Chiplever: Towards Effortless Extension of Chiplet-based System for Fully Homomorphic Encryption

Yibo Du[1,3], Ying Wang[1,3], Bing Li[4], Fuping Li[2,3] Shengwen Liang[2,3], Huawei Li[2,3], Xiaowei Li[2,3] and Yinhe Han[1,3]
{duyibo21s, wangying2009, lifuping20s, liangshengwen, lihuawei, lxw, yinhes}@ict.ac.cn, bing.li@cnu.edu.cn
[1]*CICS, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[2]*SKLP, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[3]*University of Chinese Academy of Sciences, Beijing, China* [4]*Capital Normal University, Beijing, China*

ABSTRACT

Fully Homomorphic Encryption (FHE) is one of the most promising privacy-preserving techniques that has drawn increasing attention from academia and industry due to its ideal security. Chiplet-based designs integrate multiple dies into the package delivering high performance and thereby are embraced by the resources-hungry FHE. Despite the chiplet-based system with various specialized accelerators, it falls short in supporting FHE with the novel polynomial operations. For a chiplet-based system that is not tailored for FHE, one common approach to support FHE is designing a new dedicated accelerator, However, this full design-and-build approach overlooks the existing abundant resources of accelerators in the system and incurs repeated customization and resource waste.

In this paper, we propose Chiplever, a framework enables effortless extension of Chiplet-based system for FHE. We aim to fully harness the available resources in the room for efficient FHE. To achieve this, Chiplever (1)introduces a specialized extension in I/O Chiplet guided by semantics matching (2)and proposes an efficient allocator featuring specialized dataflow scheduling. (3)Chiplever provides three-step mapping to achieve compiler-level to hardware-level support for FHE and optimizes the data communications.

1 INTRODUCTION

Fully homomorphic encryption (FHE) emerges as one of the most promising encryption techniques for its ideal security properties[1]. FHE allows the computing of encrypted data directly without decrypting it, which achieves the secure offload to an untrusted platform to harness its computing power. FHE based on LWE/RLWE[2] represents ciphertext (encrypted data) in polynomial format. Therefore operations in FHE basically involve intensive polynomial operations, which require massive hardware resources. Chiplet-based designs that integrate many chips into a large system can provide abundant resources and high computing power, thereby being embraced by resource-hungry applications in the post-Moore era[3–5]. For one thing, chiplet-based design enables a more efficient chiplet reuse compared with IP reuse. Reusing third-party efficient chiplets facilitates the agile high-performance design and amortizes the development costs. For another thing, driven by the diverse emerging applications, integrating various specialized processors, known as domain-specific accelerators (DSA), can deliver high energy efficiency and computing power.

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3657321

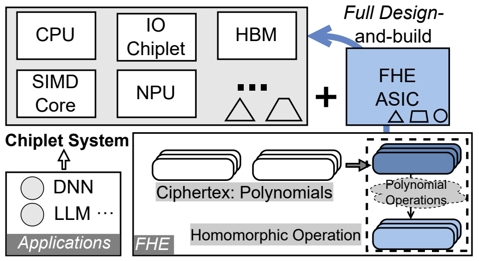

Figure 1: Chiplet-based system with abundant resources and the conventional full design-and-build approach for FHE.

However, despite with various and powerful specialized processors integrated, the chiplet-based system not designed for FHE that is primarily equipped chiplets catering to applications like deep learning falls short in supporting FHE with novel polynomial operations. In this era prioritizing data privacy, there arises an urgent necessity for systems with efficient capabilities for FHE[6]. Given that abundant resources exist in chiplet-based systems, it is crucial to fully harness the computing power already in the room and reuse these mature designs. Following this key philosophy, we propose a plug-and-play design that can take advantage of the available resources and empower existing system not designed for encrypted computing to seamlessly support FHE with effortless extension.

Compared with the conventional full design-build approach, the effortless extension design approach shows much progressiveness. When encountering the need to run FHE workload, the conventional full design-build approach straightforwardly customizes a new ASIC and integrates it into the system, as shown in Figure 1. For instance, [5] proposed a chiplet-based FHE accelerator that aims to enhance homomorphic encryption deployment. There are also specialized accelerators with dedicated units and datapath to provide efficient acceleration[7–11]. However, this full design-build approach faces some shortcomings. First, it overlooks the existing computing and memory resources in the system and suffers from low area efficiency. The specialized chiplets with DSAs can provide valuable capabilities and foundations, which are potential assets that can be utilized for FHE computing. However, the full design-build approach re-designs the hardware that overlaps with the functionality of existing chiplets, leading to repeated customization and resource waste. As FHE is known to be very greedy, redundant customization causes a worse resource consumption. FHE accelerators in the full design-build style can cost at most 472 mm^2[11]. Second, the full design-build approach faces much more complexity and escalated design efforts. Designing a whole new accelerator involves a long design flow and requires substantial efforts.

To overcome these shortcomings, we seek for an approach that can take advantage of on-hand resources as much as possible to

*Corresponding authors are Ying Wang and Yinhe Han.

achieve efficient FHE on a non-FHE-tailored system. To achieve this, we propose a framework that introduces an effortless chiplet-based extension to provide specific functionalities for the system. To minimize the changes to the system, we propose to only define the necessary functionalities in the extension that the system lacks. However, there are still challenges to achieve this method. First, an efficient extension that can empower the system to support FHE workloads poses a great challenge. It needs a co-analysis of the FHE application-level and hardware-level semantic characteristics to determine the optimal workload mapping across multiple chiplets. Second, incorporating the extension into the system and coordinating each chiplet while being transparent with the existing chiplets is challenging. To fully harness the available resources, chiplets are allocated workload partitions and work cooperatively. Efficient data communications between chiplets are required. Straightforwardly integrating a new chip may affect the topology and thereby change data communication behaviors, such as changing routing paths. In order to be transparent with the previous communication pattern, we introduce the extension with decoupled scheduling in the I/O chiplet having no effect on the original topology.

In this paper, we propose Chiplever, a framework towards the effortless extension of chiplet-based system to enable the system not originally tailored for encrypted computing to seamlessly support FHE. We aim to achieve this with minimal efforts and modifications while fully capitalizing on the abundant and mature resources of existing systems. Chiplever introduces an efficient extension that encompasses both function support and scheduling support. The newly incorporated extension needs to transfer data between chiplets. To guarantee transparency, we propose a decoupled allocator featuring dedicated task scheduling for FHE without affecting previous communication behaviors. To guarantee compatibility of extension with the system, Chiplever presents a three-step mapping facilitating compiler-level to hardware-level support for FHE.

The main contributions of this paper are as follows:

- We propose Chiplever, a framework that effortlessly introduces an extension to non-encrypted computing chiplet-based systems to seamlessly support FHE while fully leveraging on-hand resources.
- Chiplever provides essential function support for the special operations of FHE guided by semantics matching and also scheduling support in the extension that offers efficient communications and ensure transparent integration.
- we propose a three-step mapping in Chiplever to facilitate compiler-level to hardware-level support for FHE and optimize the data communications.
- Compared with the designs under the conventional full design-and-build approach, evaluations show that Chiplever achieves significant performance and area energy-efficiency improvement.

2 BACKGROUND AND MOTIVATION

2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is one of the most promising privacy-preserving techniques. FHE achieves generic computation over the encrypted data that allows computing on encrypted data directly without decrypting it. FHE schemes based on the LWE/RLWE[2] adopt the same basic ciphertext format: polynomials with degree N. The computations over the encrypted data, referred to as **homomorphic operations**, are represented as polynomial operations. For example, the multiplication of two plaintexts is transformed into ciphertext polynomial multiplications which have $o(N^2)$ complexity. To optimize polynomial operations, NTT is adopted in the current FHE implementations to reduce the complexity into $O(N \log N)$.

A homomorphic operation such as homomorphic multiplication encrypted from a simple plaintext operation involves multiple complex operations. For example, the FHE scheme has to periodically invoke a bootstrapping(BSP) operation because each FHE operation inevitably introduces a certain amount of noise into the ciphertext, which can lead to unsuccessful decryption. BSP can reduce the noise but is expensive. In TFHE[12], BSP involves loops of polynomial operations with dependencies. In addition, polynomial multiplications in CKKS[13] follow a key-switching operation to produce a ciphertext under the original secret key. To sum up, homomorphic operations in FHE consist of multiple complex operations.

2.2 Chiplet-based System and Conventional Full Design-and-build Approach

Chiplet-based design integrates many small chiplets into a large system through the silicon interposer, which enables die-to-die connection and provides high bandwidth[14]. This has been demonstrated practical to efficiently integrate abundant chiplets[15]. Therefore, chiplet-based designs are embraced by resource-hungry applications as the slowing of Moore's Law. Chiplet-based system integrated with various domain-specific accelerators (DSA), as shown in Figure 2, can deliver high energy efficiency and computing power. The I/O chiplet is typically integrated into the system with NoC and kinds of interfaces to DDR and peripherals for efficient interconnection and communication[16]. Despite this, when encountering FHE workloads, the chiplet-based system faces difficulties in supporting FHE. FHE workload cannot be directly offloaded to the DSAs as the equipped compilers for applications like deep learning can not support FHE programs. In addition, the special operations in FHE, such as NTT, cannot get efficient support from the DSA chiplets not specifically tailored for FHE.

To address this problem, some works propose dedicated FHE accelerators, typically implementing NTT units and vector units to support operations in FHE. The dedicated FHE accelerator is integrated into the system and specifically offloaded with whole FHE workloads[5, 7, 9]. However, this full design-and-build approach overlooks the potential of leveraging the abundant resources already in the room. Through the semantics matching in Section 3, we identify that certain FHE operations have the same semantics as existing chiplets. Therefore, it is possible that FHE workload can be partly accelerated by existing chiplets. In contrast, the full design-and-build approach repeatedly customizes similar hardware units already in the system, leading to redundant customizations and resource waste. We aim to propose an effortless extension that empowers systems not originally designed for FHE to support FHE by leveraging the abundant resources in the room as much as possible.

3 THE CHIPLEVER FRAMEWORK

3.1 Framework Overview.

The proposed Chiplever introduces an extension to a chiplet-based system not originally tailored for FHE like Figure 2 to empower it to support FHE workload. The framework of Chiplever is depicted

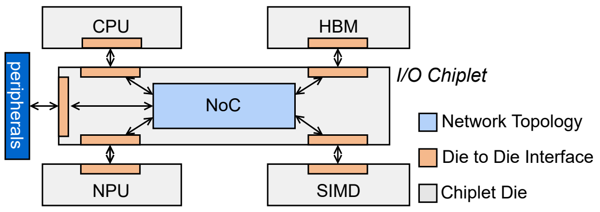

Figure 2: The baseline chiplet-based system.

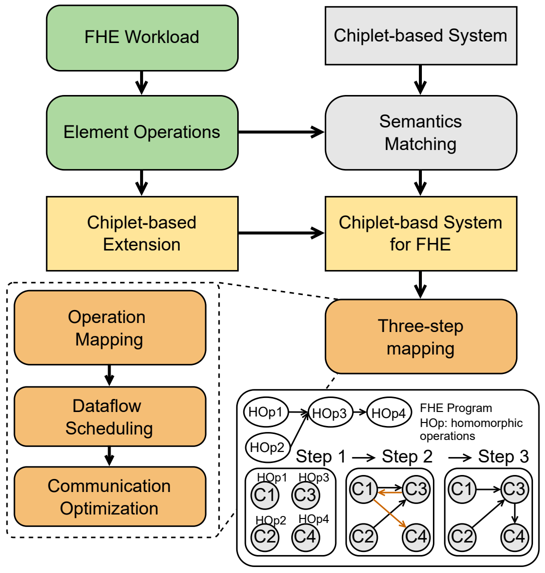

Figure 3: Framework overview of Chiplever.

in Figure 3. Chiplever aims to leverage the available Chiplets in the system and the extension to provide complete support for FHE workload. To achieve this, Chiplever first conducts semantic analysis and matching. Given a FHE workload, Chiplever analyzes its semantical characteristics to prepare for the semantics matching. The semantic matching process aligns the operations in the FHE workload with the chiplets with the same semantics in the baseline system. This semantic matching process extracts the operations that are suitable to be mapped to the available chiplets, referred to as general operations. The semantic matching process also outputs special operations that necessitate support from the extension. Therefore, guided by semantic matching, we design the extension with dedicated units for special operations. Beyond functional support, the extension is expected to offer efficient scheduling support of the FHE workload on the newly extended system. This is crucial as the newly integrated extension needs to collaborate with the existing resources of the system and transfer data. To achieve this, we customize a decoupled allocator for FHE workload. To facilitate FHE workloads and further optimize communications, we offer compiler-level to hardware-level support for the extended system. We propose a three-step mappping mechanism.

3.2 Semantics Matching

FHE workload exhibits a complex behavior. Different from deep learning applications or image signal processing, which mostly involves convolution operations or vector-matrix multiplication, FHE workloads involve a broader range of complex operations. For

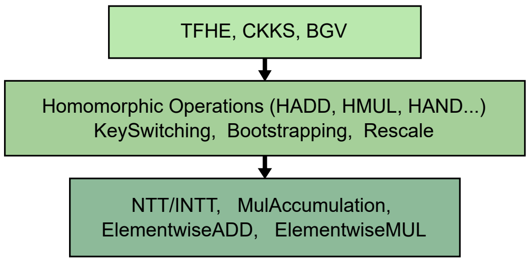
TFHE, CKKS, BGV
Homomorphic Operations (HADD, HMUL, HAND...)
KeySwitching, Bootstrapping, Rescale
NTT/INTT, MulAccumulation,
ElementwiseADD, ElementwiseMUL

Figure 4: Primitive operations in FHE.

instance, bootstrapping, a dominant procedure in TFHE, is occupied by iterations of element-wise operations and NTT operations. Therefore, for different FHE workloads such as TFHE and CKKS, we extract primitive operations by using passes to analyze the program. As shown in Figure 4, Chiplever abstracts operations by a hierarchical analysis from top to bottom. Then, Chiplever aligns the extracted primitive operations from the workload to the hardware in the chiplet system with the same semantic information. Specifically, this process conducts semantics matching to detect general operations and special operations. Chiplever extracts the **general operations** in the program that have the same semantics as the chiplets. For instance, operations such as multiplication and accumulations with clear data reuse characteristics are assigned to NPU. Element-wise operations are assigned to SIMD. Semantics matching traverses the program and the matches for the predefined patterns. By doing this, Chiplever optimally utilizes the available hardware resources in the room and takes advantage of their characteristics for high efficiency.

However, **special operations** such as NTT involved in FHE for polynomial operations cannot be matched to the existing hardware in the semantics matching process. This means that the special NTT operations are not optimally supported by the existing hardware due to their unique characteristics. Therefore, Chiplever addresses this by providing efficient support tailored for these special operations in the extension. This approach ensures that Chiplever can harness the available resources and only introduce minimal additional logic to provide efficient support.

3.3 Extension Chiplet

Based on the semantics matching results, the extension is necessitated to provide efficient support for the special operations. In addition to this, a subsequent challenge arises that how to manage dataflow scheduling between the extension and chiplets. Since the previous system does not support dataflow on the new extension, we introduce a dedicated allocator within the extension to ensure transparent integration, which does not affect original communications and dataflow scheduling. Figure 5 depicts the architecture of the extension. It includes NTT module, Allocator, and polynomial buffer for intermediate results reuse. The NTT module is implemented to support the special NTT operations in FHE. FHE represents encrypted data as the ciphertext polynomial and involves complex polynomial operations. To simply the polynomial operations, NTT and INTT are used on the polynomials, which reduces the complexity of polynomial multiplication from $O(N^2)$ to $O(N \log N)$, N is the degree of the polynomial. After input polynomials are transformed by NTT, the multiplication of two polynomials is performed as element-wise multiplication in the point-value domain. The NTT module is implemented with

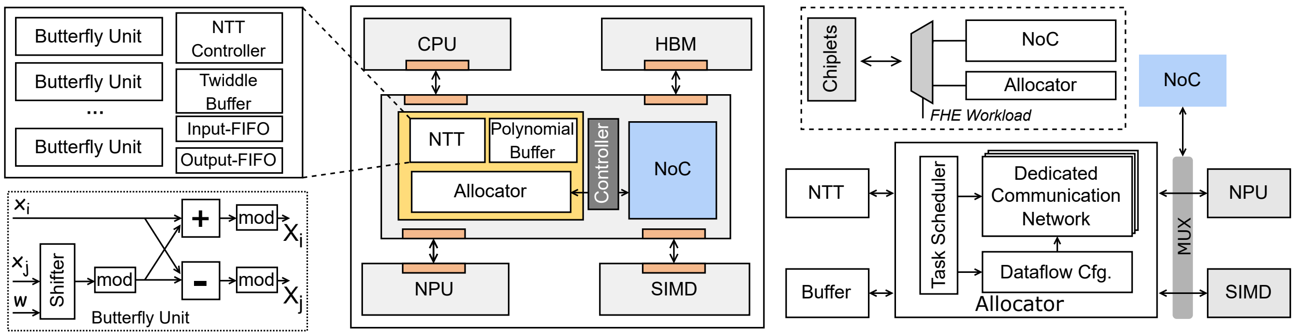

Figure 5: Chiplever extension architecture.

butterfly units unit computing the NTT. The transformation of polynomials by NTT is ($\log N$)-stage pipeline. Each stage can be executed by configurable parallel butterfly units. The butterfly unit is depicted in Figure 5. The input is multiplied by the twiddle factors. As the twiddle factors can be represented by the value of powers of 2, the multiplication can be simplified as shift operations[17].

In addition to providing the necessary functional support of special NTT operations, the extension should be able to coordinate the available chiplets and manage the dataflow on the new extended system for efficient FHE. In chiplet-based system, the dataflow is affected by the placement of chiplet and the network on chip (NoC). Directly integrating a new chiplet into the system may bring the modification of the topology or change the communication behavior such as the routing path. Therefore, we choose to extend the existing I/O chiplet and decouple the dataflow management for FHE from the original NoC and controller. This approach ensures that the extension is transparent to the system. The extension is introduced in the I/O chiplet which is integrated in the system with the specialized extension for FHE. This extended I/O chiplet retains all prior functionalities, dose not change previous communication behaviors, and is equipped with the added capability required by FHE workloads. Additionally, the I/O Chiplet serves as a critical bridge linking various chiplet dies. Adding extensions in I/O chiplet facilitates efficient communication between the extension and other chiplets. We propose a dedicated allocator in the introduced extension to achieve decoupled dataflow management for FHE. It includes a task scheduler, dedicated communication networks, and a dataflow configuration unit. The task scheduler manages the dataflow scheduling between modules for the FHE workload. It allocates data according to the operation dependencies. For instance, when the NTT module needs to transmit data to the NPU, the task scheduler allocates the data from NTT module to NPU through the dedicated communication networks. The dedicated communication networks provide multiple point-to-point connections between modules tailored to meet the special data communication needs in FHE workloads. The dedicated communication networks are under the control of the dataflow configuration unit to determine the destination and source. We decouple dataflow management in the allocator rather than in the original NoC that is general-purpose and not tailored for FHE dataflow. Besides, the decoupling is beneficial to avoid the congestion in original NoC.

3.4 Three-step Mapping

Next, we introduce the compiler-level support provided by Chiplever. For the FHE workloads based on different schemes like CKKS and TFHE, we can uniformly map them to the chiplet-based system as the Chiplever is able to support the primitive operations of FHE. As a specialized extension is introduced, the mapping and scheduling should take the new extension into consideration. Given that FHE workloads significantly differ from other workloads supported by the system such as deep learning applications, the compilers designed for deep learning are not applicable to map FHE workloads to hardware. To address these challenges, we propose a three-step mapping to facilitate the automatic mapping and scheduling of FHE workload to the chiplet-base system as shown in Figure 3.

3.4.1 Operation mapping. Firstly, operation mapping assigns each primitive homomorphic operation (as shown in Figure 4) in the FHE workload to a semantic-matching chiplet in the system to get an efficient acceleration. This step basically constructs a graph where each vertex represents a chiplet executing a primitive homomorphic operation, as the *step 1* shows in Figure 3.

3.4.2 Dataflow scheduling. Secondly, based on the operation dependencies in the FHE program, connections between vertices are established, where each edge represents the communications between hardware units. This step establishes the dataflow of the FHE workload on the Chiplet-based system. Each edge is labeled with a value representing temporal relations. The dataflow scheduling conducts scheduling according to this graph with edges to ensure an efficient pipeline of the chiplets in the system.

3.4.3 Communication optimization. Thirdly, we perform communication optimization, seeking to eliminate unnecessary data communications. For instance, as shown in Figure 3, step 3 removes the unnecessary data movement (the orange arrow in C2). C3 (Chiplet 3) receives the data from C2 and sends the data to C1 after processing. Then C4 reads data without any modifications from C1. The data communications of C3->C1 and C1->C4 are unnecessary (the orange arrows in step 2). This scenario frequently occurs in read and write accesses to DRAM. In step 3, the redundant communications are removed by directly sending data from C3 to C4. These optimizations are feasible at the compiler time due to the static nature of FHE Workload behaviors. As FHE ensures that the program does not reveal any information during runtime, the behavior of the execution is static. Leveraging this characteristic, Chiplever achieves communication optimizations during compilation.

To better understand the scheduling, we illustrate a CKKS key-switching example in Figure 6. For clarity, we only show the key operations. Chiplever maps the homomorphic operations to different chiplets in the system and schedules the dataflow by Allocator. In the beginning❶ , SIMD performs the polynomial element-wise multiplications and then sends the results to NTT module. In ❷ ,

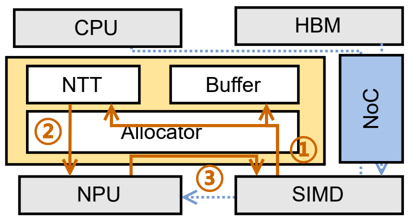

Figure 6: A Scheduling Example.

NTT module sends data to NPU and NPU performs multiplications of key-switching hints and input data. This process is the iterations in CKKS. The NTT and NPU are well pipelined. In ❸ , the final outputs of NPU are sent to SIMD to conduct the operations with data generated in ❶. With the proposed communication optimization, the data generated in ❶ is stored in the Polynomial Buffer. SIMD can read the needed data from the buffer directly, which avoids writing and reading DRAM.

4 EVALUATION

4.1 Experimental Methodology

In this section, we evaluate the Chiplever against the conventional full design-and-build approach. We emulate the scenario that Chiplever introduces the extension and integrates it into the non-FHE-tailored chiplet-based system. While the full design-and-build approach straightforwardly integrates a FHE accelerator. As the Chiplever workflow is proposed for general FHE, we evaluate two different representative FHE schemes, CKKS[13] and TFHE[12].

Modeled system. We implemented the synthesized RTL of the Chiplever extension in Verilog. The design is synthesized using Synopsys Design Compiler with the TSMC 14nm library, and the clock rate is set to 1 GHz. To evaluate micro-architectural behaviors of the Chiplever extension, we build a cycle-accurate simulator. The non-FHE-tailored chiplet-based system is similar to Figure 2, which has widely used NPU and SIMD chiplets besides CPU and HBM. The NPU is similar to with 16×1024 PEs[18][19]. The SIMD is equipped with 64 SIMD-128 cores. The HBM3[20] is adopted. The chiplets in the system are connected by the high-speed interposer[14].

Extension Configuration. The Chiplever introduces an extension as shown in Figure 5 with the configurable butterfly units. We set 64 butterfly units, with each performing 128-element NTT. We set the Polynomial buffer as 512KB for intermediate results reuse.

Table 1: FHE parameters.

FHE Scheme	Parameters
CKKS	$N = 10^{12}$, $Log(Q) = 109$
	$N = 10^{14}$, $Log(Q) = 438$
TFHE	$n = 500$, $N = 1024$, $L = 3$
	$n = 630$, $N = 2048$, $L = 3$

Benchmarks. We select different FHE schemes for evaluation. TFHE and CKKS are two representative FHE schemes. TFHE is a FHE scheme that supports homomorphic Boolean algebra with the fastest bootstrapping. CKKS is a popular approximate FHE scheme. We specifically employed the advanced version RNS-CKKS[21] with Residue Number System (RNS) that represents a single polynomial with wide coefficients as multiple polynomials with narrower coefficients. We evaluate the performance of homomorphic operations of two FHE schemes. For TFHE, we evaluate the TFHE homomorphic NAND operation (THOp) which involves homomorphic logic evaluation and bootstrapping. For CKKS, we evaluate the CKKS homomorphic multiplication (CHop) which involves key-switching operations. We evaluate two sets of parameters for homomorphic operations recommended by [12, 13] as shown in Table 1.

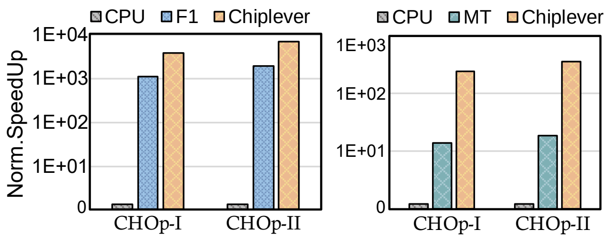

Figure 7: Performance comparison of Chiplever extension over baselines.

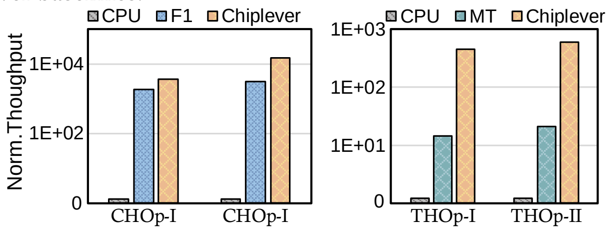

Figure 8: Throughput comparison of Chiplever extension over baselines.

Baselines. To evaluate the advanced design method of Chiplever, we conduct a comparison with the conventional full design-and-build design approach. We set the baselines designed in the conventional full design-and-build approach. MATCHA is a SOTA TFHE accelerator[9]. For TFHE application, we select MATCHA (MT) as the baseline model. For CKKS application, we select F1 as the baseline model[7]. We also set the CPU baselines Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz running the SOTA library. The CPU baselines run the SEAL[22](a CKKS library) and TFHE[12] library.

4.2 Evaluation Results

4.2.1 Performance Comparison. We first compare the performance of the Chiplever extension with the baselines. The speedups are normalized to the software solution on CPU. As MT is not designed for CKKS, we evaluate MT on the TFHE application. Similarly, we evaluate F1 on the CKKS application. In the comparisons of each baseline, the extension is scaled to the same area as the baseline for fair comparisons. As shown in Figure 7, on average, Chiplever achieves 2681.3×, 3.6×, and 18.2× speedup compared with CPU, F1, and MT respectively. With the same area consumption, Chiplever achieves higher performance. First, Chiplever conducts effective semantics matching and fully leverages the available resources of existing chipsets in the system. Second, Chiplever introduces the dedicated units in the extension to efficiently support the specific operations in FHE. Third, Chiplever fully exploits the resources in the system by aligning operations with the most suitable hardware and achieves efficient workload mapping by three-step mapping. Doing this achieves efficient acceleration for primitive operations.

4.2.2 Throughput Analysis. To evaluate the scheduling and pipeline management proposed by Chiplever, we compare the throughput of the Chiplever extension with the baselines. We evaluate the operations per second. As shown in Figure 8, on average, Chiplever achieves 4825.5×, 3.4×, and 29.5× speedup compared with F1, and MT. The main reasons are two-fold. First, Chiplever achieves an

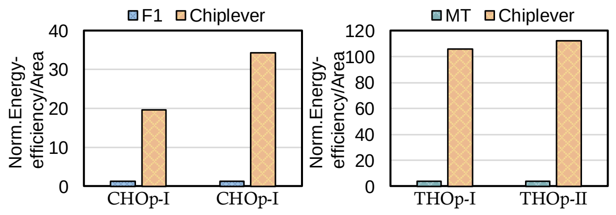

Figure 9: Energy-Efficiency/Area comparison of Chiplever extension over baselines.

efficient acceleration by designing dedicated extensions for special operations and exploiting the available resources for general operations for FHE workloads. Second, Chiplever proposes complier-level support to achieve efficient mapping and scheduling. The three-step mapping achieves the optimized pipeline and data communications, improving the overall throughput.

4.2.3 Area-efficiency. The total area of the Chiplever extension is 108.8 mm^2 under the given configuration in Section 4.1. To evaluate the energy efficiency To evaluate the efficiency under the same area consumption, we compare the area efficiency of Chiplever with baselines by dividing the energy efficiency by the respective consumed area. As shown in Figure 9, on average, Chiplever achieves 27.0× and 108.8 × energy efficiency compared with F1 and MT. This demonstrates that the proposed Chiplever is able to achieve efficient energy consumption compared with the conventional full-design approach. Chiplever is able to exploit the given area to boost the overall performance.

5 RELATED WORKS

FHE Accelerators. FHE suffers from high computational overhead which is typically 4 to 5 orders of magnitude slower than unencrypted computation. To address this, many FHE accelerators have been proposed. [7, 9, 11] propose ASIC design to improve computational efficiency. Some works also propose FPGA-based accelerators[8, 10]. As FHE requires large computing resources to achieve efficient computation, [5] propose a chiplet-based FHE accelerator to enhance the deployment in real-world scenarios.

Chiplet-based designs. Advanced packaging prompts a new design paradigm that many small chiplets can be assembled into a large system[23, 24]. Therefore, chiplet-based designs are widely embraced by applications in the post-Moore era. [25] propose a chiplet-based spatial accelerator to improve the scalability. The chiplet-based designs integrate multiple chips over an interposer through a NoC and show good scalability. The interposer can provide high-performance die-to-die interconnection. Some works also propose the dedicated interconnect chiplet or I/O chiplet in the system to improve communication efficiency between chiplets[16, 26]. The chiplet-based designs show many benefits over traditional monolithic designs.

6 CONCLUSION

In this paper, we propose Chiplever, a framework that enables the effortless extension of Chiplet-based system to enable the system not originally tailored for FHE to seamlessly support FHE. In addition to the dedicated function and scheduling support to ensure the transparency of the introduced extension, Chiplever also present the compiler support for efficient FHE workload mapping. The transparent extension seizes the opportunity to leverage the available resources in the room and achieves significant performance improvement over conventional full design-and-build approach.

7 ACKNOWLEDGEMENT

This work was supported by the National Natural Science Foundation of China under NSFC.62222411, 62025404, 92373206, 62204164.

REFERENCES

[1] Chiara Marcolla et al. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

[2] Lyubashevsky et al. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 1–23. Springer, 2010.

[3] Haozhe Zhu et al. Comb-mcm: Computing-on-memory-boundary nn processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[4] Zhanhong Tan et al. Nn-baton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture*, pages 1013–1026. IEEE, 2021.

[5] Aikata et al. Reed: Chiplet-based scalable hardware accelerator for fully homomorphic encryption. *arXiv preprint arXiv:2308.02885*, 2023.

[6] Kure et al. An integrated cyber security risk management approach for a cyber-physical system. *Applied Sciences*, 8(6):898, 2018.

[7] Axel Feldmann et al. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version). 2021.

[8] Yinghao Yang et al. Poseidon: Practical homomorphic encryption accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 870–881. IEEE, 2023.

[9] Lei Jiang et al. Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pages 235–240, 2022.

[10] Mingqin Han et al. coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1353–1358. IEEE, 2022.

[11] Nikola Samardzic et al. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 173–187, 2022.

[12] Ilaria Chillotti et al. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[13] Jung Hee Cheon et al. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*. Springer.

[14] Karim et al. A 0.65 mw/gbps 30 gbps capacitive coupled 10 mm serial link in 2.5 d silicon interposer. In *2014 IEEE 23rd Conference on Electrical Performance of Electronic Packaging and Systems*, pages 131–134. IEEE, 2014.

[15] Xiaohan Ma et al. Survey on chiplets: interface, interconnect and integration methodology. *Transactions on High Performance Computing*, 4(1):43–52, 2022.

[16] Suggs et al. The amd "zen 2" processor. *IEEE Micro*, 40(2):45–52, 2020.

[17] Soontorn Oraintara et al. Integer fast fourier transform. *IEEE Transactions on Signal Processing*, 50(3):607–618, 2002.

[18] Yu-Hsin Chen et al. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[19] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.

[20] Chae et al. A 4nm 1.15 tb/s hbm3 interface with resistor-tuned offset-calibration and in-situ margin-detection. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 1–3. IEEE, 2023.

[21] Jung Hee Cheon, , et al. A full rns variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, 2018, Revised Selected Papers 25*, pages 347–368. Springer.

[22] Microsoft SEAL (release 3.0). http://sealcrypto.org, October 2018. Microsoft Research, Redmond, WA.

[23] Fuping Li, Ying Wang, Yuanqing Cheng, Yujie Wang, Yinhe Han, Huawei Li, and Xiaowei Li. Gia: A reusable general interposer architecture for agile chiplet integration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[24] Fuping Li, Ying Wang, Yujie Wang, Mengdi Wang, Yinhe Han, Huawei Li, and Xiaowei Li. Chipletizer: Repartitioning socs for cost-effective chiplet integration. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 58–64. IEEE, 2024.

[25] Xiaochen Hao et al. Monad: Towards cost-effective specialization for chiplet-based spatial accelerators, 2023.

[26] Vidushi Goyal, , et al. Neksus: An interconnect for heterogeneous system-in-package architectures. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 12–21. IEEE, 2020.