support unified virtual addressing.</p> <p>The memory areas may not overlap. Calling <code>cudaMemcpyAsync()</code> with <code>dst</code> and <code>src</code> pointers that do not match the direction of the copy results in an undefined behavior.</p> <p><code>cudaMemcpyAsync()</code> is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero <code>stream</code> argument. If <code>kind</code> is <code>cudaMemcpyHostToDevice</code> or <code>cudaMemcpyDeviceToHost</code> and the <code>stream</code> is non-zero, the copy may overlap with operations in other streams.</p> <p>The device version of this function only handles device to device copies and cannot be given local or shared pointers. (See https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html (emphasis added).)</p> <p>As previously stated, the “CUDA programming model” includes programmatic functions, primitives, and executable libraries for CPUs (host) and GPUs (device) that are used by the Accused Products to perform numerical simulations. For example, after the “host-to-device transfer” (CPU to GPU) first main step and “[l]oad[ing] the GPU program and execut[ing]” and “caching data on-chip for performance” for</p>

Key Features	Accused Products
	<p>the second main step, the third main step of a CUDA program is “[c]opy[ing] the results from device [GPU] memory to host [CPU] memory, also known as device-to-host transfer” (GPU to CPU) (e.g., <i>the first output data to the main memory</i>).</p> <p>Let me introduce two keywords widely used in CUDA programming model: <i>host</i> and <i>device</i>.</p> <p>The host is the CPU available in the system. The system memory associated with the CPU is called host memory. The GPU is called a device and GPU memory likewise called device memory.</p> <p>To execute any CUDA program, there are three main steps:</p> <ul style="list-style-type: none"> • Copy the input data from host memory to device memory, also known as host-to-device transfer. • Load the GPU program and execute, caching data on-chip for performance. • <u>Copy the results from device memory to host memory, also called device-to-host transfer.</u> <p>(See https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/ (emphasis added).)</p> <p>As a further example, Nvidia CUDA programming model and extensions to CUDA enable concurrent execution, queuing by asynchronous calls, and dynamic parallelism of device operations. For instance, “[c]oncurrent host execution” through CUDA “is facilitated through asynchronous library functions that return control to the host thread before the device completes the requested task. Using asynchronous calls, many device operations can be queued up together to be executed by the CUDA driver when appropriate device resources are available.” Concurrent operations include “[c]omputations on the device [GPU]” and “host [CPU]” and “[m]emory transfers” between both and “within the memory of a given device” and “among devices” (e.g., <i>transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle</i>). Asynchronous CUDA operations include</p>

Key Features	Accused Products
	<p>“Kernel launches,” “Memory copies performed by functions that are suffixed with Async,” and “Memory set function calls.”</p> <p>3.2.8. Asynchronous Concurrent Execution</p> <p>CUDA exposes the following operations as independent tasks that can operate concurrently with one another:</p> <ul style="list-style-type: none"> ➤ Computation on the host; ➤ Computation on the device; ➤ Memory transfers from the host to the device; ➤ Memory transfers from the device to the host; ➤ Memory transfers within the memory of a given device; ➤ Memory transfers among devices. <p>The level of concurrency achieved between these operations will depend on the feature set and compute capability of the device as described below.</p> <p>3.2.8.1. Concurrent Execution between Host and Device</p> <p>Concurrent host execution is facilitated through asynchronous library functions that return control to the host thread before the device completes the requested task. <u>Using asynchronous calls, many device operations can be queued up together to be executed by the CUDA driver when appropriate device resources are available. This relieves the host thread of much of the responsibility to manage the device, leaving it free for other tasks.</u> The following device operations are asynchronous with respect to the host:</p> <ul style="list-style-type: none"> ➤ Kernel launches; ➤ Memory copies within a single device’s memory; ➤ Memory copies from host to device of a memory block of 64 KB or less; ➤ Memory copies performed by functions that are suffixed with <code>Async</code>; ➤ Memory set function calls. <p>(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html?highlight=cudaMemcpy#asynchronous-concurrent-execution (emphasis added).)</p>

Key Features	Accused Products
	<p>Asynchronous execution</p> <p>Each new generation of NVIDIA GPUs includes numerous architectural enhancements to improve performance, programmability, power efficiency, GPU utilization, and many other factors. <u>Recent NVIDIA GPU generations have included asynchronous execution capabilities to enable more overlap of data movement, computation, and synchronization.</u></p> <p>(See https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth (emphasis added).)</p> <p><u>The ability to create work directly from the GPU can reduce the need to transfer execution control and data between host and device, as launch configuration decisions can now be made at runtime by threads executing on the device.</u> Additionally, data-dependent parallel work can be generated inline within a kernel at run-time, taking advantage of the GPU's hardware schedulers and load balancers dynamically and adapting in response to data-driven decisions or workloads. Algorithms and programming patterns that had previously required modifications to eliminate recursion, irregular loop structure, or other constructs that do not fit a flat, single-level of parallelism may more transparently be expressed.</p> <p>(See https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism.)</p>