Tunable Memory Protection for Secure Neural Processing Units

Sunho Lee, Seonjin Na, Jungwoo Kim, Jongse Park and Jaehyuk Huh
School of Computing, KAIST
{*myshlee417*, *sjna*, *jwkim*, *jspark*}@*casys.kaist.ac.kr*, *jhhuh@kaist.ac.kr*

Abstract—**One of the key security supports for neural processing units (NPUs) is the hardware-based memory protection to provide confidentiality and integrity of NPU data. However, adopting the memory encryption and integrity protection techniques developed for CPUs do not fully utilize the NPU characteristics, incurring a significant performance degradation. To address the performance challenges, this paper proposes new improvements of memory protection for NPUs based on the unique property of NPU computation. The design first proposes a context-based memory protection which imposes the hardware memory protection only for the critical memory region of NPUs. Second, it allows adjusting the counter granularity for NPU memory to reduce the overheads of common counter-mode encryption. In addition, it exploits the read-only property of machine learning parameters, and adds a trusted communication channel between the CPU and NPU. Our evaluation with a simulated NPU shows that the performance overhead of memory protection for NPUs can be significantly reduced from the state-of-the-art CPU-oriented design, improving the performance by 13.5%.**

I. Introduction

As neural processing units (NPUs) integrated in CPUs process more and more mission-critical tasks in edge devices, protecting the data used by NPUs has become important. Considering the environments where edge devices are used, the memory-resident data of NPUs must be protected not only from compromised privileged software but also from direct physical attacks. For CPUs, hardware-based memory protection guarantees the confidentiality and integrity of memory-resident data even from physical attacks. For such hardware-based protection, counter-mode encryption is commonly used. Per-block counters guarantee the freshness of the data, and a counter tree is maintained to verify the counter values for their integrity.

However, simply adopting the mechanism designed for CPU applications does not provide optimized performance for NPUs. To overcome the limitation of applying the CPU-oriented memory encryption schemes to NPUs, this study proposes techniques to improve the performance of secure memory accesses. First, this study proposes *virtual integrity tree* supporting the context-based memory protection technique to secure only the critical memory region accessed by NPUs, instead of applying costly hardware protection to the entire physical memory. By protecting only the memory region used by the NPU context, its counter tree height is reduced significantly, shortening the latencies of handling hash cache misses.

Second, it investigates the performance overheads of counter-mode encryption for NPUs and proposes to support multi-granular counters to match the memory access patterns of NPUs. NPUs frequently access the memory at much larger granularities than a typical CPU cacheline size, as they bring the parameters and inputs from the memory to the internal Scratch Pad Memory (SPM) at a large chunk unit. Instead of fixing the counter granularity to a cacheline unit, our design allows each machine learning task to use an appropriate granularity.

This study further improves the efficiency of caching counters by exploiting read-only pages containing model parameters, as proposed by the prior work for GPU [1]. In addition, a secure communication buffer is added between CPU and NPU to avoid costly encryption. Our evaluation with a simulated NPU shows that the overhead of memory protection for NPUs can be significantly reduced from the baseline design using the start-of-art CPU-oriented techniques, improving the performance by 13.5%.

II. Background

A. Hardware-based Security

Trusted execution environment (TEE): TEE provides an isolated execution environment from the operating system. In Intel SGX, TEE is combined with hardware-based memory protection to defend against physical attacks on the TEE memory. Recently, TEE has been extended to support trusted execution in GPU and NPU [2]–[4].

Memory protection: Hardware memory protection uses encryption and integrity validation for memory-resident data. When a data block leaves the trusted processor chip, its content is encrypted. To bring the data from the memory, it needs to be decrypted and its integrity is validated. The counter mode encryption and integrity protection are commonly used for memory protection. A dedicated counter is allocated to each memory block (cacheline) and its value is increased when a memory block is updated. The encryption module generates a One-Time-Pad (OTP) using the address and allocated counter. A plaintext block is XORed with OTP, and the encrypted data is evicted to the off-chip memory. Additionally, MAC (Message Authentication Code) is written to the memory, to detect whether the value is modified when it is read later.

However, attackers can bypass MAC verification by replacing the memory contents with stale values used in the past. To prevent such replay attacks, a counter-based tree is commonly used [5]. The counter value of a cacheline is verified by recursively traversing toward the root, which does not leave the processor. For efficiency, the processor maintains two on-chip metadata caches called *counter cache* and *hash cache* to store counters and intermediate nodes of the counter tree in the on-chip area. The metadata in the counter cache or hash cache are already integrity-verified and do not incur additional validation costs. Therefore, counter and hash cache hits reduce the encryption and integrity protection latency significantly.

Memory protection for accelerators: There have recent studies to optimize memory protection techniques for GPUs and accelerators. For GPUs, Common Counter increases the utilization of a counter cache by merging multiple counters into one [1]. Since NPU workloads have even more regularity than GPU workloads, recent papers introduce software-based memory protection for NPU. Recent two studies, TNPU and MGX, set the granularity of counter into compile-level determined tensors or tiles [4], [6]. Unlike the two studies, this study relies on the hardware-based method for transparent support with the existing CPU techniques.

B. Threat Model and Baseline Architecture

Threat model: We target an edge-level NPU-integrated SoC (System-on-a-Chip) architecture that has CPU and NPU with shared memory. In this study, the trusted computing base (TCB) is the SoC chip itself and software executed in the trusted context of NPU. We assume that system software can be compromised by attackers and physical attacks are possible. We leave a side-channel attack, availability attack, and adversarial attack out of scope.

Baseline architecture: We assume CPU-side TEE support and TEE extensions for NPUs from the prior work [4]. We use the same NPU access control mechanism, and this study focuses on the hardware-based memory protection technique. We use page tables for TEEs, called *protected page table*, are isolated from the operating system [7]. Similar to the prior approach [7], untrusted and trusted parts of an NPU context use separate page tables, one in the untrusted memory, and the other in the trusted memory. Additionally, metadata for protected page table such as the base address of a trusted page table is stored in the hardware protected region. Only the security monitor and MMU can access it by comparing address ranges.

III. Motivation

A. Memory Protection for NPUs

This section presents the performance costs of the naïve memory protection with NPUs. More details of the simulation and workload setups are described in Section V.

Performance impact: Compared to the nonsecure configuration, the baseline secure design incurs a 21.5% increase of the execution time on average, requiring optimized techniques for NPUs.

Protecting part of memory: In the baseline design, the entire physical memory is protected by the counter-mode encryption using a single counter tree. As the depth of the counter tree increases, the latency of handling a hash cache miss can increase. An optimization is to protect only the NPU TEE part of the memory instead of covering the whole DRAM.

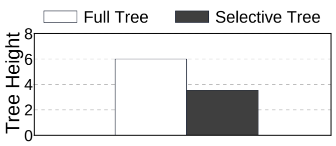

Fig. 1. Average counter tree heights: full tree vs selective tree.

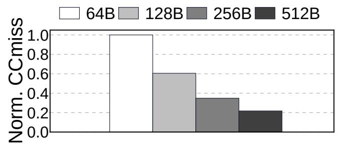

Fig. 2. Average counter cache miss rates from 64B to 512B chunks.

Figure 1 shows how much height can be reduced by the selective integrity tree. For 4GB off-chip memory with a 64-arity tree, the height of the full memory counter tree is $1 + \lceil \log_{64}(4GB/64B) \rceil = 6$. If a selective integrity tree for each NPU TEE context is used, the height of tree declines by 40.9%, reducing the latency for handling hash cache misses.

Counter granularity: The baseline uses a CPU-oriented design, which has a counter for each 64B unit of memory. However, for NPU, memory accesses occur at a chunk unit with software-managed scratchpad memory (SPM). The chunk size may vary by ML models with different layer types and algorithms. If a counter can be assigned for each chunk, instead of a cacheline, it can improve the efficiency of the counter cache. Figure 2 shows counter cache miss rates with various counter granularities. As the granularity increases, counter cache misses decrease significantly, and 512B granularity has only 21.8% of misses compared to 64B one.

IV. Architecture

A. Overall Architecture

Our architecture allows only the necessary part of the memory to be protected to minimize the cost. It maintains a small CPU protected memory similar to PRM (Processor Reserved Memory) in SGX. The CPU protected region is used for CPU TEEs and the metadata for NPU TEE contexts.

Figure 3 shows the overview of our architecture. The memory region used by NPUs can be scattered in the physical address space, but it is protected by the virtual integrity tree for the application. A counter region is allocated for each secure NPU context, and metadata for calculating the counter address, *counter_base*, is stored in the part of the context. Secure communication is supported with a 2KB SRAM buffer to transfer data between CPU and NPU. In our design, the protected page table entry has additional *read_only* and *granularity* bits. *read_only* is set if the corresponding page is read-only and *granularity* stores the best-fit granularity. In this paper, every *granularity* for an NPU TEE is fixed to a single value. Nevertheless, we maintain it in page table entries for the future extension for per-page granularity.

B. Virtual Integrity Tree

In prior memory protection, the entire physical memory is protected by the integrity tree [1], [5]. To mitigate the overheads of the integrity tree (counter tree), this study uses a selective counter tree called *virtual integrity tree* covering only memory regions used by an NPU TEE. A virtual integrity tree is created for each NPU TEE. The virtual integrity tree

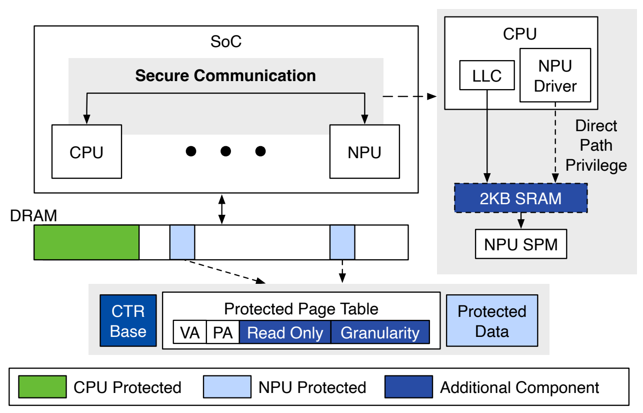

Fig. 3. Overview of proposed NPU memory protection schemes

of a TEE is based on the virtual address space, unlike the counter mapping based on physical addresses as used in the prior work for CPUs [8]. Constructing a tree based on a virtual address allows flexible allocation of the NPU TEE memory region while supporting a counter tree selectively covering only the NPU TEE memory region. Such a virtual integrity tree is possible in our design since the NPU TEE page table is isolated from the OS. PENGLAI proposed an integrity tree design covering only a TEE for CPU [9]. However, it requires mapping information for mounted tree nodes, as it is based on physical addresses.

Each NPU TEE has a *protected virtual address region* which is contiguous in its virtual memory space. For the region, the corresponding counters and internal nodes are created in a contiguous physical location for the TEE. Note that the physical pages for the NPU TEE memory can be allocated anywhere. Only the counter tree locations are in a contiguous region. The starting address of the counter region (*counter_base*) is recorded in the control structure of the NPU TEE. When the NPU TEE is scheduled and being executed in an NPU, the memory encryption engine must store *counter_base* in a dedicated register, called *counter_base_register*. For a memory access request, its virtual address is used to compute the address of the corresponding counter based on the starting address of the protected virtual address region, and *counter_base*. Once the counter address is computed, the counter cache is accessed. Since counters of an NPU TEE are allocated in contiguous physical memory, the address of the parent or child counter node is easily computed.

C. Multi-granular Counters

In this study, we decouple the unit of the counter from the physical memory interface. So, a counter is assigned to a chunk of memory instead of a cacheline. For example, if the chunk size is determined to be 512B for a machine learning application, a contiguous 512B is always moved together between the SPM and external memory.

Profiling-based chunk size selection: We assume that the best chunk size for a trained machine learning model is determined during the profiling phase. The user tries several different chunk sizes with the model and chooses the best-performing one. The best chunk size is delivered to the NPU driver during

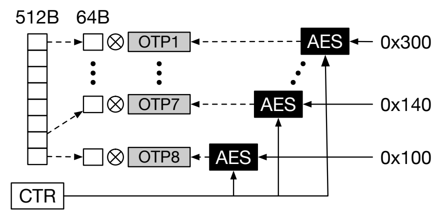

Fig. 4. Encryption for a multi-granular counter with 512B chunk size.

the initialization. Although this paper limits the scope of the profiling method to a simple trial-and-error method, improving the method will be our future work.

Multi-granular encryption: Figure 4 presents how per-chunk counter is used for encryption. Assuming the memory interface is 64B, 8 OTPs are generated to transfer 512B data. To generate an OTP for 64B, the counter (shared for the 512B chunk) and address of 64B data are used as seed, and thus each 64B has a different OTP.

D. Others

Read-only optimization: For read-only pages, we use a similar optimization proposed by the prior work [1]. The counters for an NPU TEE are reset during the context initialization. Therefore, if a page is read-only, data access to the page does not access the counter cache. Instead, it uses a fixed initial counter value of 1. To support it, the protected page table stores the read-only status and it is used when data moved into SPM.

Protected channel: The baseline communication channel between the CPU and NPU is the shared memory, encrypted by the software on both sides. However, since both CPU and NPU are in the same processor, our design establishes an 2KB SRAM buffer channel, called *channel buffer*, to be used for CPU-NPU communication without using the external DRAM. 2KB buffer is sufficient for intermediate channel as memory footprints of 30% layers in our workloads are smaller than 2KB and they show streaming characteristic. To protect the channel buffer from the OS, the control of channel buffer is given only to the NPU driver TEE. When a user TEE initializes an NPU TEE, it can request the allocation of the channel buffer between itself and the created NPU TEE.

V. Evaluation

A. Methodology

Simulation infrastructure: We simulate the proposed architecture using a cycle-accurate in-house simulator. It is an extended version of SCALE-Sim [10]. We augmented the necessary components for secure NPU, which constitute counter-mode encryption, Bonsai merkle tree with 64-arity [5].

Hardware configuration: We use an NPU configuration similar to the NVDLA architecture. It consists of a 1GHz 16 x 16 systolic array, a 192KB SPM with 8-channel 5GB/s memory bandwidth. Additionally, 512B and 2KB are allocated for counter and hash cache.

Benchmarks: As shown in Table I, We evaluate 14 benchmarks from MLperf and DeepBench. The column, *Selected Chunk*, shows the memory chunk size selected by the multi-granular counter technique, the same as optimal granularity.

TABLE I
EVALUATED BENCHMARK MODELS.

Model	Selected Chunk
Googlenet (goo), Mobilenet (mob)	256B, 256B
Yolo-tiny (yt), Alexnet (alex)	256B, 512B
FasterRCNN (rcnn), DeepFace (df)	128B, 128B
Resnet50 (res), MelodyExtractionDetection (med)	128B, 512B
Text-generation (tx), AlphaGoZero (agz)	512B, 512B
Sentimental-seqCNN (sent), DeepSpeech2 (ds2)	64B, 512B
Transformer (tf), NCF-recommendation (ncf)	64B, 256B

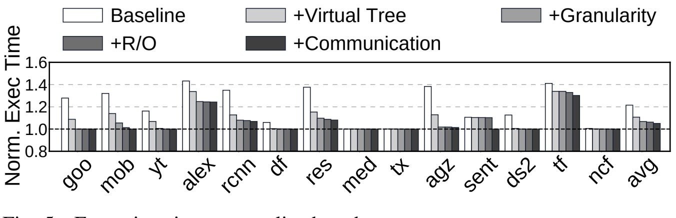

Fig. 5. Execution times normalized to the nonsecure run.

B. Performance Improvement

Figure 5 shows the execution time reduction as we incrementally augment the four optimization techniques from the baseline, which represents the NPU with naïvely adopted CPU-oriented memory protection. It is normalized to the execution time of the nonsecure version. The virtual tree optimization is the most influential factor causing an 8.9% reduction in execution time, followed by multi-granularity with 3.5%, read-only with 0.5%, and communication with 1.1% in series. With these four optimizations combined, the execution time reduces from 21.5% in the baseline secure NPU to 5.1%.

Figure 6 presents counter cache miss rates when the first three optimizations are applied, except for communication, which does not affect counter cache misses. On average, the combined techniques can reduce counter cache misses by 67.4%. The virtual tree can also reduce hash cache misses by 27.4%.

C. Scalability Study

We perform the scalability study by increasing the number of NPUs. The execution time is normalized by each baseline when one, two, and three NPUs are used. As the counter cache is evenly split and statically assigned to each NPU, it shows the similar performance to the case where the $\frac{1}{n}$-sized counter cache is used. Figure 7 shows the speedup compared to each baseline which applies the traditional integrity tree. As the number of NPU increases, our optimization techniques reduce the burden on counter and hash caches. On the other hand, more requests cause high traffic to a memory controller. Because of the balancing of the above two phenomena, the speedup ratio of 3-NPU is similar with a small improvement. It implies that our study creates more performance improvement enough to eliminate the overhead from traffic increment.

VI. CONCLUSION

This study proposed improvements of hardware-based memory protection for NPUs. It utilized the characteristics of neural computation in the integrated NPUs, reducing the

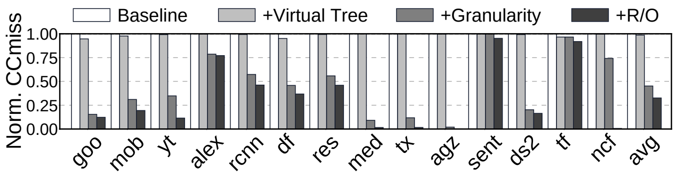

Fig. 6. Counter cache miss rates by applying optimizations.

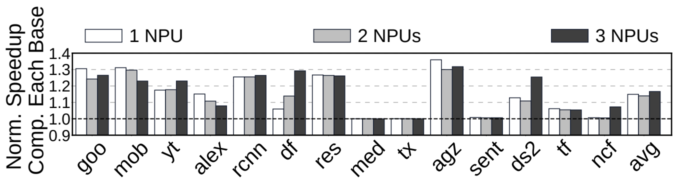

Fig. 7. Normalized speedups compared to each baseline when one, two, and three NPUs are used.

cost of counter cache misses and integrity validation. The experimental results showed that such memory protection can be supported for NPUs with minor overheads.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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.
[2] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.
[3] 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.
[4] 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.
[5] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and Bonsai merkle trees to make secure processors OS-and performance-friendly," in *International Symposium on Microarchitecture (MICRO)*, 2007.
[6] 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.
[7] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security Symposium (USENIX Security)*, 2016.
[8] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. prvulovic, "Improving cost, performance, and security of memory encryption and authentication," in *International Symposium on Computer Architecture (ISCA)*, 2006.
[9] 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.
[10] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.