TEXT ONLY

Creating the First Confidential GPUs

GOBIKRISHNA DHANUSKODI, SUDESHNA GUHA, VIDHYA KRISHNAN, ARUNA MANJUNATHA, ROB NERTNEY, MICHAEL O'CONNOR, AND PHIL ROGERS

THE TEAM AT NVIDIA BRINGS CONFIDENTIALITY AND INTEGRITY TO USER CODE AND DATA FOR ACCELERATED COMPUTING.

Today's datacenter GPU has a long and storied 3D graphics heritage. In the 1990s, graphics chips for PCs and consoles had fixed pipelines for geometry, rasterization, and pixels using integer and fixed-point arithmetic. In 1999, NVIDIA invented the modern GPU, which put a set of programmable cores at the heart of the chip, enabling rich 3D scene generation with great efficiency. It did not take long for developers and researchers to realize "I could run compute on those parallel cores, and it would be blazing fast." In 2004, Ian Buck created Brook at Stanford, the first compute library for GPUs, and in 2006, NVIDIA created CUDA, which is the gold standard for accelerated computing on GPUs today.

HOW THE IDEA FOR A CONFIDENTIAL GPU TOOK SHAPE

In addition to running 3D graphics and compute, GPUs also run video workloads, including the ability to play back protected content such as Hollywood movies. To protect such content, NVIDIA GPUs include hardware and firmware to secure the area of GPU memory, which holds the decrypted and decoded output frames. This feature is referred to as VPR (video protected region). When an area of GPU memory is set up as VPR, with the exception of a secured display engine that can read from VPR and write to HDMI or DisplayPort channels, any engine that reads from that region will fault if it attempts to write outside of VPR. When CC (confidential computing) started to be discussed, a few of us at NVIDIA started brainstorming about the question, "Can we leverage VPR, or a similar approach, to do confidential compute?" We realized that NVIDIA's Ampere series of GPUs provided the building blocks for a partial confidential compute mode. New firmware could enable an enclave in GPU memory for protected compute, where:

- Only the SEC2 secure microcontroller can read from the enclave and write outside; and when it writes outside, it will first encrypt the data.
- All other engines would fault if they tried to write outside the enclave.

CC requires both confidentiality and integrity for data and code. *Confidentiality* means that data and code cannot be read by an attacker. *Integrity* means that an attacker cannot modify the execution and, for example, cause wrong answers to be generated. The leveraged Ampere approach could provide confidentiality for data but not for code, and it could protect integrity for neither code nor data. This approach was called APM (Ampere Protected Memory) to prevent confusion with full CC capabilities.

We built a POC (proof of concept) for APM and partnered with Microsoft to enable APM in an Azure Private Preview, asking users to try it and provide feedback.

The next step was enabling full CC capability for the Hopper H100 GPUs. It was late in the H100 hardware development phase when we requested the necessary CC features, but all the teams at NVIDIA pulled together to find a way.

H100: THE FIRST CONFIDENTIAL GPU

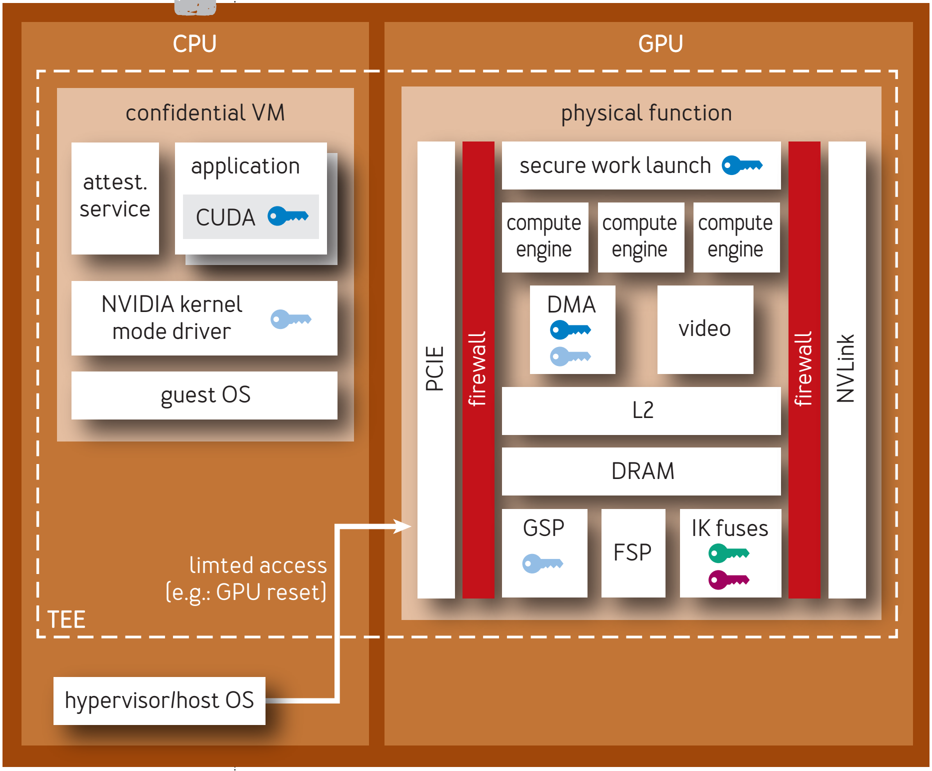
The GPU confidential compute solution relies on a CVM (confidential virtual machine) TEE (trusted execution environment) on the CPU, enabled by SEV-SNP on AMD CPUs or by TDX 1.x on Intel CPUs. Figure 1 shows the high-level architecture of the GPU CC solution.

GPU device memory is logically partitioned into protected and unprotected memory regions. The GPU CPR (compute protected region) of memory is secured so that the GPU can process data at full speed in its HBM (high-bandwidth memory). Additional details on how this is accomplished are shared later. There is no restriction on unprotected GPU memory access from outside the GPU.

When the Hopper GPU boots in confidential mode, it blocks ingress and egress for the CPR of GPU memory. The PCIe (Peripheral Component Interconnect Express) firewall blocks access by the CPU to most registers and all the GPU CPR memory, and the NVIDIA NVLink firewall blocks access by NVLink peer GPUs to GPU CPR memory.

Additionally, hardware engines that operate in CC mode have protections to ensure they cannot write outside compute protected memory unless they have hardware

FIGURE 1: **A TRUSTED EXECUTION ENVIRONMENT**



enforcement for encryption in this mode. This approach prevents engines from leaking data outside protected memory.

The DMA (direct memory access) engines are the only user-mode accessible engines that are enabled to read or write outside of CPR. DMA hardware ensures that data written outside the CPR is pre-encrypted by hardware, which ensures that no data leak is possible. The DMA engine in the H100 GPU supports AES GCM 256 encryption for this purpose, and this engine is used to transfer data between CPU and GPU in both directions.

CC protects data that is in use by performing a computation in a hardware-based, attested TEE (refer to the Confidential Computing Consortium definition). The NVIDIA H100 GPU meets this definition because its TEE is anchored in an on-die hardware RoT (root of trust), and when it boots in CC-On mode, the GPU enables hardware protections for providing confidentiality and integrity of code and data.

1. A chain of trust is established through the GPU boot sequence, with secure and measured boot.
2. An SPDM (Security Protocol and Data Model) session is used to securely connect to the driver in a CPU TEE.
3. An attestation report is generated that provides a cryptographically signed set of measurements.

Users in the CC environment can check the attestation report and proceed only if the report is valid and correct.

The firmware components that run on the GPU are within the TCB (trusted computing base) in CC mode. Only NVIDIA-signed and -attested firmware components are allowed to run in CC mode.

The NVIDIA driver in the CVM establishes a secure channel with the GPU hardware TEE to transfer data, initiate computation, and retrieve results. To communicate with the hardware, unique encryption keys are used for each guest driver component.

A new hardware feature was developed to create a limited view of the GPU registers that can be accessed using PCIe BAR0 (base address register 0). Since the host or hypervisor is not trusted in CC mode, any register that compromises the security of the GPU in CC mode (compromising integrity or confidentiality of the guest) must be protected. This new feature is referred to as the *BARO decoupler*, which allows access to a limited register space to manage the GPU while protecting the majority of the register space from the host and hypervisor.

To protect against side-channel attacks, hardware enforces that all GPU performance counters are disabled when the GPU is operating in CC mode. A new mode, called CC DevTools, supports the performance debugging of applications in CC mode. The CC DevTools mode shows in the attestation report when enabled.

Without CC enabled, the hypervisor has full access to system memory and GPU memory. *With* CC *enabled*, the hypervisor is blocked from accessing the Confidential VM in system memory and blocked from reading GPU memory, as shown in figure 2.

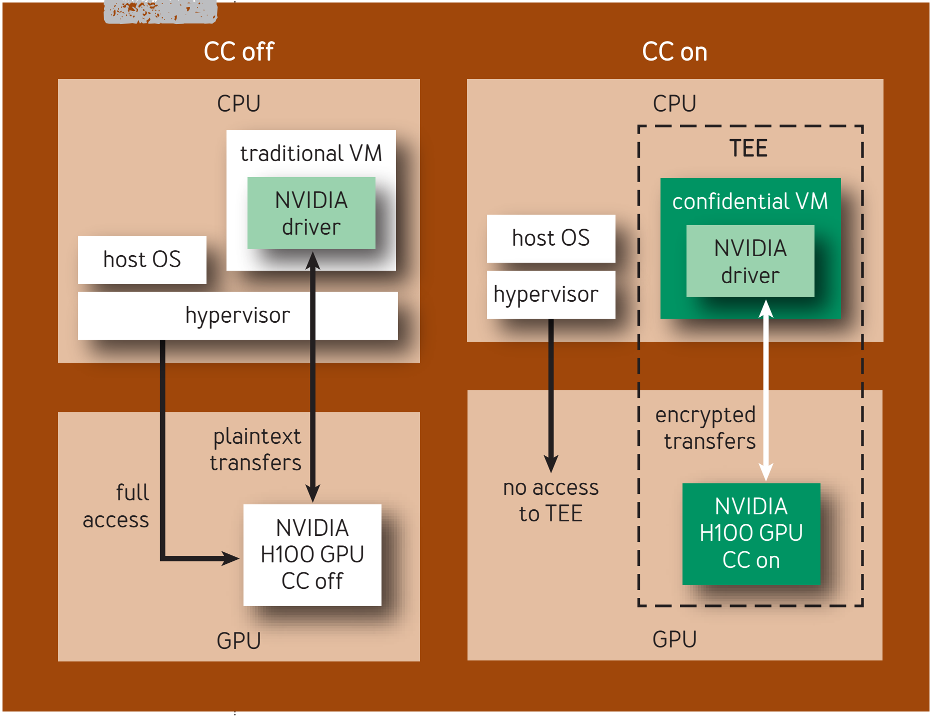
STARTING THE GPU IN CC MODE

The H100 GPU supports these operational modes:

- CC enabled (CC = on)
- CC disabled (CC = off)
- CC devtools (CC = devtools)

To make provisioning more secure, the GPU CC modes are designed to be persistent across PF-FLRs (physical function function-level-resets). GPU CC mode selection is accomplished using an H100 GPU CC control bit in the GPU EEPROM (electrically erasable programmable read-only memory) that can be set/unset by an in-band tool such as gpu_cc_tool.py or through an OOB (out-of-band) API. For

FIGURE 2: **PROTECTING THE GPU IN CONFIDENTIAL COMPUTING MODE**

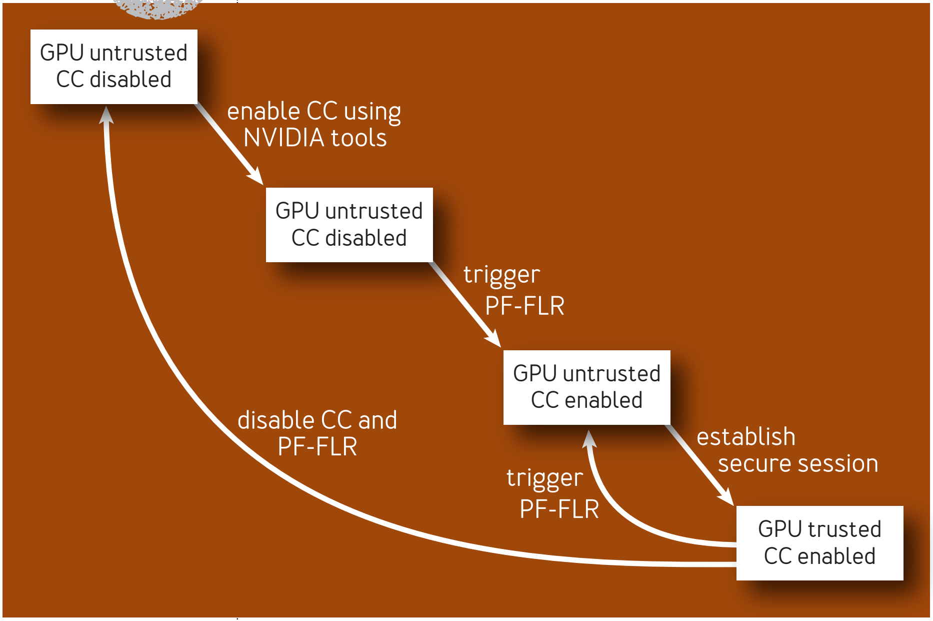


updates to this bit to take effect, a PF-FLR is required that will scrub memory and ensure all the states in registers and SRAMs (static random access memory) are correctly reset before the GPU is handed to the next tenant.

Figure 3 shows the GPU state transitions for enabling CC.

A trusted VM with an AMD SEV-SNP CPU or Intel TDX-enabled CPU is required to validate a GPU in the VM before using the GPU for confidential workloads. To validate that

FIGURE 3: **GPU STATE TRANSITIONS ENABLING CONFIDENTIAL COMPUTING**



a GPU is capable and ready to run a CC workload, these steps must be followed:

1. Authenticate the GPU to be a legitimate NVIDIA GPU that supports CC.
2. Ensure that the GPU is not revoked for CC.
3. Verify the GPU attestation report.

Authentication of the GPU uses the PKI (public-key infrastructure) method. Every NVIDIA H100 GPU carries a unique, per-device ECC (elliptic curve cryptography) keypair and its corresponding public certificate. NVIDIA hosts an OCSP (Online Certificate Status Protocol) service that allows users to check the validity of the certificate and the GPU revocation status for CC.

The GPU driver initiates a key-exchange sequence to establish a secure session with the GPU and uses SPDM messages to authenticate, attest, and perform key exchange with GPUs. Users must query the attestation report and certificate to attest the GPU, and after a successful attestation, toggle the GPU ready state to ON to allow CUDA programs to run on the GPU in CC mode.

ATTESTING THE GPU

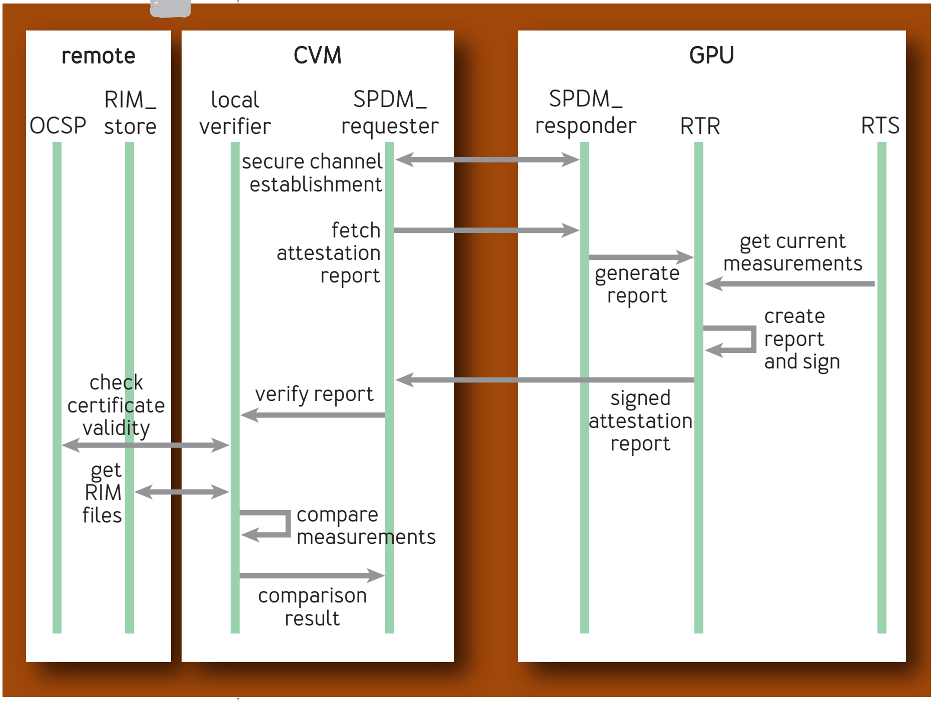
For a GPU to be included in the trust boundary of a CVM, it must be authenticated to prove legitimacy, verified to ensure it is not revoked, and requested to provide evidence of it being in a good known state. Evidence is provided in measurements, and a measurement is a one-way hash of GPU states that are critical for its security. An attestation report is evidence that is signed by the RoT of the device under evaluation. Signing ensures that measurements cannot be altered and eliminates chain-of-custody concerns. Fetching an attestation report using an established secure communication channel eliminates device-spoofing attacks.

After fetching the report, the CVM (or interested party) must validate the authenticity of the evidence and evaluate the report to judge whether the GPU is in a good known state. Evaluating a report requires a golden set of measurements called RIM (reference integrity manifest), which is generated offline by NVIDIA and released with every driver and VBIOS update. The process of comparing the measurements from attestation report with RIM is an *attestation verification*, and the entity performing this process is called a *verifier*. The verifier can be local, built into the CVM; or remote, hosted by the device manufacturer or a trusted third party. The CVM (or interested party) must authenticate and confirm the legitimacy of the verifier before trusting its results. Figure 4 shows a high-level flow of the sequence.

The sequence in figure 4 introduces two new terms:

- RTR (root of trust for reporting). Responsible for fetching the stored measurements, creating a report, and signing them with an attestation key.

FIGURE 4: **ATTESTING THE GPU WITH LOCAL VERIFIER**

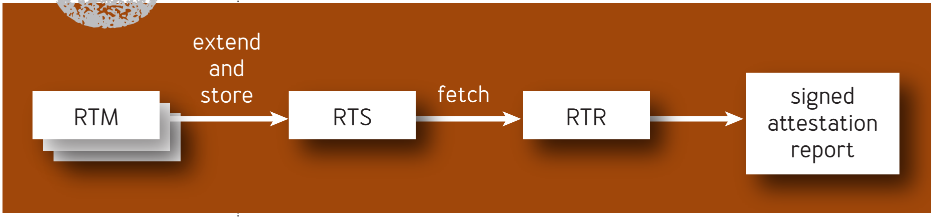


- RTS (root of trust for storage). Secure storage that tracks the measurements collected so far.

Another entity, RTM (root of trust for measurements), shown in figure 5, is responsible for measuring the selected states and saves the measurement in RTS. The NVIDIA GPU has one RTR implemented in the firmware, multiple RTMs, and one hardware-based RTS with storage for up to 64 independent measurements. RTS hardware supports measurement extensions to prevent overwriting and allows for tracking its evolution. Each slot has an RTM owner and stores a measurement that was calculated with one or more states that are related to each other and evolve in an orderly manner.

Determining the correct states to measure in a GPU is a challenging problem. Ideally, measuring all registers, video memory, and SRAMs in the GPU would provide a complete indication of GPU state, but that is impractical because of the volume of states and complexity in generating golden values for comparison. To overcome this challenge and still measure the current state of the GPU with reasonable accuracy, the selected approach is to measure select high-value registers and prove that the GPU configuration

FIGURE 5: **MEASURING THE STATES**



for CC=On has been completed as expected. In figure 6, firewalls in the form of registers, select fuses, debug registers, and all microcodes (μcodes) are measured.

Security events, error triggers, and user policies that impact the security posture of the device are also measured and logged. These policies cannot be directly compared with RIM but can be used by CVM to confirm that the intended actions have taken place. Because VBIOS and GPU drivers are released independently, each has its own RIM and the verifier requires both RIMs for verification, as shown in figure 7 and table 1.

Verifiers play a critical role in setting up a GPU to be included in a CVM trust boundary that might assist relying parties in the decision-making process. There are two classes of verifiers based on where they run: the local

FIGURE 6: **MEASURING THE FIREWALLS**

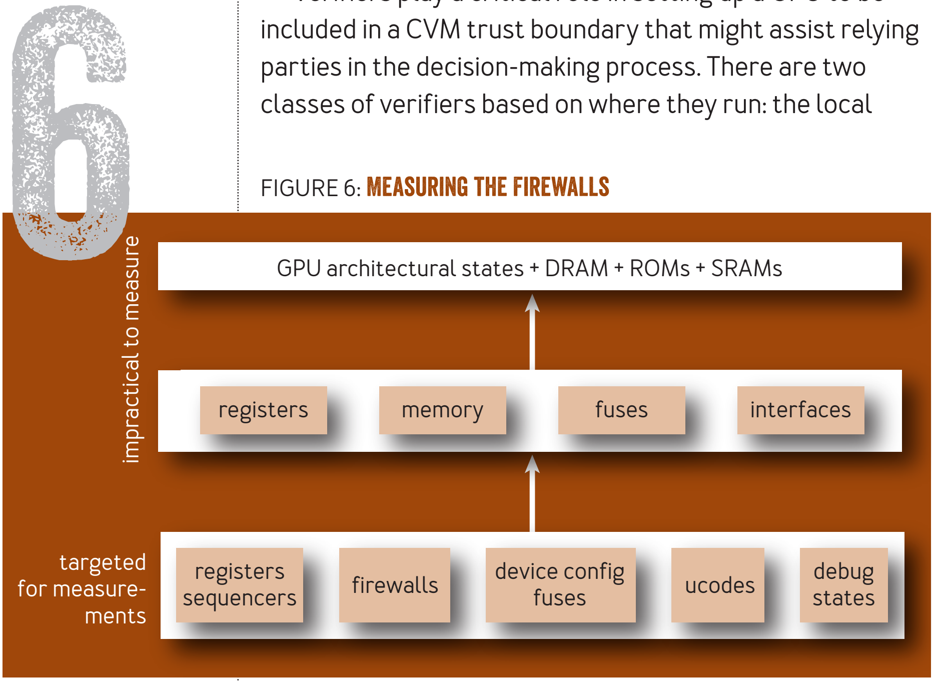


FIGURE 7: **GENERATING THE VERIFICATION RESULT**

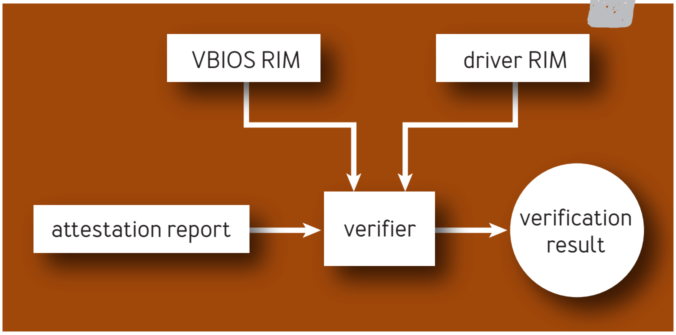


TABLE 1: **RIMS REQUIRED TO VERIFY THE GPU ATTESTATION REPORT**

	VBIOS RIM	**DRIVER RIM**
CONTENTS (MEASUREMENT GROUPS)	• Firmware/VBIOS. • Static hardware configurations critical for Secure Boot • Hardware initialization states that are critical for Secure Boot.	• Static hardware configurations that are critical for CC. • Hardware initialization states that are critical for CC. • Driver loaded ucodes. • Runtime/dynamic states.
RELEASE CADENCE	• With every VBIOS release for the H100 GPU	• With every KMD Driver release for the H100 GPU
REVOCATION	• Using the RIM certificate chain. • The Verifier can also maintain or fetch a list of allowed VBIOS versions and compare the list to the version in RIM.	• Using the RIM certificate chain. • The Verifier can also maintain or fetch a list of allowed driver versions and compare the list with the version in RIM.

verifier, which runs as a dedicated process in CVM; and the remote verifier, which is hosted by a trusted third party.

The local verifier is a stand-alone tool that is available from NVIDIA and acts as a verifier and the relying party.

The local verifier comes with a default policy that allows applications to use the GPU only *after* a successful attestation verification. The local verifier is open sourced, downloadable by the VMI (virtual machine image) creator and can be launched as part of the CVM initialization sequence. This tool is implicitly trusted by CVM to play this role. The local verifier requires these remote services hosted by NVIDIA:

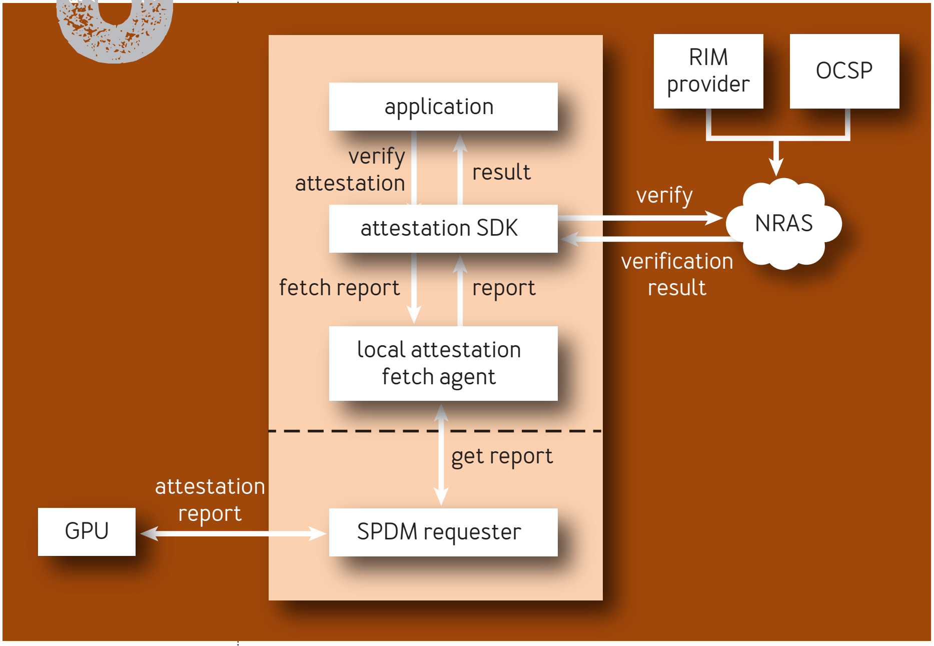
- NVIDIA OCSP service. Validates certificate chains of GPU, attestation, and RIM files.
- NVIDIA RIM provider service. A remote service that hosts RIM files for all drivers and VBIOS releases. The verifier uses unique identifiers from the attestation report to fetch appropriate RIM files.

Although the local verifier enables fast and simple adoption of CC, it has certain challenges that can hinder longer-term usage:

- the local verifier is not scalable as the portfolio of GPUs supporting CC expands.
- the local verifier must be implicitly trusted by the CVM.

The remote verifier addresses these concerns by hosting a verification service on a remote server and allowing the relying party to authenticate the hosted service before delegating report verification. NVIDIA has launched such a service, called NRAS (NVIDIA Remote Attestation Service), which currently supports GPU attestation and may be extended in the future to cover additional NVIDIA products. In addition to NRAS, NVIDIA is introducing an NVIDIA Attestation SDK to integrate the NRAS flow into applications, as shown in figure 8.

FIGURE 8: NVIDIA ATTESTATION SDK



RUNNING THE GPU IN CONFIDENTIAL COMPUTING MODE

After a CVM with the H100 has been correctly configured, booted, and attested, users can start securely processing data on their H100 GPUs. We worked to ensure as much of a *lift-and-shift* style of coding as possible. The goal is to have the existing code and kernels from users work without changes when H100 CC modes are enabled.

By default, devices are blocked from interacting with the CVM and cannot directly access CVM memory. The driver enables H100 to securely communicate with the CVM in CC mode.

A CC-capable CPU isolates the CVM by configuring the MMU (memory management unit) to isolate pages of memory so that only the associated VM can access it. This isolation does not simply present encrypted/signed data to unauthorized parties but will page-fault when a component other than the associated CVM tries to access it.

In figure 9, an H100 GPU is assigned to VM[1], which has been configured with its associated memory ASID (address-space identifier)[1]. Any access to memory in ASID[1] from outside of VM[1] will result in the previously mentioned fault unless the VM[1] specifically marks certain pages as "shared" (the gray box within ASID[1].)

The H100 GPU has DMA engines with encrypt/decrypt capability, which are responsible for the movement of data to and from the CPU's memory. In a confidential environment, DMA engines are allowed to access shared memory pages to retrieve and place data. To ensure the confidentiality and integrity of the payloads, models, and data, the data in these pages is encrypted and signed. These shared memory regions are called *bounce buffers* because they are used to stage the secured data before it is transferred into the secured memory enclaves, decrypted, authenticated, and then processed.

Figure 10 shows the layout of CPU Memory and GPU Memory and the location of encrypted bounce buffers. NVIDIA provides developers with a solution called UVM (unified virtual memory) that automatically handles page migrations between the GPU memory and the CPU memory based on a memory allocation API called `cudaMallocManaged()`. When the CPU accesses the data, UVM migrates the pages to the CPU system memory.

FIGURE 9: **CONFIDENTIAL H100 GPU WITH AN AMD SEV-SNP TEE**

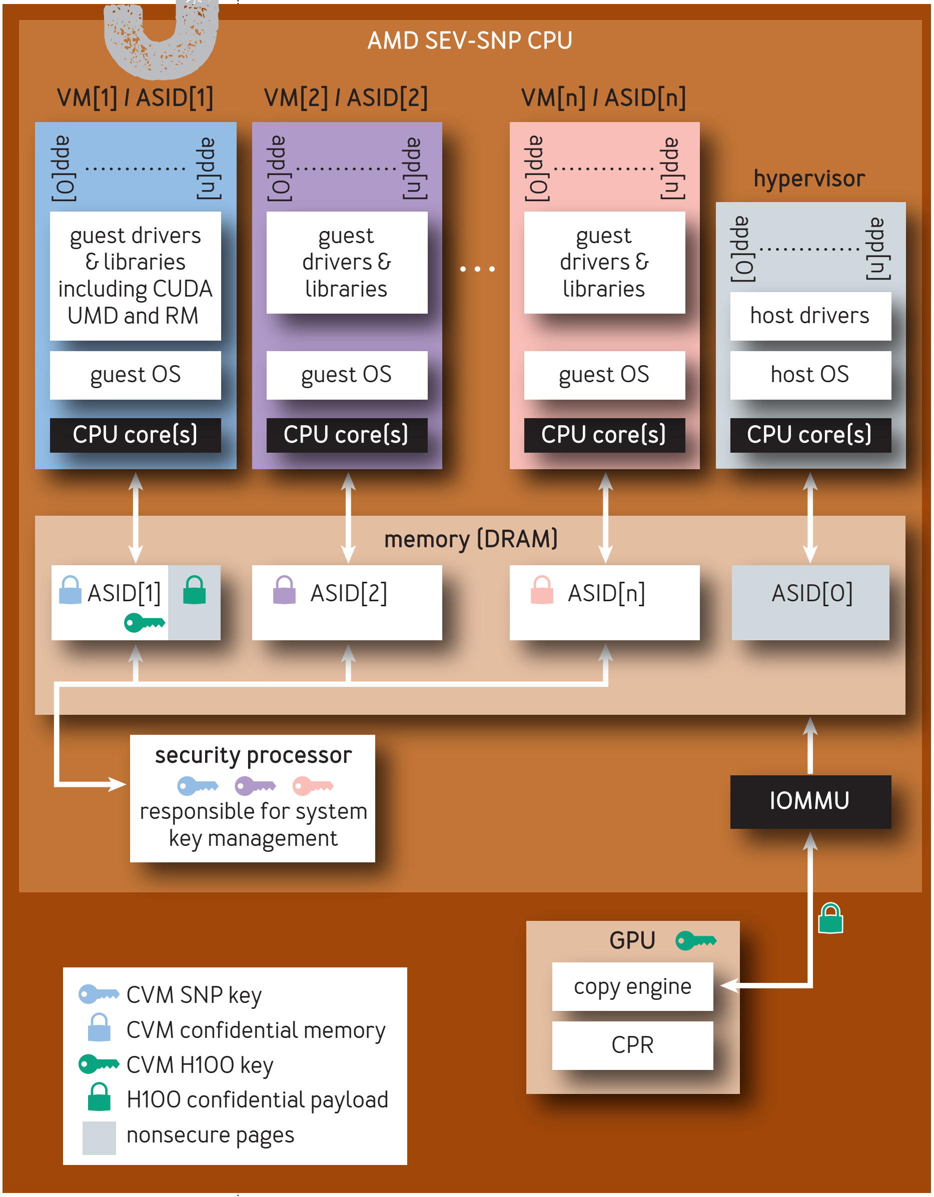
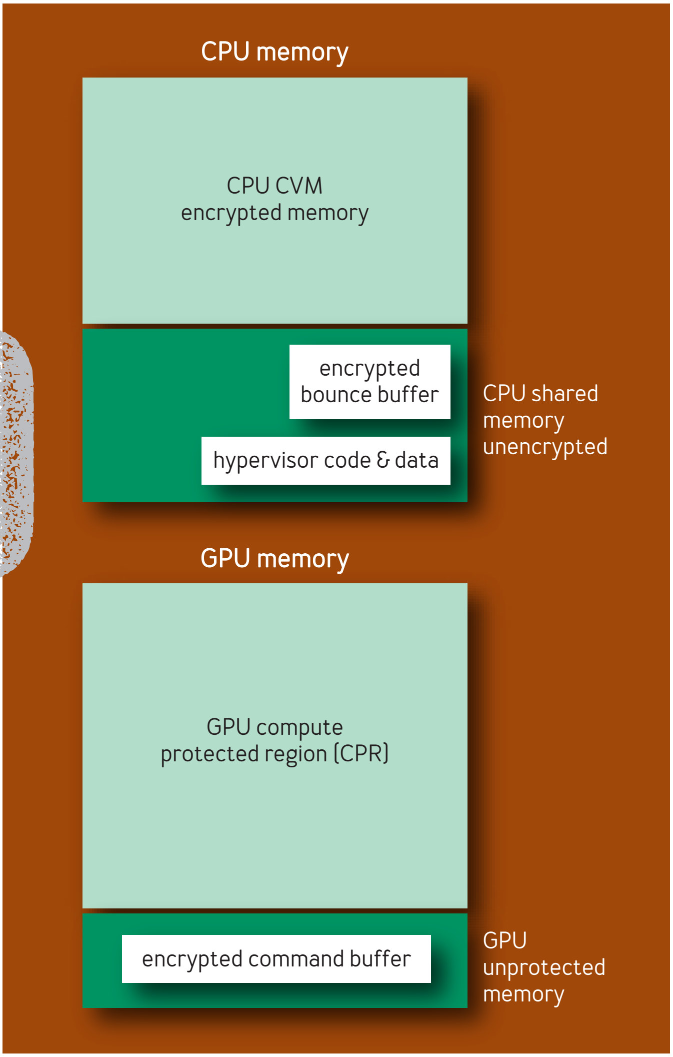


FIGURE 10: **PROTECTED MEMORY REGIONS FOR CPU AND GPU**



When the data is needed on the GPU, UVM migrates it to the GPU memory. For CC, UVM was extended to use encrypted and authenticated paging through bounce buffers in shared memory.

Following is a summary of some of the considerations that developers should be aware of when using the H100 in CC mode.

- Because of how CPU vendors isolate their CVM memory from outside sources, pinned memory allocations such as from `cudaHostAlloc()` and `cudaMallocHost()` cannot be directly accessed by the GPU. Instead, they are handled by UVM with encrypted paging, as if they were allocated by `cudaManagedAlloc()`. This means pinned memory accesses are slower in CC mode.
- `cudaHostRegister()` cannot be supported because this API gives direct access to memory created by `malloc()` or `new()` inside the CVM. This API, among a few others, will return an error code when the GPU is in CC modes. `cudaHostRegister()` does not have widespread use in NVIDIA libraries, and where it is used, we are modifying the code paths to work seamlessly with the H100 in CC mode.
- Developers must use the `nvidia-persistenced` daemon when using the H100 GPU in CC mode to keep the driver loaded, even when not in use. In a typical operation, when the NVIDIA device resources are no longer being used, the NVIDIA kernel driver tears down the device state. In CC mode, however, this would lead to destroying the shared session keys that were established during the setup SPDM phase of the driver. To protect user data, the GPU does not allow the restart of an SPDM

session establishment without an FLR, which resets and scrubs the GPU. `nvidia-persistenced` provides a configuration option called persistence mode that can be set by NVIDIA management software, such as `nvidia-smi`. When the persistence mode is enabled, the NVIDIA kernel driver is prevented from exiting. `nvidia-persistenced` does not use any device resources; it simply sleeps while maintaining a reference to the NVIDIA device state.

With these considerations in mind, users can proceed to use the H100 GPU in CC mode.

PERFORMANCE

A primary goal of delivering CC to customers is that CUDA applications can run unchanged while maximizing the acceleration potential of the underlying hardware and software. CUDA provides lift-and-shift benefits to applications that will be run in CC mode. As a result, the NVIDIA GPU CC architecture is compatible with the CPU architectures that also provide application portability from nonconfidential to CC environments.

Given the description so far, it should not be surprising that CC workloads on the GPU perform close to non-CC mode when the amount of compute is large compared with the amount of input data. When the amount of compute is low compared with the input data, the overhead of communicating across the nonsecure interconnect limits the application throughput.

To help understand performance in CC mode, these performance primitives are on par with nonconfidential mode:

FIGURE 11: **EXAMPLE SERVER TOPOLOGY WITH GPUS IN CC ON AND OFF**

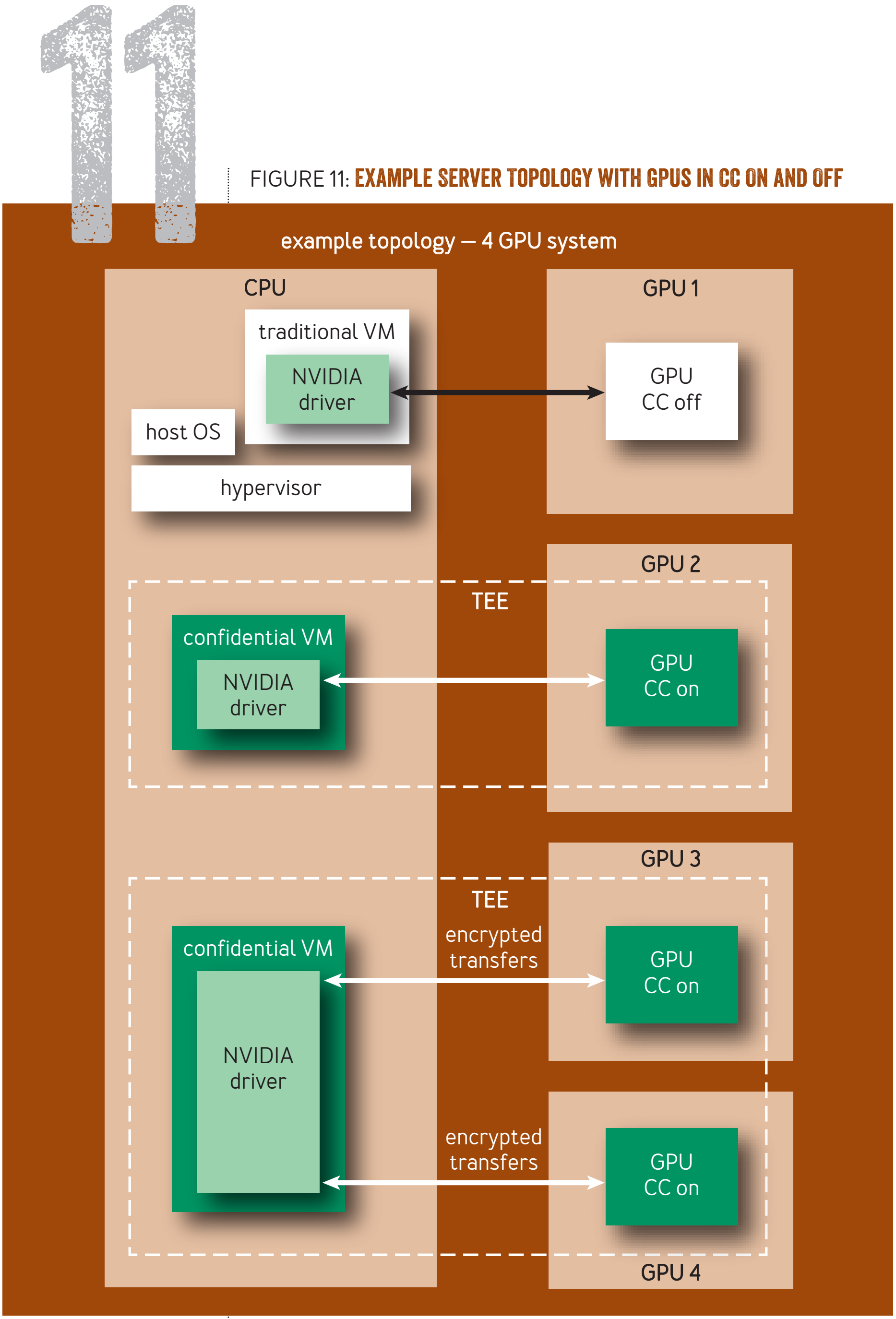
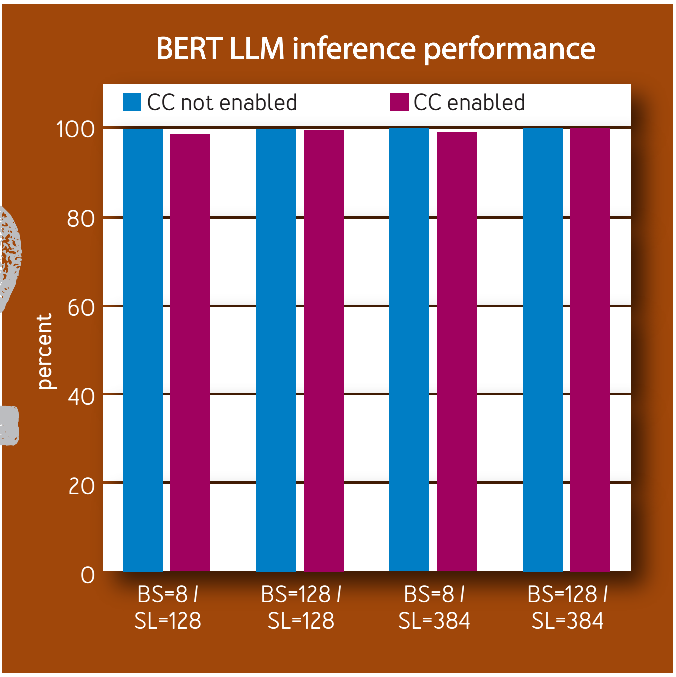


FIGURE 12: **EXAMPLE OF A WORKLOAD WITH HIGH COMPUTE TO I/O RATIO**

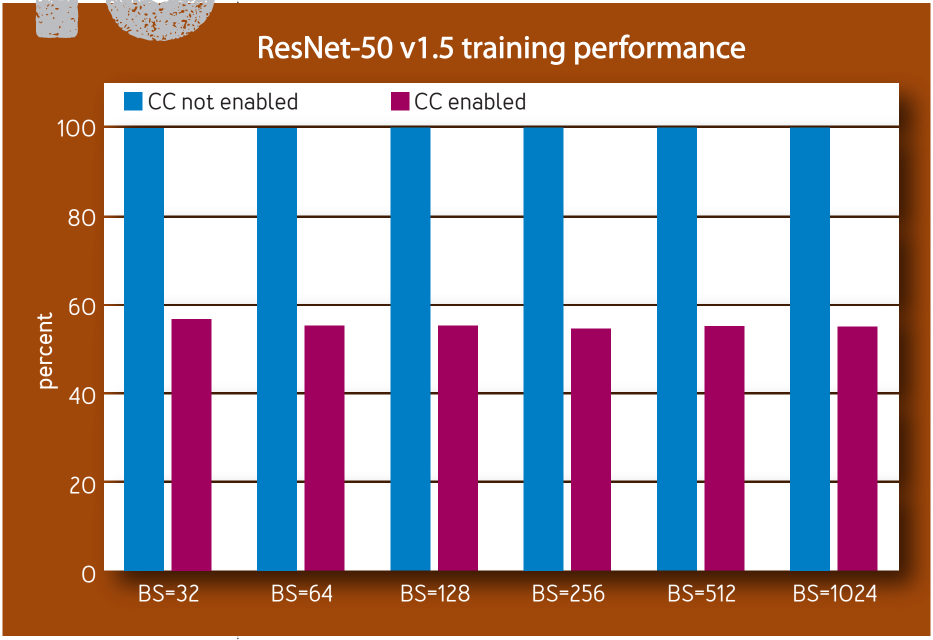


- *GPU raw compute performance.* The compute engines execute unencrypted code on unencrypted data resident in GPU memory.
- *GPU memory bandwidth.* The on-package HBM is considered secure against common physical attack tools, such as interposers, and is not encrypted.

These performance primitives are impacted by additional encryption and decryption overheads:

- CPU-GPU interconnect bandwidth is limited by CPU encryption performance. Currently this is approximately 4GB/sec.

FIGURE 13: **EXAMPLE OF A WORKLOAD WITH A LOW COMPUTE TO I/O RATIO**



- Data-transfer across the PCIe bus incurs a latency overhead for transit through encrypted bounce buffers in shared memory.

Figure 11 shows an example server topology with GPUs in and out of CC mode.

There is also an overhead for encrypting GPU command buffers, synchronization primitives, exception metadata, and other internal driver data that is exchanged between the GPU and the confidential VM running on the CPU. Encrypting and authenticating these data structures prevents side-channel attacks on the user data.

Figure 12 shows an example of a workload with a high compute-to-I/O ratio, and figure 13 is an example of a workload with a low compute-to-I/O ratio. BS is batch size, and SL is sequence length.

SUMMARY

CC is available for H100 Tensor Core GPUs as an early access feature as of CUDA 12.2, released in July 2023. The CC feature will become generally available after we complete performance optimization and allow for sufficient security soak time. The key value propositions delivered with this feature are:

- We bring CC to the most demanding workloads such as AI, machine learning, and high-performance computing.
- Existing CUDA programs, deep-learning frameworks, etc., run without changes.
- User code and user data are protected end to end.
- The GPU and its firmware are attested as part of total platform attestation.

Creating the first confidential GPU has been an exciting journey for the entire team at NVIDIA and for our collaborators at other companies who are committed to the confidential computing vision. Today, confidential computing is a great innovation. In a few years' time, we expect all computing will be confidential, and we will all wonder why it was ever any other way.

Gobikrishna Dhanuskodi *is a distinguished engineer in the GPU system software group and a lead software architect of GPU Confidential Computing at NVIDIA. During his long tenure at NVIDIA, he has worked primarily on product*

security, DRM solutions, and GPU virtualization technologies. He is currently focused on enabling the use of CC technologies in accelerated computing and making it ubiquitous across a broader spectrum of use cases.

Sudeshna Guha *is a senior system software engineer at NVIDIA. As a member of the Confidential Computing working group, she develops the CUDA driver and runtime for GPU confidential computing. In 18 years of hardware and software engineering leadership roles, she has architected and designed many hardware and software features and processes across generations of NVIDIA SOCs and GPUs.*

Vidhya Krishnan *is a distinguished architect and the lead hardware architect for NVIDIA GPU confidential compute. She has worked on GPUs for the majority of her career. She is passionate about confidential computing as a technology and looks forward to it becoming the default mode of deployment.*

Aruna Manjunatha *is a director of system software engineering in the GPU software team at NVIDIA. She has worked for almost 15 years in the kernel-mode driver software team, taking new GPU families from design to production. Her latest role has her working as the software engineering lead for GPU confidential computing. She is keen about mentoring and coaching, which she views as a great way to learn from others.*

Michael O'Connor *is a system software architect and senior distinguished engineer at NVIDIA. He has been at NVIDIA for almost 10 years, focusing primarily on GPU optimization of deep-learning frameworks such as PyTorch, TensorFlow, and*

MXNet. He joined the confidential computing team a year ago to focus on attestation and overall workflow.

Rob Nertney *is a senior technical product manager for CUDA. He has spent nearly 15 years architecting the features and deployment of accelerator hardware into hyperscale environments for both internal and external use by developers. He has several patents in processor design relating to secure solutions that are in production today. In his spare time, he loves golfing when the weather is nice, and gaming (on RTX hardware, of course) when it isn't.*

Phil Rogers *is a compute server software architect and vice president of system software at NVIDIA. He is the software leader for multiple programs at NVIDIA, including confidential computing, Fleet Command, NVIDIA-certified systems, long-term support for the whole stack, and NGC. Phil is passionate about all aspects of accelerated computing, including ease of use, performance, scalability, and security.*

Copyright © 2023 held by owner/author. Publication rights licensed to ACM.

CONTENTS