Hypnos: Memory Efficient Homomorphic Processing Unit

Haoxuan Wang*†§, Yinghao Yang*†, Hang Lu*†§✉, Xiaowei Li*†§
*SKLP, Institute of Computing Technology, Chinese Academy of Sciences,
†University of Chinese Academy of Sciences,
§Zhongguancun Laboratory, Beijing, China
{wanghaoxuan23p, luhang, lxw}@ict.ac.cn, yinghao.y@foxmail.com

***Abstract*—Fully Homomorphic Encryption (FHE) introduces a novel paradigm in privacy-preserving computation, extending its applicability to various scenarios. However, Operating on encrypted data imposes significant computational challenges, particularly elevating data transmission and memory access demands. Consequently, developing an efficient system storage architecture becomes vital for FHE-specific architectures. Traditional FHE accelerators use a Host+ACC topology, often focusing on enhancing computational performance and efficient using of on-chip caches, with the assumption that very large volumes of encrypted data are already present in the accelerator's memory while neglecting the inefficiencies of the unavoidable PCIe bus. In this paper, we propose Hypnos—a memory-efficient homomorphic encryption processing unit. In Hypnos, we abstract operators from FHE schemes into commands suitable for memory-efficient processing units and combine them with a homomorphic encryption paged memory management system designed for memory access, significantly reducing the memory access and execution time of homomorphic encryption applications. We implement Hypnos on the QianKun FPGA Card and highlight the following results: (1) outperforms SOTA ASIC and FPGA solutions by 2.58× and 4.43×; (2) the communication overhead is reduced by 3.78× compared to traditional architectures; (3) up to 27.6× and 19.06× energy efficiency improvement compared to ASIC-based CraterLake and FPGA-based Poseidon for ResNet-20 respectively.**

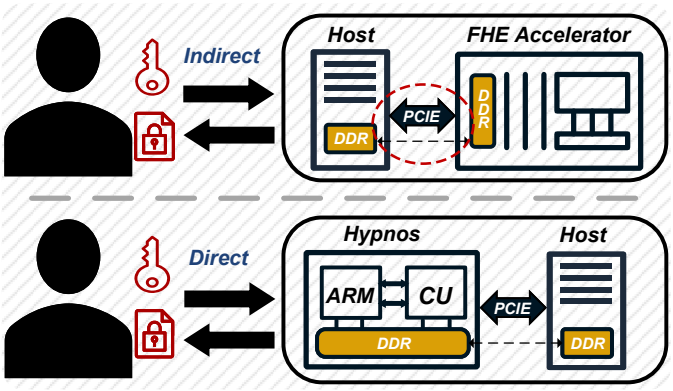

Fig. 1. Comparison with Traditional FHE Accelerators: In Hypnos, data processing is conducted directly by the CPU within the device, eliminating the dependency on the host CPU and bypassing the congested system PCIe bus.

I. Introduction

With the rapid development of the internet and cloud computing, privacy protection and security have become major challenges in information processing. Against this backdrop, the importance of data privacy computation technology has become increasingly prominent. Fully Homomorphic Encryption (FHE) is an important privacy computation technology that is crucial in achieving secure computations [1], [2].

The primary advantage of FHE is its ability to perform unlimited addition and multiplication operations without decrypting the data. This allows complex computations to be executed while the data remains encrypted. However, this characteristic of FHE results in an explosive expansion in data volume and computational complexity, ranging from 10,000 to 100,000 times. This places considerable strain on memory and computational systems, making it impractical for general-purpose computing hardware to handle FHE computations effectively. Therefore, it is essential to develop specialized architectures to enhance the efficiency of FHE computations.

In recent years, several FHE-specific accelerator architectures have been for FHE acceleration based on FPGA [3], [4] and ASIC [5]–[9]. The advantage of using FPGAs lies in their lower cost and programmable nature, which allows for flexible iterations and upgrades of hardware designs to achieve rapid deployment in response to the evolving FHE algorithms. In contrast, ASICs deliver higher performance and lower power consumption, allocating more substantial hardware resources to address the bottlenecks in FHE, such as employing up to 180MB-512MB of on-chip scratchpad to mitigate the overhead caused by data transfers between off-chip memory and computational units (CUs) [6]–[8]. However, these ASICs are currently in the simulation phase, making it difficult to rapidly deploy them for real-world use.

Prior works have typically assumed that substantial volumes of ciphertext and keys are always stored within the DDR/HBM of the accelerator. However, this assumption is overly idealistic, as handling intricate FHE applications often exceeds the capacity of on-chip memory. This indicates that communication between the host and the accelerator via the PCIe bus will be frequent. As depicted in the upper section of Fig.1, communication between the user and the accelerator is mediated by host memory, necessitating data to traverse the PCIe bus after initial storage in host memory. This bottleneck can potentially undermine the performance gains reported in earlier studies.

Current designs of FHE accelerator architectures emphasize enhancing computational efficiency and optimizing the utilization of on-chip memory [10]. They often neglect the utilization efficiency of off-chip memory. In fact, in FHE schemes based on Residue Number Systems (RNS), the sizes of ciphertexts and keys vary dynamically during computation. Allocating and managing memory with a fixed size will lead to a large amount of redundant information being stored in memory, thereby reducing memory utilization efficiency. It will exacerbate the overhead of PCIe communication.

Haoxuan Wang and **Yinghao Yang** are co-first authors. **Hang Lu** is the corresponding author. Bo Wang <wangbo@phytium.com.cn>, Changshan Su <suchangshan1101@phytium.com.cn>, Xiaoyu Li <lixiaoyu2261@phytium.com.cn>, Gen Li <ligen@phytium.com.cn>, Yuxing Tang <tangyuxing@phytium.com.cn> (all with the Phytium Information Technology Co., LTD) contribute significantly to this paper in terms of the manuscript submission and rebuttal, and should all be regarded as co-authors.

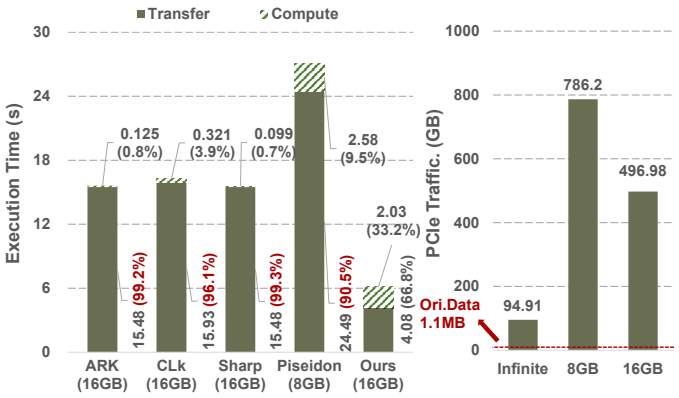

Fig. 2. Execution time breakdown of ResNet-20. The data size under FHE encryption is severely inflated compared to plaintext (1.1MB marked in red), introducing significant PCIe communication overheads. PCIe transfer overhead is high for existing accelerator architectures, over 96% for ASIC accelerators and up to 90% for FPGA-based Poseidon.

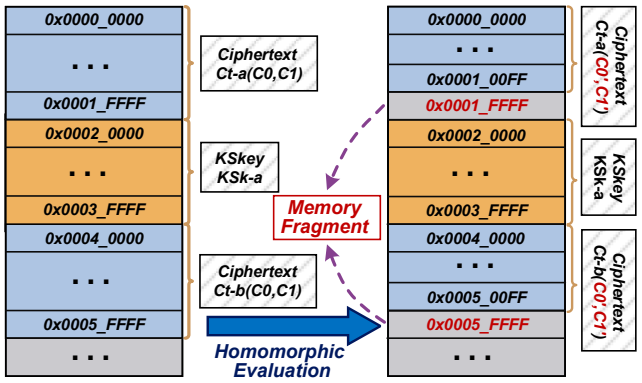

Fig. 3. Memory fragmentation during homomorphic computation. Changes in the number of RNS components of ciphertext lead to memory fragmentation.

In this paper, we propose Hypnos, a memory-efficient homomorphic processing unit that changes the way computational units access encrypted data and significantly reduces the amount of ciphertext and key transmission. For RNS-based FHE applications where the data volume exceeds the off-chip memory, we propose a homomorphic encryption page memory management system, which reduces memory fragmentation and the volume of data communicated with the host.

In summary, our contributions are as follows:

- We propose a practical, memory-efficient homomorphic processing unit named Hypnos. It aims to mitigate the performance overhead caused by inefficient PCIe communication and improve off-chip memory utilization.
- We propose a novel memory management system based on "HE page" to manage memory at the RNS component level, significantly increase memory utilization, and further reduce the data transfer volumes during the execution of FHE applications.
- We implement a prototype of Hypnos on the commercial Qiankun, and evaluate its performance in practical FHE applications. We highlight the following results: (1) Hypnos outperforms SOTA ASIC and FPGA solutions by 2.58× and 4.43×, respectively; (2) communication overhead is reduced by 3.78× compared to traditional accelerator architectures; (3)In terms of energy efficiency, Hypnos also surpasses existing ASIC and FPGA accelerators by a factor of 27.6× and 19.06×, respectively.

II. Background & Motivation

A. Platform Limitations for FHE Accelerators

Due to the on-board memory limitation problems, accelerators must frequently transmit large volumes of ciphertext and keys when processing FHE applications. This extensive data movement between the host DDR and the accelerator's local memory can severely impact performance. Prior works have often underestimated this issue. In our evaluation of the ResNet-20 application based on the SEAL library, as depicted in the left half of Fig. 2, despite the superior computational efficiency of ASIC accelerators, the overhead associated with PCIe transfers can account for up to 96% of the total execution time. Similarly, FPGA accelerators confront analogous issues attributable to their limited onboard memory. The right side of Fig. 2 shows the increase in PCIe communication overhead under different onboard memory sizes, where the "infinite" item represents the communication scenario when all data is loaded into the onboard memory at once. The results show that, under memory constraints, PCIe communication volume can surge dramatically. Therefore, reducing PCIe communication overhead is a critical consideration for FHE accelerators. Otherwise, continuously optimizing the performance of the accelerators alone will not eliminate the PCIe communication bottleneck within the entire acceleration system.

To address this bottleneck, we propose integrating a CPU onto the accelerator to manage the direct transmission of ciphertexts and keys, thereby reducing the overhead associated with PCIe communications. Hypnos exhibiting similar computational performance compared to the FPGA-based accelerator Poseidon achieves superior overall performance due to its reduced PCIe overhead. Even when benchmarked against more computationally powerful ASICs, Hypnos can elevate system performance through optimized PCIe communication.

B. Neglected Memory Fragmentation

In RNS-based FHE schemes, ciphertexts and keys are represented as polynomials. As shown in Fig. 3, during computations, the number of polynomials changes, leading to variations in ciphertext size and the number of RNS components required for keys. Fixed-size memory management results in memory fragmentation and decreased utilization, particularly evident in applications like ResNet-20 and LR-Train using the CKKS scheme, where memory efficiency is notably reduced, increasing the frequency of data replacement and PCIe communication overhead. However, the BFV scheme, with its constant ciphertext and key sizes, is not affected by this issue.

Fortunately, the size of the RNS components in RNS-based FHE schemes is predetermined, meaning that all ciphertexts and keys can be considered composed of multiple fixed-size data segments. This provides an opportunity for optimizing memory utilization efficiency.

III. Hypons Architecture

A. Architecture Overview

As shown in Fig.4, the Hypnos architecture comprises three distinct regions: the Processing System (PS), Programmable Logic (PL), and Network-On-Chip (NOC).

The PS region is equipped with a CPU that runs an operating system, responsible for initializing the state of the compute units and executing FHE computation applications. This includes handling FHE operations, data transfers, and monitoring the status of the compute units.

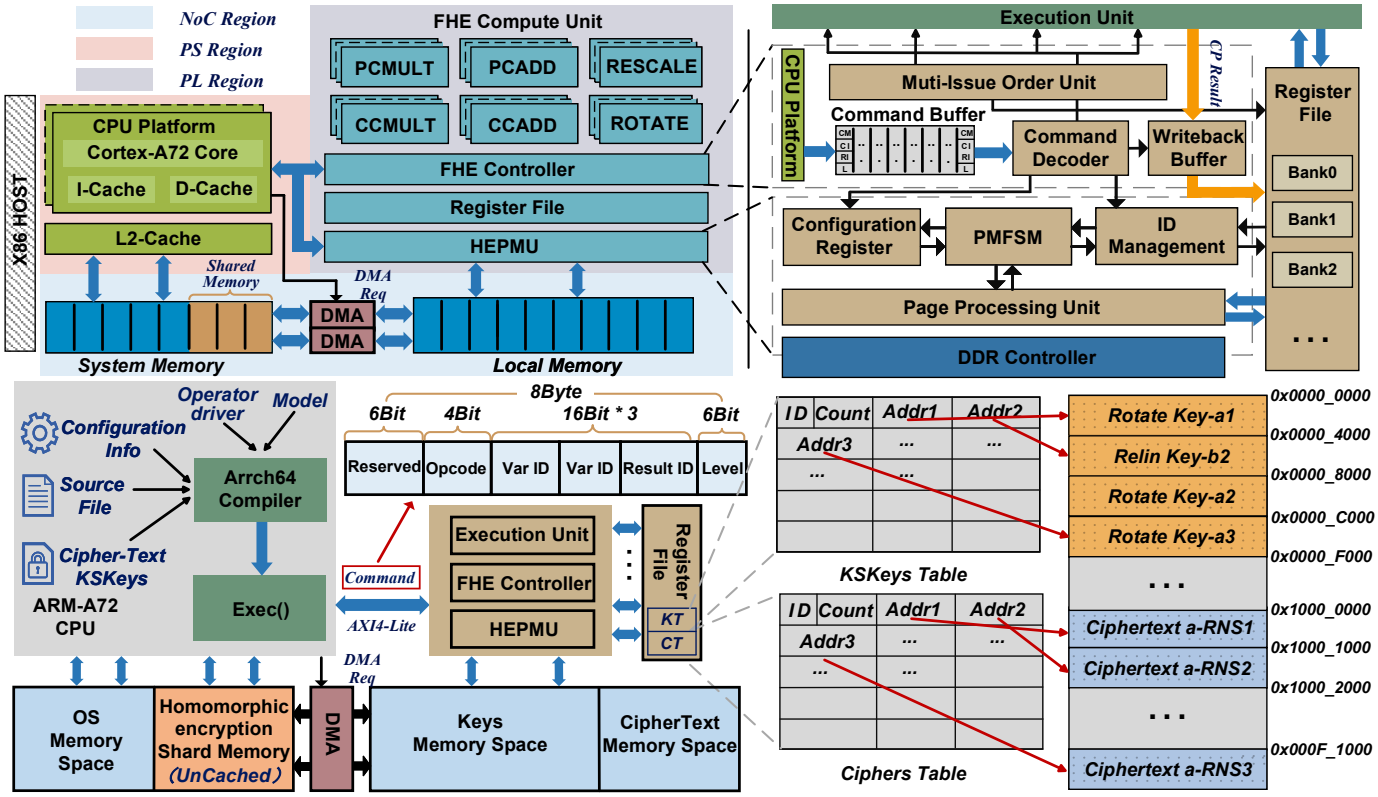

Fig. 4. Hypnos overall architecture. The heterogeneous ARM+FPGA computing architecture features a Cortex-A72 CPU, connected through a NoC to various components including the FHE CUs, HEPMU, and memory subsystems. This setup highlights the multi-layered interaction between processing units, local and system memories, DMA channels, and specialized encryption hardware, illustrating a memory-efficient system for FHE acceleration.

The PL region offers reconfigurable hardware resources serving as compute unit(CU), composed of six fundamental FHE operators that support computations for various FHE schemes. The CU includes an FHE controller, register file, and Homomorphic Encryption Page Management Unit(HEPMU), responsible for command execution, caching, and memory management, respectively.

The NOC region acts as the interconnection network, utilizing the on-chip buses of the AMD Xilinx Versal ACAP series to facilitate high-speed communication between components and support point-to-point data transfers.

B. FHE Execution&Controller Unit

Hypnos adopts an abstracted methodology utilizing operational-level operators, as opposed to basic operators (such as NTT, MM, MA, etc.), providing 64 parallel processing units. Within these operational-level constructs, basic operators function in a cascaded arrangement, with each unit possessing its own cache, thereby mitigating accesses to main memory and enhancing the efficiency of request processing alongside the overall system performance. Internally, the Barrett Reduction algorithm [11] is employed for modular multiplication (MM), while the Number Theoretic Transform (NTT) utilizes a radix-8 design and incorporates optimizations derived from Poseidon [4].

The FHE controller is responsible for scheduling, whereas the command decoder assigns tasks to the functional units. The role of the Multi-Issue Order Unit is to disseminate processed command information to the CUs. Computed results are subsequently stored in registers via a write-back buffer. The HEPMU manages memory operations and returns results either to the CPU or for further computations.

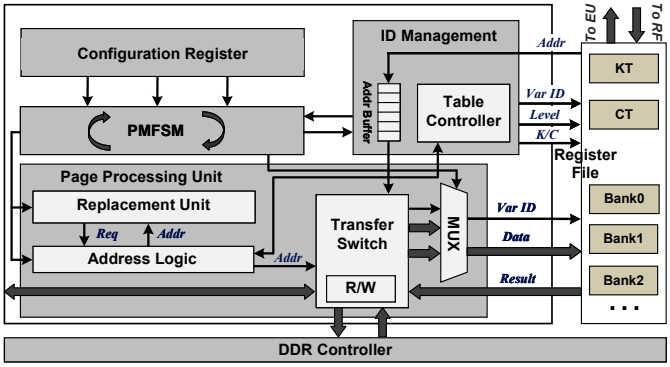

Fig. 5. Micro-architecture of Homomorphic Encryption Page Management Unit (HEPMU).

C. HEPMU

The cornerstone of Hypnos is the HEPMU, which implements paged memory management at the granularity of RNS components. The microarchitecture of the HEPMU is illustrated in Fig.5. The detailed workflow of the HEPMU will be elaborated upon in Sec.IV; herein, we focus solely on the hardware architecture design.

Configuration Register provides scalable memory management granularity and table entries of varying sizes based on different applications and security parameters. Configuration details are written

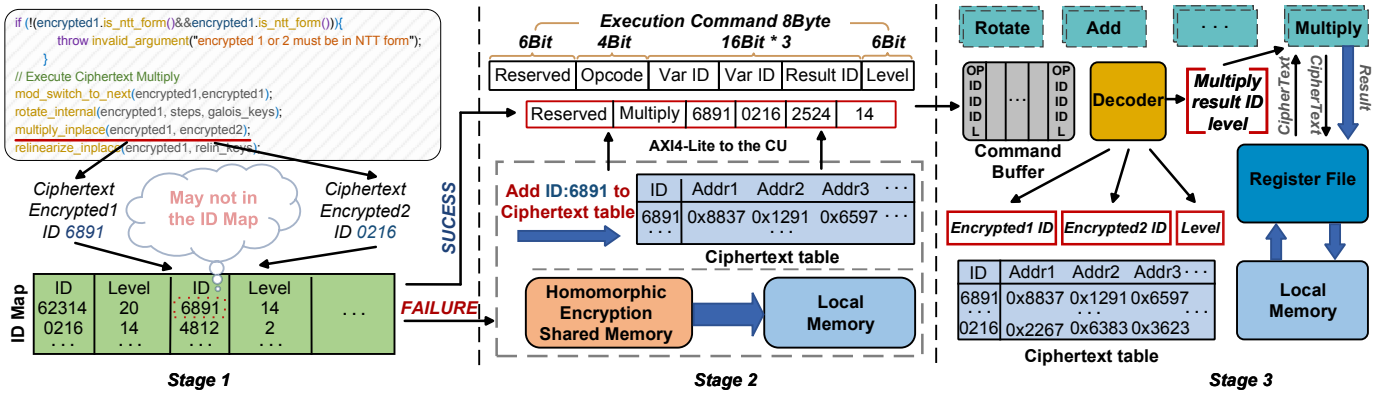

Fig. 6. This diagram illustrates the multi-stage execution process of a command within Hypnos, providing a detailed account of the entire workflow. Each stage is managed and invoked via specific operation IDs, addresses, and level parameters, ensuring the accurate execution of the encrypted command.

via the AXI4-Lite bus and are utilized during the execution of FHE applications. Additionally, the CPU can read this configuration information through the same bus to determine the status of Hypnos.

Page Management Finite State Machine (PMFSM) acts as the central control logic, switching between idle, write, eviction, and read states according to the sequence of commands, coordinating task scheduling within the HEPMU.

ID Management performs ID-to-address translation, querying addresses from the Key Table (KT) and Ciphertext Table (CT) based on the type of ciphertext variable, and writing addresses to the buffer. Notably, the CT and KT are stored in a register file to accommodate dynamic changes in table sizes due to varying parameter configurations. For example, when the modulo chain length $L = 32$ and the memory size is 16GB, the sizes of CT and KT are 69KB and 17.25KB, respectively.

Page Processing Unit (PPU) executes operations related to the RNS page table, retrieving RNS components based on address information and placing them in the register file for computation. When page replacement is required, it writes page information back to the KT or CT.

It is noteworthy that, as shown in the lower-left portion of Fig.4, a portion of the system memory is designated as shared memory, specifically reserved for storing ciphertexts and keys, and configured as uncacheable. This design choice is because, during homomorphic computations, the CPU processes address information rather than the ciphertexts and keys themselves. Moreover, the uncacheable attribute of the shared memory eliminates the computational overhead associated with cache coherence checks and cache flushing. Given that different applications have varying requirements for ciphertext and key memory spaces, the sizes of these spaces are dynamically specified by the FHE program running on the CPU. For instance, in CNN applications that require a large number of keys, a larger key storage space can be allocated to mitigate the need for frequent data exchanges.

Regarding the design of page table replacement strategies in the HEPMU, we evaluated several strategies for ResNet-20 on Hypnos. The performance of ResNet-20 indicated that, both in terms of execution time and the number of replacements, the Least Recently Used (LRU) strategy demonstrated performance closest to the optimal (OPT). Consequently, we adopted LRU as the page table replacement strategy in the HEPMU.

IV. HE Paged Management System

In prior accelerator designs, despite the adoption of RNS decomposition, ciphertexts and keys were still transmitted and stored as whole units. Only during the execution phase were the necessary RNS components used for computation, leading to unused RNS components inefficiently occupying memory space. To address this, we propose a memory-efficient homomorphic paging memory management system designed to manage ciphertexts and keys in memory with precision, thereby optimizing memory usage and enhancing performance.

This system operates at the level of RNS components and achieves software-hardware collaboration. The CPU is responsible for executing the top-level homomorphic encryption application code, while the CU manages the physical addresses of homomorphic variables and executes FHE operators. The close collaboration between the CPU and CU is key to achieving efficient computation. Fig. 6 illustrates the execution flow of homomorphic encryption application computations, including the details of memory management.

CPU Side: FHE applications execute on the CPU after specialized compilation. As shown in Stage ❶ of Fig. 6, when the program requires acceleration via FHE operators, it first queries the *ID Map* maintained on the software end to verify whether the required variables exist in the *CU* memory. In Stage ❷, if all variables are present, the program sends the command data to the CU and waits for a result signal. In the lower part of Stage ❷, if any variables are missing, they are sent to the *CU*, and the *PPU* updates the variable IDs and their levels in the *ID Map* and the *CT* or *KT*. During this process, the *CPU* only accesses the *IDs* and *Level* of the variables and awaits an interrupt signal indicating the completion of the computation.

Compute Unit Side: Address management, data replacement, and write-back within the memory system are all performed on the *CU* side. Upon receiving a command, the compute unit verifies that all ciphertext variables are present in memory. It then retrieves and decodes the command content from the *Command Buffer*, using the *IDs* to locate the physical addresses of the variables in the *CT* and *KT*. The *Level* indicates the required depth of multiplication. The *HEPMU* transfers the variables to the register file, and the decoder inputs the type of computation and auxiliary information to the operator. Once the computation is complete, the *result ID* and the computed result are written back to the *register file*, as shown in Stage ❸. The *CU* notifies the *CPU* of the result and its physical address via an interrupt. The *CPU* then fetches the result to complete the command execution cycle.

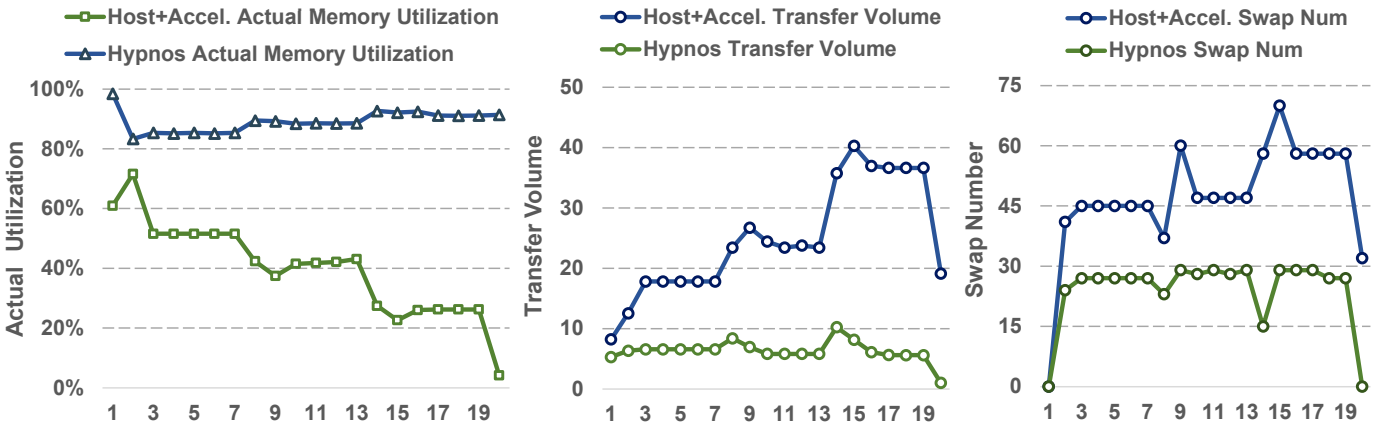

Fig. 7. How (a) memory utilization, (b) transfer volume, (c) swap number change at each layer of ResNet-20 under encrypted execution when comparing Hypnos to traditional architectures, with both systems featuring a uniform memory size of 16GB and differing in granularity: the traditional architecture managing memory at the granularity of entire ciphertexts, while Hypnos operates at the granularity of a single RNS component.

TABLE I
PERFORMANCE COMPARISON OF FOUR BENCHMARK APPLICATIONS WITH THE BASELINE FHE ACCELERATOR, MEASURED IN TWO TIME UNITS: SECONDS (S) AND MILLISECONDS (MS)

Works	ResNet-20 (in s)	LR-Train (in s)	PSI (in ms)	PIR (in ms)
(ASIC) ARK	15.61	0.34	422.73	306.10
(ASIC) Craterlake	15.80	0.25	408.45	303.97
(ASIC) Sharp	15.58	0.18	417.73	305.23
(FPGA) Poseidon	27.07	2.32	689.79	389.77
(FPGA) FAB	22.68	3.22	1168.12	472.30
(FPGA)Hypnos	**6.11**	**1.89**	**278.27**	**86.72**

V. EVALUATION

A. Experimental Setup

Platform. Hypnos is implemented on the Qiankun FPGA Card from Ant Inc., featuring a dual-core ARM A72 processor, over 1.7 million LUTs, and more than 7000 DSPs. The specific FPGA model is the xcvp1502-vsva2785-2MHP-e-S.

Baseline. In our experiments, Hypnos was compared with state-of-the-art ASIC-based FHE accelerators, including ARK [6], Craterlake [5], and SHARP [8], and FPGA-based accelerators such as FAB [3] and Poseidon [4]. Some accelerators only implemented portions of the Benchmark. For clarity, we evaluated these architectures based on the complexity of each Benchmark.

Benchmark: We use the following 4 benchmarks (as shown in TableI) for evaluation:

(1) ResNet-20: Implemented and trained with the CIFAR-10 dataset, featuring 19 convolutional layers and one fully-connected layer, with ReLU activation functions [12].

(2) Logistic Regression (LR): Based on the HELR algorithm using the CKKS scheme [1], incorporating Bootstrapping with a multiplication depth of $L = 38$ evaluated over 10 iterations supported by two Bootstrapping operations.

(3) Private Set Intersection (PSI): Evaluated using the SEAL APSI open-source library for computing the intersection of two asymmetric sets, with the sender's set containing 2^{28} elements and the receiver's set 2^{10} elements [13].

(4) Private Information Retrieval (PIR): Benchmark evaluates the SEAL PIR open-source library for encrypted queries on the server side, involving a query for a single item from a dataset of 2^{20} entries [14].

B. Full-system Performance

The design goal of Hypnos is to provide an architecture capable of running complete FHE applications. We demonstrate the full-system performance of Hypnos in comparison to previous accelerators. As shown in Table I, for data-intensive applications such as ResNet-20, PSI, and PIR, the primary bottleneck is the overhead of memory access. Hypnos can leverage its strengths, achieving up to a 2.58× improvement in performance on ASIC-based accelerators and up to a 4.4× improvement on FPGA-based accelerators. LR-Train, on the other hand, is a HE application that is computationally intensive with minimal memory access; it does not involve large-scale databases or extensive rotating keys. In terms of computational performance, Hypnos achieves a 1.7× improvement over FPGA accelerators, though a performance gap remains compared to ASIC accelerators.

C. Energy Efficiency

We utilize Energy-Delay Product (EDP) as the metric for energy efficiency. TableII presents the energy efficiency performance of Hypnos compared with ASIC-based accelerators ARK and Crakelake, and the FPGA-based accelerator Poseidon across four benchmarks. In ResNet-20, PSI and PIR, Hypnos outperforms both ASIC and FPGA accelerators, achieving enhancements of 27.6×, and 19.06× respectively. This is primarily due to the substantial data transfers, which contribute to significant PCIe overhead. Hypnos' memory-efficient architecture excels in managing these demands. However, in the LR-Train applications, Hypnos does not match the performance of ASIC accelerators in terms of EDP, due to FPGA's limited resources, yet it still surpasses the FPGA accelerator Poseidon.

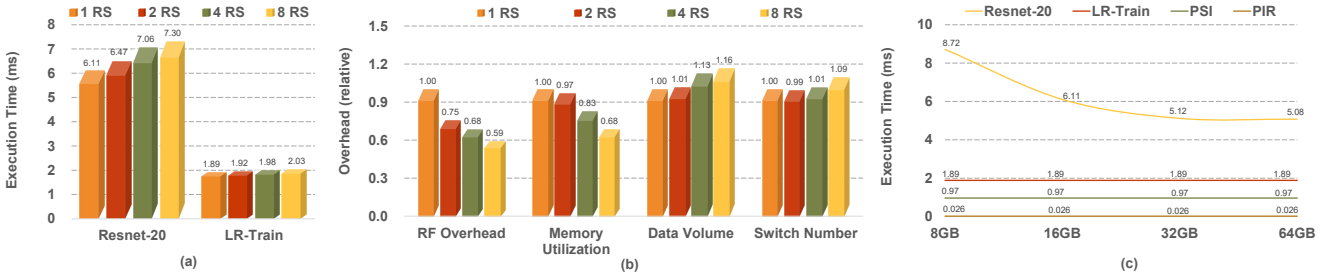

Fig. 8. Analysis of sensitivity. (a) and (b) explore the granularity of memory management within ResNet-20 and LR-Train. (c) examines the impact of memory size on the Hypnos for selected benchmarks.

TABLE II
EFFICIENCY ANALYSIS. WE USE ENERGY-DELAY PRODUCT (EDP) AS THE METRIC. LOWER IS BETTER.

Work	ResNet-20	LR-Train	PSI	PIR
ARK	68545.1	32.5	50267.4	26356.8
Craterlake	51675.5	12.9	34533.3	19125.8
Poseidon	87619.1	643.6	56892.1	18165.3
Hypnos	**4730.1**	**452.6**	**9810.7**	**952.8**

TABLE III
HARDWARE RESOURCE UTILIZATION

Resources	LUT(K)	FF(K)	BRAM	DSP
Used	860	1,115	2,511	4,990
Percentage	(50.02%)	(32.41%)	(65.36%)	(67.07%)

D. Memory

In prior work, both ASIC and FPGA-based FHE accelerators utilize a Host as the master end for HE applications, necessitating data transfers via the bandwidth-limited PCIe protocol. Consequently, we abstract this arrangement into a Host+Accelerator model. Fig.7 uses ResNet-20 as an example, with the CNN layers represented on the horizontal axis, to compare and analyze the advantages of the Hypnos architecture over traditional architectures through three metrics.

(1) Memory Utilization: Defined as the percentage of RNS components used for computation at each layer. For example, with L_{max}=30 and an encryption operation level of 10, the MU is 33.33%. Traditional accelerators exhibit low MU due to unused data occupying memory; MU can drop below 20% at the final layer. Hypnos does not reach 100% MU because some variables are frequently reused in higher-level computations, optimizing which would degrade performance.

(2) Transfer Volume: Refers to the data volume transferred via PCIe at each layer. As shown in Fig.7 (b), initially, Hypnos exhibits similar transfer volumes to traditional architectures since variables are fresh ciphertexts at higher levels. Mid-layer volumes are reduced by 73.55% due to Hypnos' homomorphic encryption page-based memory management. At the final layer, volumes decrease sharply as it is a fully connected layer requiring less data.

(3) Swap Number: Counts the number of memory variable swaps due to insufficient memory. The decrease in swap frequency can be attributed to the increase in MU. As shown in Fig.7 (c), Hypnos decreases the total number of swaps by more than 50% compared to traditional architectures.

E. FPGA Resource Utilization

Hypnos operates on an ARM+FPGA heterogeneous computing platform. TableIII delineates the resource utilization within the PL region. This includes the CU, controllers, register files, and the HEPMU. A significant portion of the DSP blocks are allocated to operator computations, whereas the DSP blocks within the HEPMU primarily support the hardware implementation of the LRU replacement algorithm and manage physical addresses. In the context of the HEPMU, BRAM is utilized for caching homomorphically encrypted variables. It is worth noting that the ARM processor, NoC, and DDR controller on the FPGA card are implemented as hard IP cores and thus do not consume any of the FPGA's programmable resources.

F. Sensitivity

Fig. 8 explores the (a) (b) impact of memory management granularity based on RNS components and (c) the size of the memory. Larger RNS granularity leads to increased memory fragmentation, reducing FHE storage system efficiency and overall performance. Using ResNet-20 as an example, increasing the granularity to L_{max} diminishes memory utilization, increasing data transfer volume and swap frequency to pre-optimization levels. For applications with large data volumes, increasing off-chip memory from 8GB to 16GB significantly reduces execution time, with minimal gains beyond 16GB. Smaller data volume applications are less sensitive to memory size, indicating that memory size allocation should be tailored to meet specific application requirements

VI. CONCLUSION

In this paper, we implemented the memory-efficient homomorphic processing unit Hypnos on the FPGA card. By integrating a CPU, the CU can directly access homomorphically encrypted data. Additionally, we introduce a homomorphic encryption paging memory management system that operates at the granularity of RNS components to manage memory, thereby reducing data movement during execution. Hypnos demonstrates remarkable efficiency in performing data-intensive computations. Compared to FHE accelerators based on ASICs, utilizing FPGAs as an implementation platform has more practical significance, enabling lower-cost updates and iterations of FHE algorithms. We hope our work will inspire new design concepts for domain-specific accelerators tailored for FHE applications.

VII. ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the CCF-Phytium Fund 2023.

REFERENCES

[1] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 9466–9471.

[2] H. Chabanne, R. Lescuyer, J. Milgram, C. Morel, and E. Prouff, "Recognition over encrypted faces," in *Mobile, Secure, and Programmable Networking: 4th International Conference, MSPN 2018, Paris, France, June 18-20, 2018, Revised Selected Papers 4*. Springer, 2019, pp. 174–191.

[3] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[4] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 870–881.

[5] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data." in *ISCA*, 2022, pp. 173–187.

[6] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[7] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 711–725.

[8] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[9] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252.

[10] R. Agrawal, L. De Castro, C. Juvekar, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Mad: Memory-aware design techniques for accelerating fully homomorphic encryption," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 685–697.

[11] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[12] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[13] "Microsoft APSI," https://github.com/microsoft/APSI, Dec. 2023, microsoft Research, Redmond, WA.

[14] "Microsoft SealPIR," https://github.com/microsoft/SealPIR, Jun. 2023, microsoft Research, Redmond, WA.