Supporting Secure Multi-GPU Computing with Dynamic and Batched Metadata Management

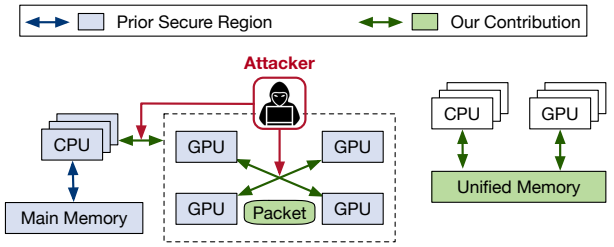
Seonjin Na†*, Jungwoo Kim‡, Sunho Lee‡ and Jaehyuk Huh‡
Georgia Institute of Technology†, KAIST‡
† seonjin.na@gatech.edu, ‡ jwkim@casys.kaist.ac.kr, myshlee417@casys.kaist.ac.kr, jhhuh@kaist.ac.kr

***Abstract*—With growing problem sizes for GPU computing, multi-GPU systems with fine-grained memory sharing have emerged to improve the current coarse-grained unified memory support based on page migration. Such multi-GPU systems with shared memory pose a new challenge in securing CPU-GPU and inter-GPU communications, as the cost of secure data transfers adds a significant performance overhead. There are two overheads of secure communication in multi-GPU systems: First, extra overhead is added to generate one-time pads (OTPs) for authenticated encryption. Second, the security metadata such as MACs and counters passed along with encrypted data consume precious network bandwidth. This study investigates the performance impact of secure communication in multi-GPU systems and evaluates the prior CPU-oriented OTP precomputation schemes adapted for multi-GPU systems. Our investigation identifies the challenge with the limited OTP buffers for inter-GPU communication and the opportunity to reduce traffic for security meta-data with bursty communications in GPUs. Based on the analysis, this paper proposes a new dynamic OTP buffer allocation technique, which adjusts the buffer assignment for each source-destination pair to reflect the communication patterns. To address the bandwidth problem by extra security metadata, the study employs a dynamic batching scheme to transfer only a single set of metadata for each batched group of data responses. The proposed design constantly tracks the communication pattern from each GPU, periodically adjusts the allocated buffer size, and dynamically forms batches of data transfers. Our evaluation shows that in a 16-GPU system, the proposed scheme can improve the performance by 13.2% and 17.5% on average from the prior cached and private schemes, respectively.**

I. Introduction

As the problem range of GPU applications has been increasing, multi-GPU systems have emerged to solve large-scale problems, which cannot be handled by a single GPU. Such multi-GPU systems can employ dedicated networks to increase the bandwidth among GPUs while lowering latencies. Commercial systems integrate 8-16 GPUs in a single node [16], [31], and a recent study proposed a rack-scale multi-GPU system by using PCIe extension techniques [51]. Another important new aspect of multi-GPU systems is to support block-level shared memory, instead of using the conventional page-level copy-based mechanism. A GPU can access the memory of another GPU at cache-block granularity, which enables efficient fine-grained memory sharing among GPUs [33], [48].

Meanwhile, with the increasing adoption of GPUs for mission-critical domains, the security requirement for GPU computing has become important from private data centers to public GPU cloud services. To improve the security support for GPUs, recent studies extended trusted execution environments, which have been used only for CPUs, to GPU systems [21], [45]. Such trusted GPU supports not only isolate a GPU context from the malicious operating system but also provide the protection of its external memory. However, the prior protection techniques are limited to a single GPU system, and the consideration of emerging multi-GPU systems with fine-grained shared memory is yet to be investigated.



(a) Physical view (b) Logical view

Fig. 1: Overview of our proposed multi-GPU system and contribution. Supporting secure communication between GPUs can extend a trusted execution environment to a multi-GPU system with unified memory.

A new problem arising from the trusted multi-GPU systems is the efficient protection of communication among GPUs, in addition to the communication between the host CPU memory and GPU. As shown in the prior work [45], the memory of each GPU is protected from physical attacks by using 3D stacked DRAMs, which have become common in high-performance GPUs. However, the communication channels between GPUs are vulnerable to physical attacks, and thus they must be encrypted and integrity-validated. Figure 1 describes the communication security problem with a trusted multi-GPU system. GPUs share the memory with fine-grained block-level accesses, while the communication channels are vulnerable.

However, securing the channels has performance overheads. Encrypting plaintext data and generating message-authentication code (MAC) in the sender GPU, and decrypting the ciphertext and validating the MAC in the receiver GPU add extra latencies for inter-GPU communication. The data encryption uses a unique counter for each message to randomize the ciphertext and to detect replay attacks. Such protection schemes incur two overheads in multi-GPU systems. First, the

*This work was done at KAIST.

encryption and MAC generation add extra latencies for data transfers. Second, the security metadata such as MAC and counter sent along with the ciphertext consume the limited network bandwidth.

To reduce the overheads of OTP generations, the prior studies on shared memory multiprocessors pre-compute one-time pads (OTPs) in the sender and receiver [35]. They use the pre-computed and buffered OTPs to avoid generating OTP on demands, hiding the latency from the critical path of communication. To hide even the address and request type, a recent study proposed to encrypt the request packet content too [34]. Such OTP pre-computation raises a problem of how to manage OTP buffers.

A straightforward scheme is *Private*, which has a separate buffer array for each pair of source and destination. Per-pair buffers allow a pair of sender and receiver to synchronize the counter perfectly, as the counter advances for each communication. However, the required buffer size increases quadratically as the number of GPUs increases. The prior study investigated alternative schemes to reduce the buffer overheads, by sharing the buffers in each source (*Shared*), or by caching only OTPs for recently communicating pairs (*Cached*) [35]. However, the capacity-optimized schemes have lower performance than the *Private* scheme.

This paper first revisits the OTP buffering schemes designed for CPU multiprocessors and adapts them to multi-GPU systems. We evaluate the performance impact of communication encryption and the effect of different OTP management schemes for multiple GPUs. In the evaluation, although *Private* and *Cached* schemes excel *Shared*, they still incur significant performance overheads, calling for better OTP management techniques for multi-GPU systems. In addition, the analysis shows that multi-GPU systems commonly exhibit bursty communications between GPUs, which provide an opportunity to reduce the traffic increase by security metadata.

To address the problem of OTP precomputation, this paper proposes a new dynamic OTP buffer management scheme to identify the communication patterns among GPUs and to adjust the buffer allocation dynamically. To support such dynamic buffer management, this study proposes a communication pattern detection algorithm, and the buffer adjustment scheme to reflect communication pattern changes. The inter-processor communication is constantly tracked and the buffer size for each source-destination pair is dynamically adjusted.

To address the bandwidth consumption by the security metadata, this study uses a *dynamic batching* scheme which combines multiple data responses. Only one set of security metadata is sent along with the combined batched data. Such traffic reduction schemes for security metadata have been studied for GPU memory accesses [30], [49], but this paper applies them to the secure CPU-GPU and inter-GPU communication problem.

We evaluate the proposed dynamic OTP buffer management and metadata batching schemes with simulated 4, 8, and 16 GPU systems. Our scheme improves the performance of the prior *Private* technique by 11.6%, 17.1%, and 17.5% for 4, 8, and 16 GPU systems, respectively. Moreover, our approach results in performance improvements of 8.4%, 9.2%, and 13.2% for 4, 8, and 16 GPU systems compared to the previous *Cached* mechanism. This study investigates the performance impact of inter-GPU communication for secure multi-GPU systems with shared memory. The contributions of the paper are as follows:

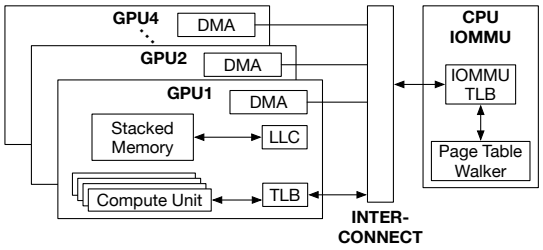


Fig. 2: Target Multi-GPU architecture. Each GPU has stacked high bandwidth memory and GPUs share their memories at cacheline granularity.

- This study investigates the costs of securing communications for multi-GPU systems by adopting the prior CPU-oriented message encryption schemes for shared memory.
- It proposes a dynamic OTP buffer allocation scheme to maximize the efficiency of pre-generated OTPs by exploiting communication patterns of multi-GPU systems.
- It proposes to dynamically batch multiple data responses to amortize the bandwidth consumption for security metadata.

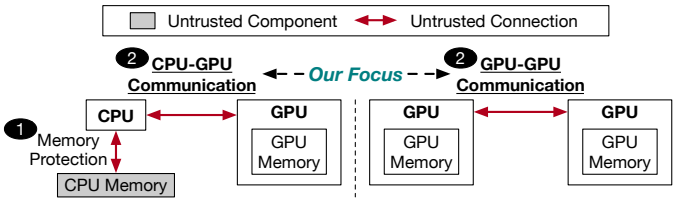
II. Background

A. Multi-GPU Architecture

Figure 2 shows the discrete multi-GPU system organization. Each GPU in the system consists of many compute units (CUs) and communication engines. There are multiple single instruction multiple data (SIMD) units within each CU. For the organization of the GPU memory hierarchy, each CU has a private L1 data cache and the L1 instruction cache is shared across all CUs in a shader array. All the CUs in the GPU share a unified L2 cache. Each CU has a private L1 TLB as shown and the L2 TLB is shared among all CUs within a GPU. Miss requests in the L2 TLB are forwarded to the Input/Output Memory Management Unit (IOMMU) on the CPU side.

Inter-processor communication mechanisms: In a multi-GPU system, The GPUs are connected through high-bandwidth off-chip interconnects such as NVLink or InfiniBand, while the CPU and GPUs communicate over off-chip PCIe interconnects. GPUs can communicate with each other or with CPUs using two different methods: (1) page migration and (2) direct block access.

For CPU-GPU and GPU-GPU communication through page migration, the required data page is transferred and mapped to the requesting GPU's memory. This allows subsequent data requests by the GPU to be accessed locally. However, such a page migration incurs significant performance overheads due to additional processes required by the GPU driver including



(a) CPU-GPU Communication (b) GPU-GPU Communication

Fig. 3: The required secure communication between CPU-GPU and inter-GPU connection.

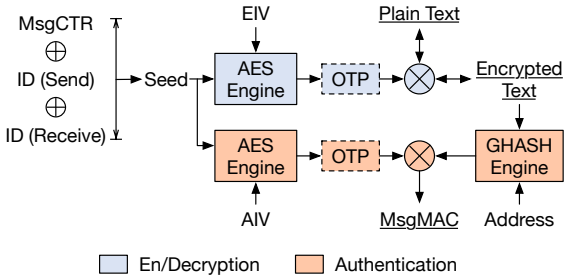


Fig. 4: En/decryption and authentication in a multi-GPU system. The message counter and both IDs of sender and receiver are leveraged to generate pads and each pad is used for en/decryption and authentication.

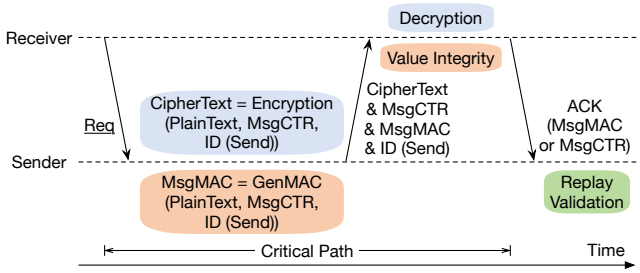


Fig. 5: The flow of secure communication between two processors.

TLB shootdowns. Hence, the page migration is best suited for data that demonstrates high locality.

On the other hand, inter-processor direct block transfers enable peer-to-peer, cacheline granularity data access without page migration. When a GPU accesses data with low localities, the direct block access is preferable over page migration. Utilizing the locality API provides hints to the GPU driver, enabling the optimal choice between page migration and direct block access as explained in [32].

B. Threat Model

We assume the multi-GPU architecture adopts High Bandwidth Memory (HBM) instead of GDDR-based GPU memory. In this work, we assume a unified multi-GPU model [6], [7], [24], [48] that enables GPU programs written for a single-GPU system to seamlessly operate on the multi-GPU system [38], [39]. The unified memory can provide a single address space accessible to the CPU and GPUs in the system, and the memory can be allocated and migrated on demand. Our target architectures allow both page migration and direct block access for CPU-GPU and inter-GPU communications.

Trusted Computing Base (TCB) includes both the CPU and GPU chips, as well as the GPU software operating within the Trusted Execution Environments (TEEs). The secure monitor that manages enclave resources, such as page tables, is also part of the TCB similar to previous study [13]. In this study, we assume that attackers can gain control over privileged software (such as the OS and hypervisor) and attackers have physical accesses to the entire system. Therefore, hardware components, including off-chip DRAM in CPUs, I/O interconnects, and inter-GPU networks are vulnerable to attacks. Although the CPU-side off-chip DRAM is untrusted, the GPU-side High Bandwidth Memory (HBM) is considered secure from physical attacks [45]. However, the CPU-GPU data transfer via PCIe interconnects, and the GPU-GPU communications are vulnerable to physical attacks.

However, this study does not address side-channel attacks, including those related to cache-timing attacks [11], [46] and transient-execution attacks [25], [29]. Furthermore, it does not cover availability attacks on GPU computation and denial-of-service attacks, aligning with the prior work [9], [21], [30], [45].

C. Secure Communication Mechanism between Processors

There have been several studies for providing secure communication for CPU-based multi-processors with counter-mode encryption schemes [26], [35], [36], [40]. Such communication protection schemes can be applied to the CPU-GPU and inter-GPU communications. Figure 3 illustrates that the secure multi-GPU architecture needs data protection not only between the processor and memory but also CPU-GPU and GPU-GPU communications. In this study, our primary focus is on protecting the transmitted data through the interconnects for such communications.

Counter-mode encryption/authentication: Figure 4 shows how counter-mode protection is used in communication between processors. The sender processor starts by creating a unique seed that combines a message counter (MsgCTR), sender ID, and receiver ID. This seed, along with initialization vectors, is used in the AES engines to generate one-time pads (OTPs) for both encryption/decryption and authentication. It transforms a plaintext memory block into the ciphertext and vice versa, requiring only an additional 1 cycle for an XOR operation. For authentication, a separate pad generates a message authentication code (MsgMAC) for the communication message, ensuring its integrity by comparing the generated MsgMAC with the transferred MsgMAC. As generating the pad requires the related MsgCTR without the actual data block, the latency of secure communication can be hidden if all pads can be pre-generated.

Replay attack protection: Attackers can perform a replay

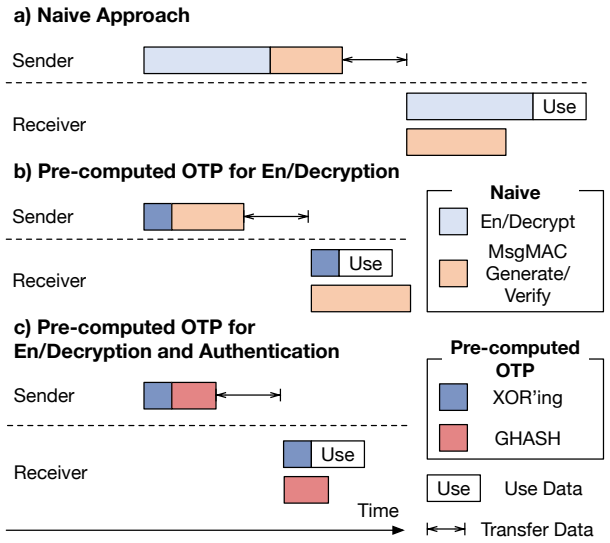


Fig. 6: The performance improvement enabled by the prepared pad. Pre-generated pads reduce the latency of en/decryption and authentication to only an XOR and a GHASH computation time.

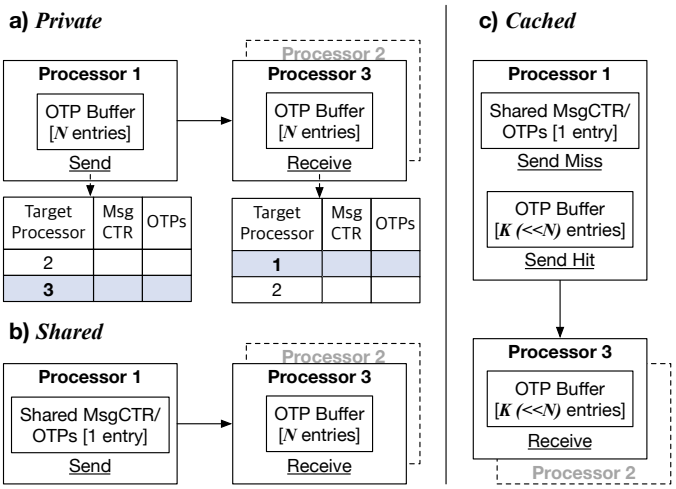


Fig. 7: Three methods for OTP buffer management: a) *Private:* Each processor maintains two pad tables to accelerate secure send and receive operations. Since these pad tables hold different message counters (MsgCTR) for each sender-receiver pair, the overhead of storage is high. b) *Shared:* Single MsgCTR is leveraged for a send procedure. As it eliminates a sender pad table, the pad-hit only occurs in a back-to-back send message to the same processor. c) *Cached:* It is a hybrid method of *Private* and *Shared*. MsgCTR and pad are stored in cache-like pad tables which store only a small portion of entries and occasionally remove an entry in the LRU manner.

attack by resending an earlier ciphertext along with its metadata, which includes the message counter (MsgCTR), message authentication code (MsgMAC), and sender ID. To prevent this attack, the sender stores either the MsgCTR or MsgMAC until it receives an acknowledgment (ACK) from the receiver. This ACK also contains the MsgCTR or MsgMAC. Upon receiving the ACK, the sender can compare its value with the stored value to verify its freshness.

Procedure of secure communication: Figure 5 shows the process of inter-processor secure communication. When a request reaches the sender, it encrypts the plaintext using the message counter (MsgCTR) and simultaneously generates a message authentication code (MsgMAC). The sender then transmits the encrypted data along with MsgCTR, MsgMAC, and its ID. Upon receipt, the receiver simultaneously carries out value integrity authentication and decrypts the message. To complete the process, the receiver sends back an acknowledgment (ACK), which includes either the MsgMAC or MsgCTR. The sender uses this ACK to check for any replay attacks.

Advantage of pad pre-generation: Figure 6 compares the total latency to perform inter-processor secure communication, depending on whether the encryption and authentication pads can be prepared in advance. Figures 6 a) and b) show that pre-creating the encryption/decryption pad can reduce the time by replacing the lengthy encryption/decryption process with a simple one-cycle XOR operation.

Similarly, as shown in Figure 6 c), generating an authentication pad in advance significantly speeds up the authentication process, similar to the encryption/decryption process. To take full advantage of these pre-generated pads, they must be created using the same message counter (MsgCTR) value that the sender uses. Therefore, keeping the MsgCTR values in sync between the sender and receiver is essential to minimize any extra latency caused by securing the communication.

Policies of the prior OTP buffer managements: Figure 7 represents three different mechanisms of secure inter-processor communication proposed for CPUs [35]: a) *Private*, b) *Shared*, and c) *Cached*. In the *Private* mechanism, each processor has its message counters for every other processor it communicates with and also maintains separate pad table entries for these destination processors as shown in Figure 7 a). Since there are pad table entries for every communicating pair, the possibility of hiding latency during the secure communication process is high due to the pre-generation of encryption and authentication pads, offering a performance advantage. However, as the number of processors in the system increases, the additional storage required for the pad table also increases quadratically. To reduce storage overhead, *Shared* uses a single table entry with a common MsgCTR for sending data to any processor. It generates a seed used in encryption/decryption and authentication pads without needing the receiver's ID. However, as illustrated in Figure 7 b), having only one entry for the send direction can hide the encryption latency in the sender, but the receiver can pre-generate pads correctly only when the sender transfers data back-to-back to the same receiver.

Cached maintains the table with the static number of entries with the least recently used (LRU) policy. When the pre-generated pad exists in the table of *Cached*, the process of security verification follows *Private*, and otherwise, it adopts *Shared* using the maximum MsgCTR value.

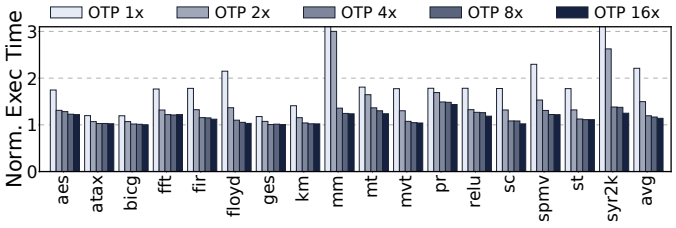


Fig. 8: Performance impact of OTP buffer entries with *Private* in a 4-GPU system.

	OTP Config	1×	2×	4×	8×	16×
4 GPUs	Storage	2.50 KB	5.01 KB	10.02 KB	20.03 KB	40.06 KB
	# of OTPs	32 OTPs	64 OTPs	128 OTPs	256 OTPs	512 OTPs
8 GPUs	Storage	10.02 KB	20.03 KB	40.06 KB	80.13 KB	160.25 KB
	# of OTPs	128 OTPs	256 OTPs	512 OTPs	1024 OTPs	2048 OTPs
16 GPUs	Storage	40.06 KB	80.13 KB	160.25 KB	320.50 KB	641.00 KB
	# of OTPs	512 OTPs	1024 OTPs	2048 OTPs	4096 OTPs	8192 OTPs
32 GPUs	Storage	160.25 KB	320.50 KB	641.00 KB	1282.00 KB	2564.00 KB
	# of OTPs	2048 OTPs	4096 OTPs	8192 OTPs	16384 OTPs	32768 OTPs

TABLE I: On-chip storage overhead and the number of total OTP buffer entries in *Private* scheme.

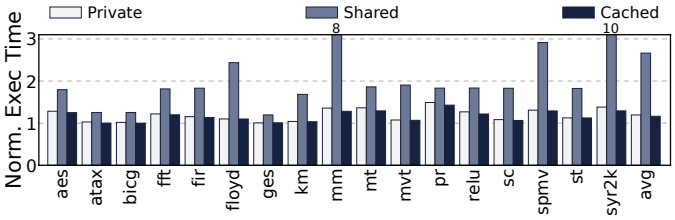


Fig. 9: Performance overhead by secure communication with *OTP 4x* in a 4-GPU system.

D. GPU TEE

To provide a trusted execution environment (TEE) on GPUs, recent studies proposed extending the CPU TEE protection to GPUs [21], [45]. Graviton uses a hardware-based solution to provide isolated GPU execution from untrusted privileged software. In current GPU systems, since critical GPU resources such as GPU page tables are managed by the untrusted GPU driver, a compromised GPU driver can violate the isolation between GPU contexts. To provide GPU context isolation, Graviton assumes the GPU command processor as a trusted computing base (TCB) and makes the trusted command processor perform critical GPU operations instead of the untrusted GPU driver [45]. HIX, on the other hand, ensures GPU context isolation by securing the PCIe I/O path between CPU and GPU without necessitating any changes to the GPU hardware [21]. It defines a GPU enclave to separate the GPU driver from the untrusted OS, managing essential resources and controlling GPU.

Both Graviton and HIX assume a traditional copy-and-execute GPU programming model, where authenticated encryption occurs only at the start and end of GPU kernel operations. However, our focus is on multi-GPU systems with unified memory, where page migration between CPU-GPU and GPU-GPU happens even during GPU kernel execution. This study focuses on minimizing performance overheads caused by protecting the data transmitted through the interconnects in the CPU-GPU and GPU-GPU communications. Therefore, we adopt a system equipped with protected per-enclave page tables and scalable memory protection techniques, similar to PENGLAI [13], and assume GPU TEE can be provided as in Graviton [45].

III. Motivation

A. Performance Implications of Secure Multi-GPU

To explore the performance overheads of secure multi-GPU systems, we conducted empirical studies through simulation, The detailed methodology for the multi-GPU simulation is described in Section V.

Performance impact of the number of OTP buffer entries: To address the performance impact caused by burst requests during authenticated encryption, we conducted a sensitivity analysis by varying the number of OTP buffer entries for each send/receive table under the *Private* technique.

Figure 8 represents the performance changes within the *Private* technique in a 4-GPU system as the number of OTP buffer entries for each send/receive OTP table is increased from 1 to 16. The configuration, OTP *Nx*, represents *N* OTP buffers for each source-destination pair. For OPT 1x, for each pair of source and destination GPUs, each of the source and destination GPUs has one OPT buffer entry for the communication path. OTP 1x results in a 121.1% performance degradation on average under the *Pivate* mechanism. However, this degradation decreases to 14.0% when OTP 16x is used.

Nevertheless, it is important to note that with the increase in the number of GPUs in a system, the on-chip storage required for these OTP buffers increases rapidly. Table I presents the total on-chip storage overhead with the number of OTP buffer entries in a multi-GPU system. A 16× system with 32 GPUs may necessitate up to 2564KB of on-chip storage and 32768 OTPs. With the growing demand for large-scale systems, it is expected that future systems will incorporate a substantial number of GPUs. This increase complicates the feasibility of utilizing a large number of OTP buffer entries. Through our sensitivity analysis, we observed that efficient OTP usage is crucial in such scenarios.

Comparison of prior OTP buffer management schemes: Figure 9 shows the execution times of the prior OTP buffer management schemes normalized to the unsecure multi-GPU system. In this experiment, we compare three secure communication techniques discussed in Section II-C. Throughout these comparisons, the size of the on-chip OTP buffer is kept constant for all techniques, aligned with the *Private* scheme. In a 4-GPU system with *OTP 4x*, there are 4 (3(GPUs) + 1(CPU)) × 2 (Send/Recv) × 4 OTP buffers in each GPU with the *Private* scheme. In the *Shared* scheme, each GPU has 32 OTP buffers, 1 buffer for sending data blocks to all processors,

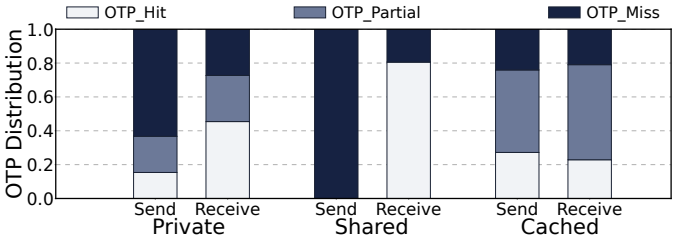


Fig. 10: The distribution of one-time pad (OTP) latency within authenticated encryption/decryption on the 4-GPU system (*OTP 4x*).

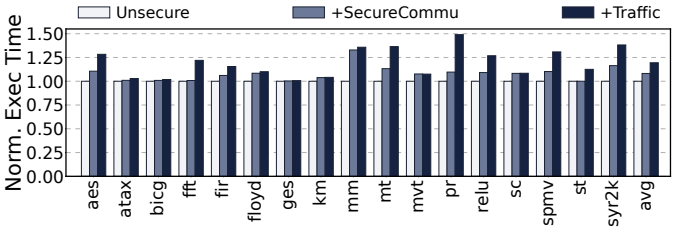


Fig. 11: Execution times when secure communication and security metadata are cumulatively considered, normalized to the unsecure 4-GPU system.

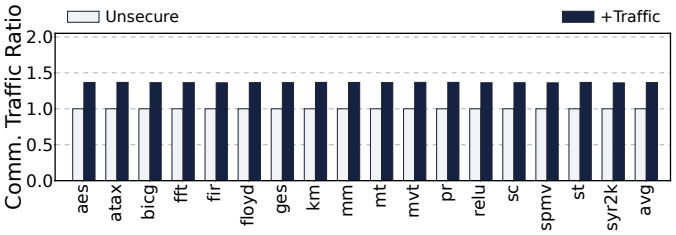


Fig. 12: Ratio of communication traffic for considering memory protection, secure communication, and security metadata together, compared to the unsecure 4-GPU system.

and 31 buffers for receiving data blocks as shown in Figure 7. In the *Cached* scheme, we assume there are 32 OTP buffers per each GPU.

This experiment investigates the performance degradation due to the OTP generation. On average, we observed performance degradations of 19.5%, 166.3%, and 16.3% for the *Private*, *Shared*, and *Cached* techniques, respectively, across all evaluated workloads. With the same number of OTP buffers, *Private* and *Cached* excel *Shared*. We will compare our techniques against the two schemes (*Private* and *Cached*) in the evaluation.

Executing GPU applications in the multi-GPU system with high levels of thread-level parallelism demonstrates a tendency to initiate bursts of remote requests to other GPUs. Such burst remote requests lead to OTP misses in the pad table, increasing the overhead of secure communication. The authenticated encryption latencies with pad table can be fully hidden (*OTP_Hit*), partially hidden (*OTP_Partial*), or not-hidden (*OTP_Miss*). Figure 10 represents the distribution ratio of average latency hiding for authenticated encryption/decryption across all evaluated benchmarks. In the *Private* mechanism, 36.9%, and 72.7% of the authenticated encryption latency is fully hidden or partially hidden in the send direction and receive direction, respectively. On the contrary, *Shared* mechanism cannot hide the authenticated encryption latency on the sender side since the MsgCTR counter is shared for sending requests to any GPUs. In the *Cached* technique, due to the more flexible allocation of OTP buffer entries in both the send and receive directions, On average, it can partially or fully hide 75.9% of the authenticated encryption latency and 79.0% of the authenticated decryption latency.

Performance overhead analysis: To better understand the factors contributing to performance degradation in secure multi-GPU systems, we investigated the impact of OTP 4x in the *Private* scheme. Figure 11 illustrates the performance degradation of securing a multi-GPU system compared to an unsecure system. We analyze two scenarios in Figure 11: **+SecureCommu**, which applies secure communication without the metadata overhead, and **+Traffic**, which considers additional bandwidth required for security metadata during secure communication. In the multi-GPU system with *OTP 4x*, we observed that the average performance overhead by secure communication is 8.2%. This overhead further increases by 11.3% when considering the additional bandwidth required for metadata. To achieve secure communication, extra metadata (i.e. MsgCTR, MsgMAC, sender ID, and ACK) are required for encryption, authentication, and replay attack protection. Since the metadata are added to each cacheline communication, it significantly increases the communication traffic, leading to performance degradation. Figure 12 shows the interconnect communication traffic analysis which is normalized to a vanilla multi-GPU system without any data protection techniques. On average, the transfer of additional security metadata results in a 36.5% increase in communication traffic.

B. Communication Patterns of Multi-GPU Applications

Dynamic behavior of communication patterns: Figure 13 and 14 show the communication patterns of matrixmultiplication workload observed from a GPU in a 4-GPU system. Figure 13 shows the ratios of send vs. receive on GPU1 as the execution progresses. In the figure, we observe dynamic changes in the ratios during the GPU kernel execution. For further analysis, we break down the remote send requests into each processor (CPU or GPU) destination. Figure 14 shows the decomposition of send requests from GPU1 depending on the destination, CPU or GPU 2, 3, and 4. As shown in the figure, during a time interval, GPU1 sends most of its send requests to the CPU or one or two remote GPUs, showing a certain level of communication locality. In the figure, the communications are not uniform across all GPUs, and the ratios of different destinations also change over the execution. In Section IV, we will introduce an efficient OTP buffer management that exploits the dynamic communication locality to reduce the performance degradation caused by secure communication.

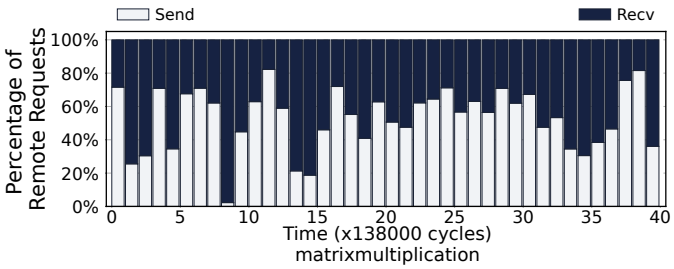


Fig. 13: Distribution of send/receive requests during matrix-multiplication execution on GPU 1.

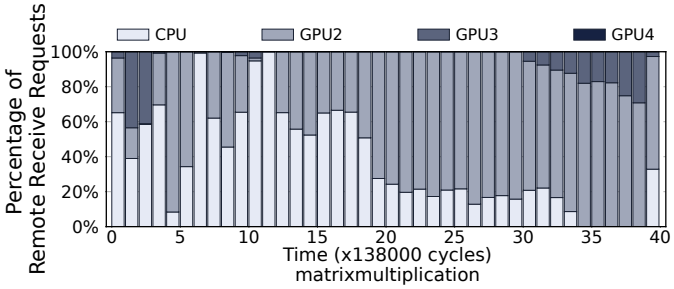


Fig. 14: Distribution of receive requests from GPU 1 during matrix-multiplication execution.

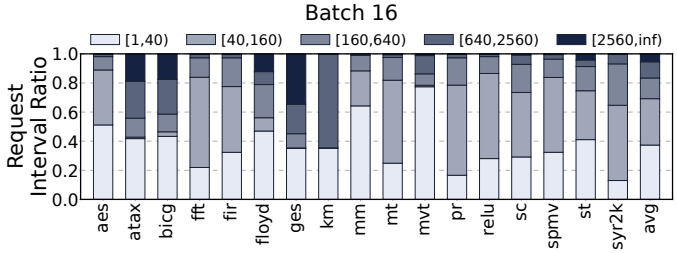


Fig. 15: The ratios of time intervals until 16 data blocks are accumulated. For instance, [40, 160) indicates the range where it takes 40 cycles or more but less than 160 cycles for 16 data requests to arrive.

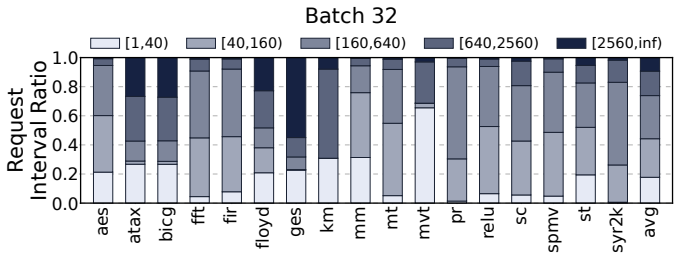


Fig. 16: The ratios of time intervals until 32 data blocks are accumulated. For instance, [40, 160) indicates the range where it takes 40 cycles or more but less than 160 cycles for 32 data blocks to arrive.

Burstiness of communication: We observed that there is a significant amount of data blocks between processors (CPU-GPU or GPU-GPU) within a short time interval. Figure 15 and Figure 16 show the distribution of cycle times required for 16 and 32 data blocks to gather in GPU applications. On average, cases where 16 and 32 data blocks are accumulated within 160 cycles account for 69.2% and 44.2% respectively. Most applications exhibit a trend of generating 16 requests within 160 cycles. This pattern of bursty communication is primarily a result of the multiple thread blocks operating in each GPU.

In this study, we leverage bursty communication in multi-GPU systems to introduce a security metadata batching technique, reducing the bandwidth consumed by additional security metadata over the interconnect.

IV. Architecture

A. Overview

Figure 17 shows an overview of the proposed secure multi-GPU architecture. In this study, we assume that multiple GPUs are connected to CPUs via the PCIe bus, and GPUs communicate with each other through faster interconnects such as NVLink [7], [16], [31]. Although the CPU-GPU and GPU-GPU interconnects are considered untrusted, the stacked memory of each GPU, such as HBM, is regarded as a trusted component [45].

For secure communication, both the CPU and GPU are equipped with our proposed dynamic OTP allocator and batching controller, along with fully pipelined AES-GCM engines. To reduce the performance overhead of secure multi-GPU computing, we propose the dynamic OTP management scheme in Section IV-B by leveraging the communication patterns of multi-GPU workloads discussed in Section III-B. In addition, we introduce a batching technique for security metadata in Section IV-C to reduce the interconnect bandwidth consumed by the metadata based on the observation we explained in Section III-B.

Trusted multi-GPU execution: In our proposed secure multi-GPU system, we assume that an access control mechanism with the GPU TEE (Trusted Execution Environments) support is provided similarly to the prior studies [2], [13], [21], [45]. With the GPU TEE, a user application on the CPU side runs within an isolated environment provided by CPU TEEs, and its GPU kernels run on the extended TEE on each GPU. The remote attestation checks the TEEs on CPU and GPUs along with the application binary.

To protect against attackers violating the trusted GPU context, the security monitor in the CPU-side TEEs verifies all page table entries related to the GPU context. This prevents the malicious operating system from modifying the page table entries used in MMU and IOMMU and accessing the GPU memory directly via malicious mapping. For GPUs, the trusted command processor provides memory isolation similar to the previous work and commercial GPU [8], [45]. From these security guarantees, when a GPU needs to request an IOMMU translation result, The GPU TEE support must ensure that all translation results are correct. Additionally, our target multi-GPU system leverages a unified memory, allowing allocated memory to be accessed by any CPU or GPU in the system.

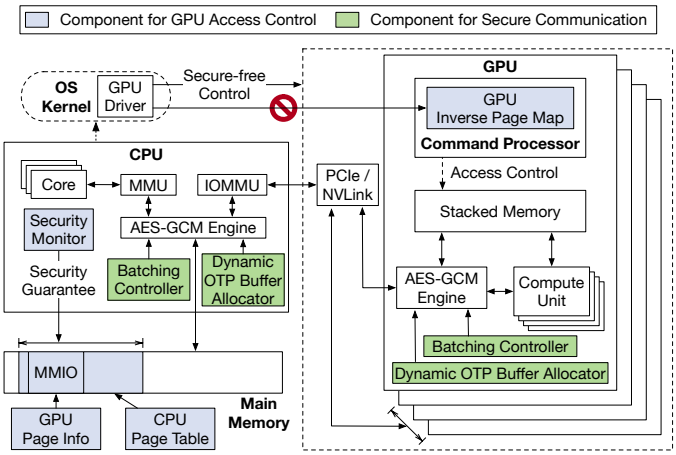


Fig. 17: The overview of our proposed architecture. We assume that access control mechanisms for CPU TEE, and GPU TEE are provided similarly with previous studies [9], [13], [21], [45].

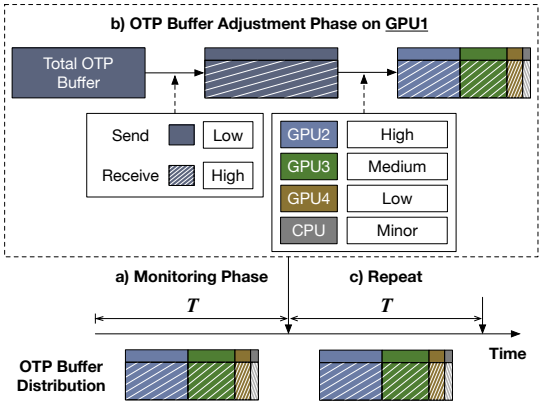


Fig. 18: Pad allocation algorithm in *Dynamic*. Every T interval, the number of pads is modified based on the monitored information of remote requests.

We assume that CPU TEEs can provide a memory protection mechanism over the entire CPU memory region, as proposed in recent works [13], [37].

The CPU and GPUs exchange a key during the system boot as discussed in the prior studies [34], [35], [36]. This exchanged key is used for authenticated encryption/decryption for CPU-GPU and GPU-GPU communications.

B. Dynamic OTP Buffer Management

Motivated by Section III-B, we introduce a dynamic OTP buffer management scheme (*Dynamic*) to reduce the performance overhead caused by OTP management. The *Private* mechanism allocates a fixed number of table entries for send/receive directions and distributes them evenly across each processor. On the other hand, the proposed *Dynamic* mechanism allocates more OTP buffer entries in the direction (send or receive) with more requests. Moreover, within each direction, the *Dynamic* mechanism adjusts the number of OTP buffer entries allocated for each GPU based on the communication patterns.

Variables	Description
$TotalOTPBuffer$	The number of OTP buffer entries in a processor
$SReq_i$ $RReq_i$	The number of send and receive requests in the i-th interval.
$SReq^m_{n\,i}$, $RReq^m_{n\,i}$	The number of send and receive requests in the i-th interval from processor n to processor m.
S_i	The weight of send direction in the i-th interval.
$S^m_{n\,i}$, $R^m_{n\,i}$	The weight of send/receive direction in the i-th interval from processor n to processor m.
$SPad_i$, $RPad_i$	The number of pads for send/receive requests in the i-th interval.
$SPad^m_{n\,i}$, $RPad^m_{n\,i}$	The number of pads from processor n to processor m within send/receive direction in the i-th interval
α	The rate that the history is forgotten in adjusting OTP size for each send/receive direction.
β	The rate that the history is forgotten in adjusting OTP size for each destination processor.

TABLE II: Variables Description

Figure 18 provides an overview of the *Dynamic* mechanism. Our *Dynamic* mechanism consists of two main phases, which are repeated at each pre-defined interval (T) during the GPU kernel execution: (1) **Monitoring phase** to observe and record communication patterns, and (2) **OTP buffer adjustment phase** to adjust OTP buffer entries based on the communication patterns identified during the monitoring phase.

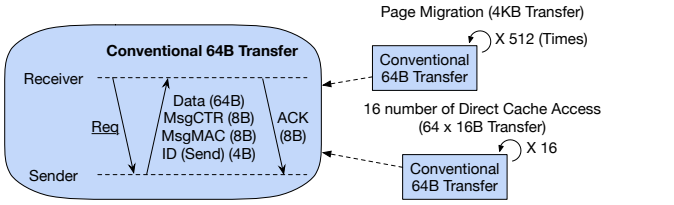
Monitoring phase: In this phase, each GPU tracks the number of communications to identify trends in two key aspects : (1) the send/receive direction and (2) communicated processors (CPU or GPUs) from each GPU. Based on this data, we calculate a weighted value representing the overall communication trend during execution, which is then utilized in the OTP buffer adjustment phase. To compute this value, we adopt the exponentially weighted moving average (EWMA) method [20] which is a statistical technique for analyzing time series data.

OTP buffer adjustment phase: When the initial GPU kernel is launched, *Dynamic* mechanism initially allocates an equal number of OTP buffer entries for each send/receive direction and each processor, similar to the *Private* mechanism. However, as the application runs, the dynamic OTP allocator adjusts the allocation of OTP buffer entries based on the communication patterns monitored at each interval (T).

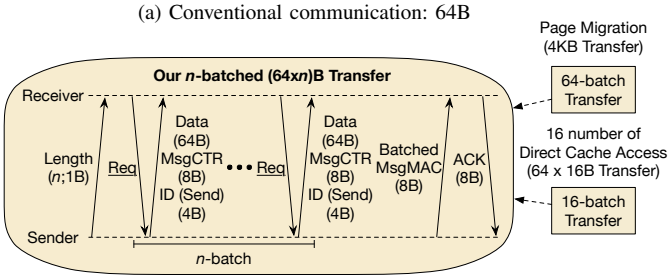
$$S_{i+1} = (1-\alpha) \times S_i + \alpha \times \left(\frac{SReq_i}{SReq_i + RReq_i}\right) \tag{1}$$

We define the variable S_{i+1} to represent the send direction weight as shown in Table II. This variable represents the weighted ratio of send direction compared to receive direction at the $(i+1)$-th interval based on the identified communication patterns. S_{i+1} is calculated by Formula 1, where $SReq_i$ and $RReq_i$ denote the total number of remote requests in the send and receive directions at the i-th interval, respectively. The variable α is used to determine the pace for updating the weighted ratio, where a larger α means placing more weight on the currently measured values.

$$\begin{aligned} SPad_{i+1} &= TotalOTPBuffer \times S_{i+1} \\ RPad_{i+1} &= TotalOTPBuffer - SPad_{i+1} \end{aligned} \tag{2}$$



(a) Conventional communication: 64B



(b) Our n-batched communication: (64×n)B

Fig. 19: Our batching-based communication mechanism compared to the conventional security metadata transfer.

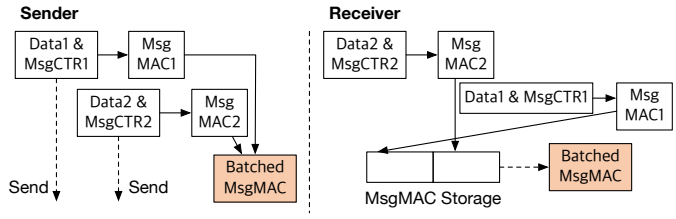


Fig. 20: Our batched MsgMAC generation. We utilize the MsgMAC storage to support out-of-order requests.

Utilizing the ratio S_{i+1} obtained from Formula 1, the allocation of OTP buffer entries for the send and receive directions is determined by Formula 2. In Formula 2, $TotalOTPBuffer$ represents the total number of available OTP buffer entries while $SPad_{i+1}$ and $RPad_{i+1}$ indicate the total number of OTP buffer entries to be allocated for the send and receive direction at the $(i+1)$-th interval, respectively.

$$\begin{aligned} S^m_{n\,i+1} &= (1-\beta)\times S^m_{n\,i} + \beta \times \left(\frac{SReq^m_{n\,i}}{SReq_i}\right) \\ R^m_{n\,i+1} &= (1-\beta)\times R^m_{n\,i} + \beta \times \left(\frac{RReq^m_{n\,i}}{RReq_i}\right) \end{aligned} \tag{3}$$

$$\begin{aligned} SPad^m_{n\,i+1} &= (SPad_{i+1}) \times S^m_{n\,i+1} \\ RPad^m_{n\,i+1} &= (RPad_{i+1}) \times R^m_{n\,i+1} \end{aligned} \tag{4}$$

Similar to Formula 1 and Formula 2, the number of OTP buffer entries for each GPU can be calculated by the EWMA method as shown in Formula 3 and Formula 4. The β value, similar to α, represents the weight given to the GPU requests measured in the i-th interval. Based on experiments, we set α to 0.9 and β to 0.5.

C. Security Metadata Batching

To leverage the burstiness in communication characteristics discussed in Section III, we propose a metadata batching technique to reduce performance degradation caused by transferring security metadata. Figure 19 illustrates our proposed security metadata batching technique. The proposed batching technique exploits the bursty communication patterns observed in sender-receiver pairs within a short period. Rather than transmitting security metadata for each cacheline-granular data transfer, our method aggregates and sends security metadata in a coarse-grained manner.

Figure 19 (a) and (b) compare the transmission of security metadata between the conventional and batched methods. The main difference lies in the frequency of security metadata transmissions, specifically MsgMAC and ACK. In the conventional approach (Figure 19 (a)), MsgMAC and ACK are sent for every 64B data transfer. In contrast, our batched approach (Figure 19 (b)) sends a coarse-grained MsgMAC to the receiver and receives a single ACK for n batches of 64B data transfers. For example, during a 4KB data transfer such as page migration, the conventional method sends 512 transmissions of MsgMAC, MsgCTR, sender ID, and ACK. However, as shown in Figure 19 (b), our proposed technique generates MsgMAC for each page and only a single ACK per page. For direct cache access transfers of 64B with n=16, our method sends a single MsgMAC and ACK per batch. Our mechanism adds a 1B batch size (length; n) information to the first request of each batch to inform the receiver. In this study, we set the batch size to 16, based on the result discussed in Section III-B.

$$\begin{aligned} &Batched\_MsgMAC = Concat(MsgMAC_1,\ \ldots\ ,\ MsgMAC_n) \\ &\textbf{when}\ MsgMAC_i = Generate\_MAC(Data_i,\ MsgCTR_i) \end{aligned} \tag{5}$$

Figure 20 illustrates the generation and computation processes for our coarse-grained Batched_MsgMAC, which is calculated from concatenated MsgMACs derived from the data and MsgCTR. To handle out-of-order data transmission, each calculated MsgMAC within the batch is stored in the MsgMAC storage. Once all requests in the batch are received, the receiver computes the Batched_MsgMAC in order, validates the integrity, and sends it back to the sender for replay attack protection.

It is important to note that we adopt a lazy integrity verification scheme [40], [41], [47]. In the security metadata batching technique, necessary metadata for data decryption, such as sender ID and MsgCTR, are transmitted with every 64B data transfer, similar to conventional methods. Although the MsgMAC verification may be delayed, the receiver can continue execution after data decryption. Therefore, despite potential delays in MsgMAC verification, our mechanism ensures the same security guarantee compared to conventional methods, minimizing performance degradation and reducing the bandwidth required for transmitting security metadata.

D. Hardware Overhead

Our dynamic OTP buffer management technique requires 64-bit counters for run-time monitoring of inter-processor communication patterns. We use a 4-GPU system as our

GPU Core Configuration	
System Overview	64 CUs per GPU, 4 GPUs in a system
Shader Core	1.0 GHz, 64 work items per wavefront
Cache and Memory Configuration	
L1 Vector Cache	16KB, 4-way, LRU
L1 Inst Cache	32KB, 4-way, LRU
L1 Scalar Cache	16KB, 4-way, LRU
Shared L2 Cache	2MB, 16-way, LRU
DRAM	HBM, 512GB/s
AES-GCM Latency	40 cycles [1], [49], [50]
Interconnect Configuration	
CPU-GPU	PCIe-v4 bus, 32 GB/s
GPU-GPU	Similar with NVLink2, 50GB/s
Hyperparameter Configuration	
α	0.9
β	0.5
T	1000

TABLE III: Configuration of simulated GPU system.

Category	Benchmark Suite	Workload (Abbr.)
High RPKI	AMD APP SDK	matrixtranspose (`mt`)
	DNNMark	relu (`relu`)
	Hetero-Mark	pagerank (`pr`)
	Polybench	syr2k (`syr2k`)
	SHOC	spmv (`spmv`)
Medium RPKI	AMD APP SDK	simpleconvolution (`sc`), matrixmultiplication (`mm`)
	Polybench	atax (`atax`), bicg (`bicg`), gesummv (`ges`), mvt (`mvt`)
	SHOC	stencil2d (`st`), fft (`fft`)
	Hetero-Mark	kmeans (`km`)
Low RPKI	AMD APP SDK	floydwarshall (`floyd`)
	Hetero-Mark	aes (`aes`), fir (`fir`)

TABLE IV: Evaluated Benchmarks

baseline, where each GPU requires 512 bits (4 (CPU and 3 GPUs) $\times$ 64 bits $\times$2 (send and receive)) for counters to track communication patterns. As shown in Table I, the required on-chip storage for 4 GPUs is from 2.50KB to 40.06KB (0.63KB to 10.02KB per each GPU) since an OTP buffer entry consists of a valid bit (1 bit), an encryption pad (512 bits), an authentication pad (64 bits), and a counter (64 bits).

Furthermore, our security metadata batching mechanism has the on-chip MsgMAC storage at the receiver side of each processor-processor pair. Since each Batched_MsgMAC can be generated by aggregating either 16 or 64 MsgMACs, max(16, 64) $\times$4 (CPU, and 3 GPUs) $\times$ 8B (MsgMAC) = 2KB on-chip memory is necessary per each GPU.

V. Evaluation

A. Methodology

Simulator: We use MGPUSim [43] to evaluate the performance of our proposed architecture with a 4-GPU system. Table III shows the simulated GPU configuration which models AMD R9 Nano GPU [3], [43]. Each GPU is equipped with HBM. GPUs in the system are connected with CPUs via PCIe bus and communicate with each other through high-speed interconnects, such as AMD Infinity fabric [5] or NVLink [14]. We extended MGPUSim to model secure inter-processor communication techniques including our proposed schemes. We use 40-cycle AES-GCM latency for authenticated encryption and decryption, similar to prior studies [1], [49], [50]. For the page migration policy, we adopt an access counter-based page migration policy, similar to the approach used in NVIDIA Volta GPUs [15].

Workloads: Table IV lists the evaluated workloads. We selected workloads with diverse communication traffic and different ratios of page migration to direct block access. These workloads are provided from the assorted GPU benchmark suites including Polybench [17], AMD APP SDK [4], Hetero-Mark [44], SHOC [10], and DNNMark [12]. We categorized the evaluated workloads into three groups based on their Remote requests Per Kilo Instructions (RPKI): high RPKI (greater than 1000), medium RPKI (less than 1000 but greater than 100), and low RPKI (less than 100).

B. Performance Improvement

To evaluate the performance improvement by the proposed techniques, we compare our techniques to the prior private and cached schemes in a 4-GPU system. Figure 21 shows the execution times for different configurations: The baseline *Private (OTP 4x)* mechanism with 32 OTP buffers per GPU and 128 OTP buffers (*Private (OTP 16x)*), *Cached (OTP 4x)* mechanism, and our proposed methods (*Dynamic (OTP 4x)*, *Batching (OTP 4x)*). Each result is normalized to a vanilla 4-GPU system without secure communication. On average, the *Private (OTP 4x)* and *Cached (OTP 4x)* exhibit performance degradation of 19.5% and 16.3%, respectively. With the same number of OTPs, applying *Dynamic (OTP 4x)* technique reduces the degradation to 14.7% on average. Increasing OTPs to 16x (*Private (OTP 16x)*) reduces the degradation to 14.0% with 4x area overheads than the OTP 4x configurations. Combining *Dynamic* with our metadata batching technique (represented as *Batching*) can further reduce the average performance degradation to 7.9%. In summary, when compared to the *Private (OTP 4x)* and *Cached (OTP 4x)* schemes, the application of *Dynamic* and *Batching* mechanisms results in average performance improvements of 11.6% and 8.4%, respectively.

For the workloads (`aes`, `mm`, `mt`, `pr`, `relu`, `spmv`, and `syr2k`), performance degradations range from 26.8% to 49.0% in the *Private (OTP 4x)*. Performance overheads of these workloads can be reduced ranging from 20.1% to 44.6% with *Dynamic* technique. When combining *Dynamic* with the security metadata batching technique (*Batching*) as well, the performance degradations of these workloads further reduce to 7.3% to 26.3%. While *Dynamic* mechanism exhibit lower performance improvement for some applications (`fir` and

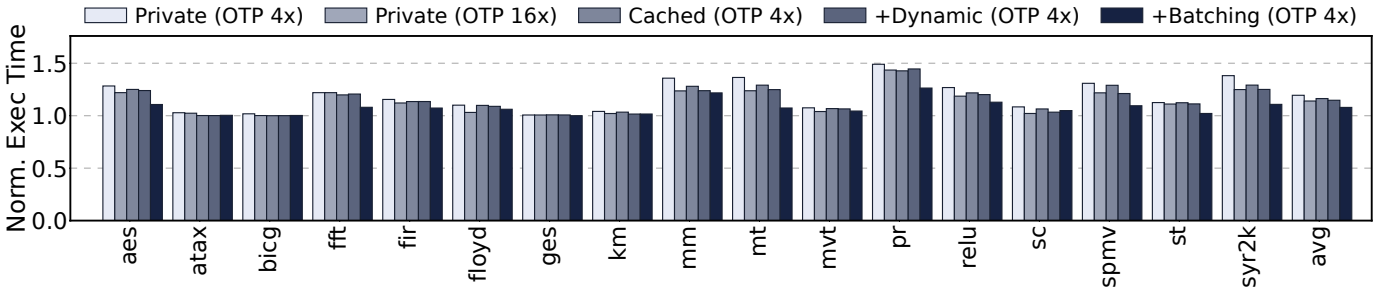


Fig. 21: Execution times with 4 GPUs for *Private (OTP 4x)*, *Private (OTP 16x)*, *Cached (OTP 4x)*, *+Dynamic (OTP 4x)*, and *+Batching (OTP 4x)* normalized to the unsecure system.

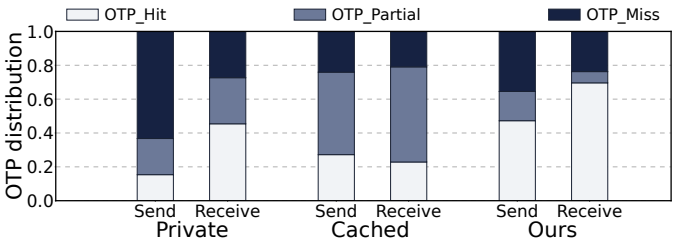


Fig. 22: The distribution of one-time pad (OTP) latency within authenticated encryption/decryption on the 4-GPU system (*OTP 4x*).

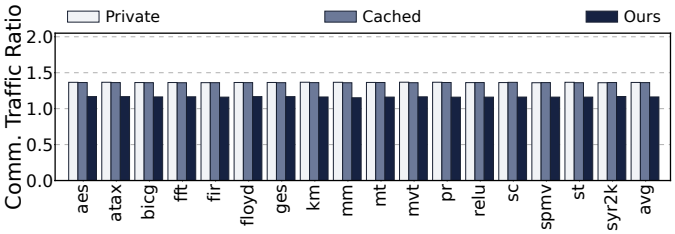


Fig. 23: Ratio of communication traffic for *Private*, *Cached* and *Ours*, compared to the unsecure multi-GPU system. All schemes use *OTP 4x*.

`floyd`), we observe the overall performance degradation can be significantly reduced compared to the *Private (OTP 4x)* by the performance gain from the batching technique. Furthermore, even when compared to using emphPrivate (OTP 16x), our mechanism can achieve less performance degradation. While increasing the number of OTP buffers can effectively reduce the performance overhead by secure communication, it cannot address the performance degradation caused by additional security metadata.

OTP distribution: Figure 22 shows the average distribution of *OTP_Hit*, *OTP_Partial*, and *OTP_Miss* rates across all benchmarks for three different mechanisms (*Private*, *Cached* and *Ours*). *Ours* represents our secure and efficient multi-GPU system with both *Dynamic* and *Batching* mechanisms. On average, *Private* mechanism can hide 36.8%, 72.6% of the authenticated encryption and authenticated decryption respectively, either fully or partially, while the *Cached* can hide 75.9%, 79.1% of the authenticated encryption and authenticated decryption. As shown in Figure 22, since *Ours* can allocate more OTP buffers to the required processor based on the runtime-changing communication patterns, *Ours* can hide 64.6%, 76.2% of the authenticated encryption and authenticated decryption, respectively.

For the *OTP_Hit*, where authenticated encryption/decryption can be fully hidden, *Ours* can increase on average 31.9%, 20.1% for the send direction and 24.2%, 46.7% for the receive direction compared to the *Private (OTP 4x)* and the *Cached (OTP 4x)*. These significant increases in the ratios of *OTP_Hit* reduce performance degradation caused by the authenticated encryption/decryption, making *Ours* exhibit less performance degradation compared to the *Private* and *Cached* schemes.

Communication traffic: We observed that the amount of additional traffic caused by security metadata can be reduced by using the proposed security metadata batching technique during page migration and cacheline granularity communications. Figure 23 presents the normalized communication traffic with *OTP 4x* when applying our *Dynamic* and *Batching* mechanism compared to the *Private)*. On average, we achieved a 20.2% and 20.0% reduction in communication traffic through interconnects compared to the *Private* and *Cached* mechanisms, respectively. For workloads that experience significant performance degradation due to security metadata, such as `fft`, `mt`, `pr`, `spmv`, and `syr2k`, applying security metadata batching can reduce the performance loss by an additional 11.7% to 18.2%, compared to using the *Dynamic* technique only.

C. Sensitivity to AES-GCM Latency

To see the performance impact of AES-GCM latency, we measure the performance of *Private*, *Cached*, and our mechanism while varying the AES-GCM latency from 10 cycles to 40 cycles. Figure 26 shows how the performance varies as the AES-GCM latency changes. We observe that the performance changes in the *Private*, *Cached*, and our scheme are not significant. When changing from 40 cycles to 10 cycles, the average performance degradation for the *Private* mechanism changes from 19.5% to 17.3%, for the *Cached* mechanism it reduces from 16.3% to 13.6%, and for our method, it shifted from 7.9% to 5.6%.

As discussed in Section III-A, the overhead of secure communication consists of the performance overhead due

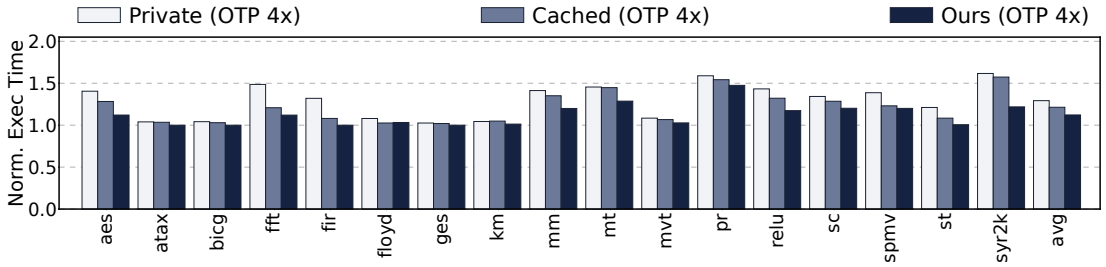


Fig. 24: Execution times for *Private*, *Cached* and *Ours* with 8 GPUs normalized to the unsecure 8-GPU system.

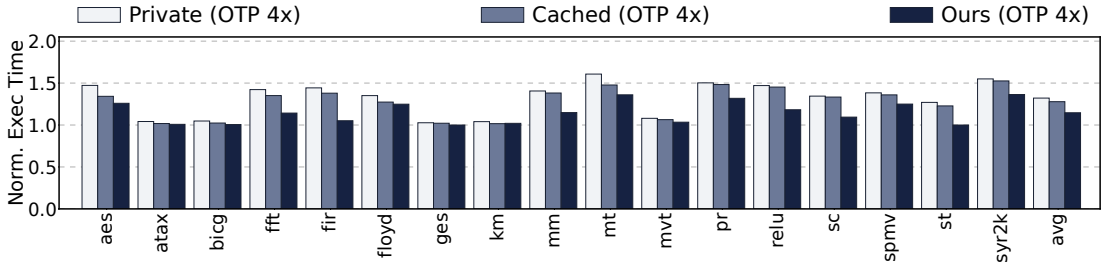


Fig. 25: Execution times for *Private*, *Cached* and *Ours* with 16 GPUs normalized to the unsecure 16-GPU system.

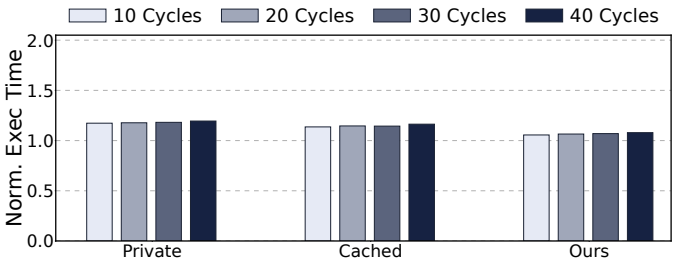


Fig. 26: Execution times under various AES encryption latency scenarios normalized to the unsecure multi-GPU system.

to authenticated encryption and the additional bandwidth consumed by transferring security metadata, such as sender ID, MsgMAC, and MsgCTR. Therefore, even with a reduction in AES-GCM latency, significant performance improvement is unattainable due to the persistent issue of additional bandwidth requirements for accessing security metadata.

D. Sensitivity to the Number of GPUs

We compare our mechanism with *Private* and *Cached* in 8-GPU and 16-GPU systems. We use the same input size as in the 4-GPU system experiments and use 64 OTP buffers per GPU and 128 OTP buffers per GPU, respectively. Figures 24 and 25 present the execution times for 8-GPU and 16-GPU systems, comparing the *Private* with our mechanism that integrates the *Dynamic* technique and security metadata batching optimization (*Batching*), normalized to the unsecured 8-GPU and 16-GPU systems. On average, the *Private* mechanism showed a 29.3% performance degradation for 8 GPUs and 32.1% for 16 GPUs. In addition, the *Cached* mechanism showed a 21.4% performance degradation for 8 GPUs and 27.8% for 16 GPUs. Our techniques resulted in a 12.2% degradation for 8 GPUs and 14.6% for 16 GPUs. In summary, for the 8-GPU system, our technique can reduce the average performance overhead by 17.1% and 9.2% compared to the *Private* and *Cached* schemes, respectively. In a 16-GPU system, the corresponding performance reductions are 17.5% and 13.2% compared to the *Private* and *Cached* schemes.

As the number of GPUs increases, the cost of securing communications with OTPs increases when the buffer size is limited. Our techniques can reduce the cost of communication security significantly by maximizing the utilization of OTP buffers and by using the network bandwidth effectively.

VI. Related Work

Accelerator Security: To provide trusted accelerator execution, protection mechanisms, including isolated execution, memory protection, and mitigation of side-channel attacks need to be supported. To support access control and memory protection, many studies have proposed trusted execution environments (TEEs) with hardware and software techniques [18], [19], [21], [27], [28], [30], [45]. HIX and Graviton focused on a secure access control mechanism by isolating GPU-related components from privileged software such as operating systems [21], [45]. Common Counters [30], PSSM [50], and SHM [49] suggested hardware-based memory protection techniques for GPU TEEs by utilizing unique features of GPU applications. TNPU, Tunable NPU, GuardNN, and MGX propose the NPU TEEs by leveraging the regular and chunk-level access patterns of NPUs [18], [19], [27], [28]. To protect against side-channel attacks exploiting the coalescing mechanism of GPUs, BCoal, and RCoal camouflage the degree of coalescing to hide the number of coalesced requests [22], [23]. While prior studies deal with secure access, memory protection, and access control in a single GPU or NPU environment, this paper proposes the communication protection technique across multiple GPUs.

Trusted Multi-Processors System: Since multi-processors add an extra protection dimension by shared memory supports across multiple processors, there have been several studies about secure communication techniques to support the secure cache-coherent communication [35], [36]. Rogers et al. [35] proposed a two-level encryption and authentication scheme consisting of two separate techniques for protecting distributed shared memory multiprocessors: one for CPU memory protection and the other for secure communication. In particular, they used AES-GCM (Galois Counter Mode) to hide the latency from counter-mode encryption and MAC verification. The subsequent work [36] proposed a single-level memory encryption and authentication for protecting both processor-memory and processor-processor communication. The single-level mechanism can hide the latency by performing decryption and authentication only once on the requester side, which reduces the critical path of remote data requests. SDSM uses Trusted Coherence Manager (TCM) to manage counters for each process, which prevents the need to maintain the counter cache [42].

VII. Conclusion

This paper investigated the protection mechanism against physical attacks on communication channels in multi-GPU systems with fine-grained memory sharing. It showed that the prior CPU-oriented inter-processor communication protection does not fully consider the bursty and dynamic characteristics of GPU communication. To mitigate the performance and area cost of the protection mechanism, this study proposed a new secure metadata batching to reduce the additional traffic by increasing the secure granularity, and dynamic OTP buffer management technique which adjusts the buffer capacity allocation based on the traffic patterns. Our evaluation showed that when the OTP buffer size is limited to 128 per GPU in an 16-GPU system, the proposed scheme can improve the performance by 17.5% and 13.2% on average from the prior private and cached schemes, respectively.

VIII. Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) funded by the Ministry of Science and ICT, Korea (IITP2017-0-00466 SW StarLab) and National Research Foundation of Korea (NRF-2022R1A2B5B01002133). This work was also partly supported by Samsung Electronics Co., Ltd. (IO201209-07864-01).

References

[1] R. Abdullah, H. Zhou, and A. Awad, "Plutus: Bandwidth-efficient memory security for GPUs," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2023.

[2] S. Aga and S. Narayanasamy, "Invisimem: Smart memory defenses for memory bus side channel," *ACM SIGARCH Computer Architecture News*, 2017.

[3] AMD, "AMD Radeon R9 series gaming graphics cards with high-bandwidth memory," 2015.

[4] AMD, "AMD APP SDK 3.0 getting started," 2017.

[5] AMD, "AMD infinity architecture: The foundation of the modern datacenter," 2017.

[6] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[7] T. Baruah, Y. Sun, A. T. Dinçer, S. A. Mojumder, J. L. Abellán, Y. Ukidave, A. Joshi, N. Rubin, J. Kim, and D. Kaeli, "Griffin: Hardware-software support for efficient page migration in multi-gpu systems," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[8] J. Choquette, "Nvidia Hopper H100 GPU: Scaling performance," in *IEEE Micro*, 2023.

[9] V. Costan and S. Devadas, "Intel SGX explained," in *IACR Cryptology ePrint Archive*, 2016.

[10] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Workshop on General Purpose Computation on Graphics Processing Units (GPGPU)*, 2010.

[11] S. Deng, W. Xiong, and J. Szefer, "A benchmark suite for evaluating caches' vulnerability to timing attacks," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[12] S. Dong and D. Kaeli, "DNNMark: A deep neural network benchmark suite for GPUs," in *Workshop on General Purpose Computation on Graphics Processing Units (GPGPU)*, 2017.

[13] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the PENGLAI enclave," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

[14] D. Foley and J. Danskin, "Ultra-performance Pascal GPU and NVLink interconnect," in *IEEE Micro*, 2017.

[15] D. Ganguly, Z. Zhang, J. Yang, and R. Melhem, "Adaptive page migration for irregular data-intensive applications under GPU memory oversubscription," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2020.

[16] N. A. Gawande, J. A. Daily, C. Siegel, N. R. Tallent, and A. Vishnu, "Scaling deep learning workloads: Nvidia DGX-1/Pascal and Intel Knights Landing," in *Future Generation Computer Systems (FGCS)*, 2020.

[17] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *Innovative Parallel Computing (InPar)*, 2012.

[18] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "GuardNN: Secure accelerator architecture for privacy-preserving deep learning," in *Design Automation Conference (DAC)*, 2022.

[19] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "MGX: Near-zero overhead memory protection for data-intensive accelerators," in *International Symposium on Computer Architecture (ISCA)*, 2022.

[20] J. S. Hunter, "The exponentially weighted moving average," in *Journal of quality technology*, 1986.

[21] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity GPUs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[22] G. Kadam, D. Zhang, and A. Jog, "RCoal: Mitigating GPU timing attack via subwarp-based randomized coalescing techniques," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[23] G. Kadam, D. Zhang, and A. Jog, "BCoal: Bucketing-based memory coalescing for efficient and secure GPUs," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[24] G. Kim, M. Lee, J. Jeong, and J. Kim, "Multi-GPU system design with memory networks," in *International Symposium on Microarchitecture (MICRO)*, 2014.

[25] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *Communications of the ACM*, 2020.

[26] M. Lee, M. Ahn, and E. J. Kim, "I2SEMS: Interconnects-independent security enhanced shared memory multiprocessor systems," in *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2007.

[27] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "TNPU: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[28] S. Lee, S. Na, J. Kim, J. Park, and J. Huh, "Tunable memory protection for secure neural processing units," in *International Conference on Computer Design (ICCD)*, 2022.

[29] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," in *Communications of the ACM*, 2020.

[30] S. Na, S. Lee, Y. Kim, J. Park, and J. Huh, "Common Counters: Compressed encryption counters for secure GPU memory," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[31] NVIDIA, "DGX-2/2H system," 2020.

[32] NVIDIA, "CUDA C++ programming guide," 2022.

[33] X. Ren, D. Lustig, E. Bolotin, A. Jaleel, O. Villa, and D. Nellans, "Hmg: Extending cache coherence protocols across modern hierarchical multi-GPU systems," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[34] Y. Ro, S. Jin, J. Huh, and J. Kim, "Ghost routing to enable oblivious computation on memory-centric networks," in *International Symposium on Computer Architecture (ISCA)*, 2021.

[35] B. Rogers, M. Prvulovic, and Y. Solihin, "Efficient data protection for distributed shared memory multiprocessors," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[36] B. Rogers, C. Yan, S. Chhabra, M. Prvulovic, and Y. Solihin, "Single-level integrity and confidentiality protection for distributed shared memory multiprocessors," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[37] G. Saileshwar, P. Nair, P. Ramrakhyani, W. Elsasser, J. Joao, and M. Qureshi, "Morphable Counters: Enabling compact integrity trees for low-overhead secure memories," in *International Symposium on Microarchitecture (MICRO)*, 2018.

[38] N. Sakharnykh, "Unified memory on Pascal and Volta," in *GPU Technology Conference (GTC)*, 2017.

[39] N. Sakharnykh, "Everything you need to know about unified memory," *GPU Technology Conference (GTC)*, 2018.

[40] W. Shi, H.-H. Lee, M. Ghosh, and C. Lu, "Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems," in *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2004.

[41] W. Shi, H.-H. S. Lee, C. Lu, and M. Ghosh, "Towards the issues in architectural support for protection of software execution," in *ACM SIGARCH Computer Architecture News*, 2005.

[42] O. Shwartz and Y. Birk, "SDSM: Fast and scalable security support for directory-based distributed shared memory," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

[43] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao *et al.*, "MGPUSim: Enabling multi-GPU performance modeling and optimization," in *International Symposium on Computer Architecture (ISCA)*, 2019.

[44] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. McCardwell, A. Villegas, and D. Kaeli, "Hetero-mark, a benchmark suite for CPU-GPU collaborative computing," in *International Symposium on Workload Characterization (IISWC)*, 2016.

[45] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[46] D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, and C. Rossow, "Osiris: Automated discovery of microarchitectural side channels," in *USENIX Security Symposium (USENIX Security)*, 2021.

[47] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *ACM SIGARCH Computer Architecture News*, 2006.

[48] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, "Combining HW/SW mechanisms to improve NUMA performance of multi-GPU systems," in *International Symposium on Microarchitecture (MICRO)*, 2018.

[49] S. Yuan, A. Awad, A. W. B. Yudha, Y. Solihin, and H. Zhou, "Adaptive security support for heterogeneous memory on GPUs," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[50] S. Yuan, Y. Solihin, and H. Zhou, "PSSM: Achieving secure memory for gpus with partitioned and sectored security metadata," in *International Conference on Supercomputing (ICS)*, 2021.

[51] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, and D. Meng, "Enabling rack-scale confidential computing using heterogeneous trusted execution environment," in *Symposium on Security and Privacy (S&P)*, 2020.