# A Heterogeneous Computing Framework for Accelerating Fully Homomorphic Encryption

Cheng-Jhih Shih[1], Shih-Hao Hung[2,4,5], Ching-Wen Chen[3], Chiy-Ferng Perng[6], Meng-Chao Kao[6], Chi-Sheng Shih[2,5], and Tei-Wei Kuo[2,4,5]

[1] National Taiwan University, Taipei, Taiwan.
cs861324@gmail.com
[2] National Taiwan University, Taipei, Taiwan.
{hungsh,cshih,ktw}@csie.ntu.edu.tw
[3] National Taiwan University, Taipei, Taiwan.
wartytw@gmail.com
[4] Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE.
{ShihHao.Hung,TeiWei.Kuo}@mbzuai.ac.ae
[5] High Performance and Scientific Computing Center, NTU.
[6] Wistron Corporation.
{Alex_Perng,Andy_Gao}@wistron.com

**Abstract**

In the digital age, privacy is increasingly important. The General Data Protection Regulation (GDPR) [1] lays out rules for how data is collected and protected, posing challenges for many organizations. While there are several privacy preserving techniques, fully homomorphic encryption (FHE) stands out as the most mathematically secure. FHE enables computations on encrypted data without the need for decryption. However, it introduces significant computational overhead compared to non-encrypted computations. In this paper, we introduce a heterogeneous computing framework designed to accelerate fully homomorphic encryption.

This framework encompasses homomorphic evaluations on multiple platforms including CPU, GPU, and FPGA, paired with a tailored task scheduling algorithm. Each platform is equipped with comprehensive FHE functionalities and employs state-of-the-art implementations, allowing for standalone evaluation of FHE applications. The task scheduling algorithm strategically divides the computational tasks across the heterogeneous system, taking into account data transfer times to optimize application performance. Results show that the system reduces the latency effectively with additional computational platforms and provides more flexibility and accessibility of FHE for contemporary applications.

**Keywords:** Heterogeneous system, Homomorphic encryption, Hardware, accelerators, Parallel architectures, Security and privacy, Cryptography

## 1 Introduction

In today's data-driven world, where growing privacy concerns coincide with the rise of cloud computing that processes vast amounts of sensitive information, the demand for privacy preserving techniques has notably increased. Techniques such as TEE, MPC, and HE are frequently employed to address these concerns, and we provide brief descriptions of each.

Trusted Execution Environments (TEEs) provide a secure area within a main processor that ensures data and operations are protected from external threats. Meanwhile, Multi-Party Computation (MPC) allows multiple parties to collaboratively compute a function over their

The 7th International Conference on Mobile Internet Security (MobiSec'23), Dec 19-21, 2023, Okinawa, Japan, Article No.41

inputs without revealing those inputs to each other. Among these techniques, Fully Homomorphic Encryption (FHE) stands out as the most mathematically secure. It allows operations on encrypted data without the need to decrypt it, which is a significant advancement over standard encryption methods that expose data during processing.

Introduced in 2009 [2], FHE had its limitations in the beginning, but intensive research has transformed it into a practical tool for various tasks, finding applications in areas like deep learning and linear algebra. Both the private sector and governments are now heavily investing in its potential. However, the speed challenge persists. FHE is notably slower than working with non-encrypted data. One promising avenue to tackle this is the use of hardware solutions, specifically FHE accelerators [3, 4, 5, 6, 7, 8, 9].

However, while special-purposed FHE hardware accelerators may provide great speedups, their applications can be very limited and it can be expensive to design and manufacture such special chips. Thus, although various FHE accelerator chips have been proposed [3, 4, 5, 6], they have not been integrated into current production systems yet. On the other hand, graphical processing units (GPU) offer high computing power with affordable prices and are widely available in many systems, including mobile phones, which make them an alternative solutions for accelerating FHE [10, 11, 12, 13, 14, 15], although the acceleration from GPUs are less than special-purposed chips.

In this research work, we propose and develop an innovative heterogeneous FHE computing framework to accelerate the FHE computations with CPUs, GPUs, and accelerator chips. This framework automates the design space exploration process to provision and optimize system configurations for the targeted applications. This paper will present the proposed framework with experimental results to demonstrate its effectiveness. The remaining of this paper is organized as follows: **Sec 2** provides the necessary background and context; **Sec 3** describes our research methodology and approach; **Sec 4** presents the evaluation, experiments, and results; and **Sec 5** concludes the paper, highlighting key takeaways and possible future directions.

## 2 Background

Homomorphic Encryption is an encryption paradigm that allows for computations on encrypted data without the need to decrypt it. In addition to the usual `Enc()` and `Dec()` functions found in standard public key encryption, HE includes functions for homomorphic evaluations. Some HE schemes have a limited number of these evaluations. When the limits are reached, the data must be decrypted and then re-encrypted to continue with the computation. For unlimited computations, an extra step called *bootstrapping* is needed. Incorporating bootstrapping at various points during the computation allows the system to achieve fully homomorphic encryption.

Modern FHE schemes mostly rely on lattice-based cryptography. A common feature among them is the encoding of data vectors into plaintexts, facilitating SIMD (Single Instruction, Multiple Data) operations. However, the data types vary across schemes. BFV [16] and BGV [17] are designed to encode vectors of integers, while CKKS [18] allows for the encoding of real numbers. FHEW [19] and TFHE [20], on the other hand, encode binary data and support homomorphic boolean operations. All these schemes employ the RLWE (Ring Learning With Errors) assumption [21] to encrypt their inputs into ciphertexts.

While the proposed heterogeneous computing framework may be implemented to support the aforementioned algorithms, as a case study, we choose to accelerate the CKKS scheme in the open-source Lattigo libraries. The following subsections further discuss CKKS, mention the related works on FHE accelerators, and survey the challenges to scheduling tasks on heterogeneous computing systems.

## 2.1 The CKKS Scheme

Consider the CKKS scheme, a vector of complex data from $\mathbb{C}^{N/2}$ is first encoded as a polynomial in $Z/x^N + 1$, where $N$ is the polynomial degree. Subsequently, this polynomial undergoes encryption, resulting in a ciphertext in $Z_Q/x^N + 1$. Here, $Q$ is expressed as the product $q_0q_1...q_L$, with $L$ denoting the prime depth.

The encryption process can be depicted as:

$$ct = Enc(m) = (as + e + m, a) \tag{1}$$

where $a$ is a randomly sampled polynomial from $R_Q$. $s$ stands for the secret key, $e$ is a randomly sampled small error, and $m$ represents the encoded plaintext polynomial. For decryption, the process is:

$$Dec(ct) = ct[0] - ct[1] \cdot s = m + e \approx m \tag{2}$$

The result is close to $m$ when the error $e$ remains small relative to $m$. If $e$ grows too large, the decryption can fail.

The CKKS scheme supports six homomorphic operations, including `HAdd`, `HMul`, `Rescale`, and `Rotate`. As illustrated in Figure 1, these CKKS operators manipulate plaintext and/or ciphertext data in vectors, where each vector contains N/2 elements, and each element can hold an logQ-bit integer, which actually represents a floating-point number by dividing it with a scaling factor. The current efficient implementation of CKKS [22] utilizes the residue number system (RNS). This system divides a coefficient of a polynomial into multiple smaller numbers associated with smaller primes. Computations in the smaller prime domain ($q_0q_1...q_L$) are isomorphic to those in the original larger prime ($Q$). This approach allows for further parallelization of computations, leading to enhanced speedup opportunities. Furthermore, each prime level involves in rescaling the ciphertext to maintain it at a desired scale, without causing an exponential increase in plaintext. With each rescaling, the level drops and the ciphertext size decreases, resulting in faster computations.

In the CKKS scheme, bootstrapping refreshes the ciphertext level, ensuring that the data remains within permissible noise limits, thus preserving decryption accuracy. Essentially, bootstrapping homomorphically decrypts to reset the noise in the ciphertext. As depicted in Figure 2, bootstrapping involves multiple steps and consumes several levels, leaving the rest for homomorphic evaluations in various applications. Notably, several studies [23, 24, 25, 26] aimed at reducing their complexity to make FHE more practical for everyday use.

## 2.2 FHE Accelerators

Hardware accelerators for homomorphic encryption have evolved rapidly, with innovations across ASICs and FPGAs addressing various HE tasks. In the realm of ASIC-based solutions, designs such as [3, 4, 5, 6] have introduced algorithmic enhancements, programmability, and the capacity to handle expensive FHE programs. These ASIC implementations have realized speedups in the tens of thousands compared to CPUs, leading to a promising future for widespread FHE applications.

On the FPGA landscape, works like [7, 8, 9] offer diverse contributions. HEAX pioneers acceleration in keyswitching through its NTT engine while FAB and Poseidon specialize in bootstrappable FHE but adopt distinct methodologies. In performance metrics, FPGAs record speedups ranging from hundreds to thousands over CPUs, serving as a transitional bridge between CPUs and high-performance ASICs.

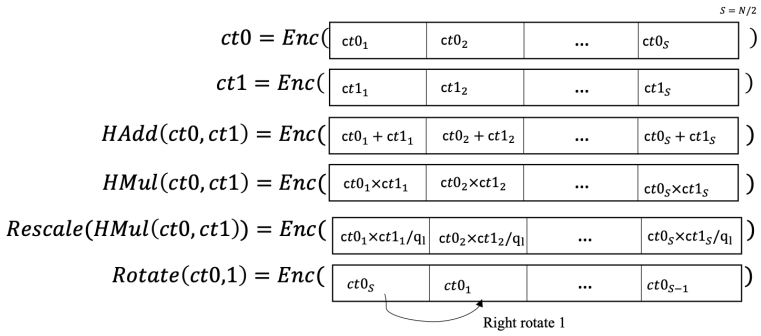


Figure 1: Homomorphic operations on vectors of data

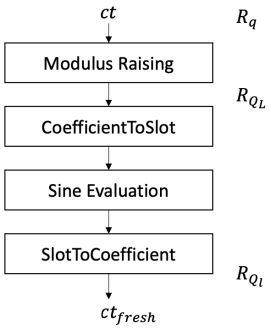


Figure 2: Boostrapping process, where $Q_L$ is maximum level and $Q_l$

In GPU-based research, [11, 12] tailors variants of NTT and algorithm for enhanced GPU performance, while [10, 13] focuses on the organization of overarching operations on GPUs. Specifically targeting neural networks, [14] optimizes HE CNNs on GPUs to provide the real use case. Finally, [15] investigates HE on multi-GPU setups, detailing its performance and scalability aspects.

### 2.3 Task Scheduling on Heterogeneous Systems

To enhance job dispatching efficiency across heterogeneous machines, it is imperative to employ well-suited task-scheduling algorithms. In pursuit of this objective, we have implemented two list-based algorithms, namely HEFT (Heterogeneous Earliest Finish Time) and PEFT (Predict Earliest Finish Time), within our established heterogeneous computing system.

HEFT [27], an extensively validated algorithm in the realm of heterogeneous computing, leverages the fundamental concepts of upward and downward rank, EST (Earliest Start Time), and EFT (Earliest Finish Time). It meticulously evaluates the tasks within the workflow, seeking the task with the highest upward rank. Once identified, this task is intelligently dispatched to an available machine with the smallest EFT. HEFT's efficacy lies in its ability to make informed decisions based on task properties and machine availability.

PEFT [28], an extension building upon the foundation of HEFT and incorporating a Look-ahead algorithm, introduces the concept of Predicted Earliest Finish Time. PEFT employs a 2D OCT (Optimistic Cost Table) to calculate the rank of each task, optimizing the selection process. The task with the highest rank, often indicative of the shortest expected completion time, is designated as the selected task. Notably, PEFT streamlines computation time by considering multiple layers of task successors.

## 3 Methodology

In this section, we present a platform-aware heterogeneous computing framework for accelerating FHE. In addition to the software-level compiler optimization, it also includes optimized hardware backends and profile-guided optimization (PGO) to exploit the full potential of the target platform and support hardware/software codesign. Figure 3 gives a high-level overview

of the framework. Given an application and user requirements, the proposed framework generates the data flow graph by analyzing the FHE operations invoked in the program and then compiles the FHE operations into HE microinstructions by considering the configuration of the target hardware platform. The HE microinstructions are scheduled to execute on a variety of accelerators. For each accelerator, the framework provide a backend to execute the HE microinstructions. In addition, the framework collects performance-related events during the runtime to facilitate PGO, which allows the compiler to improve the scheduling with the performance profile collected from previous runs. The profile is also useful for programming the FPGA and improving the system design.

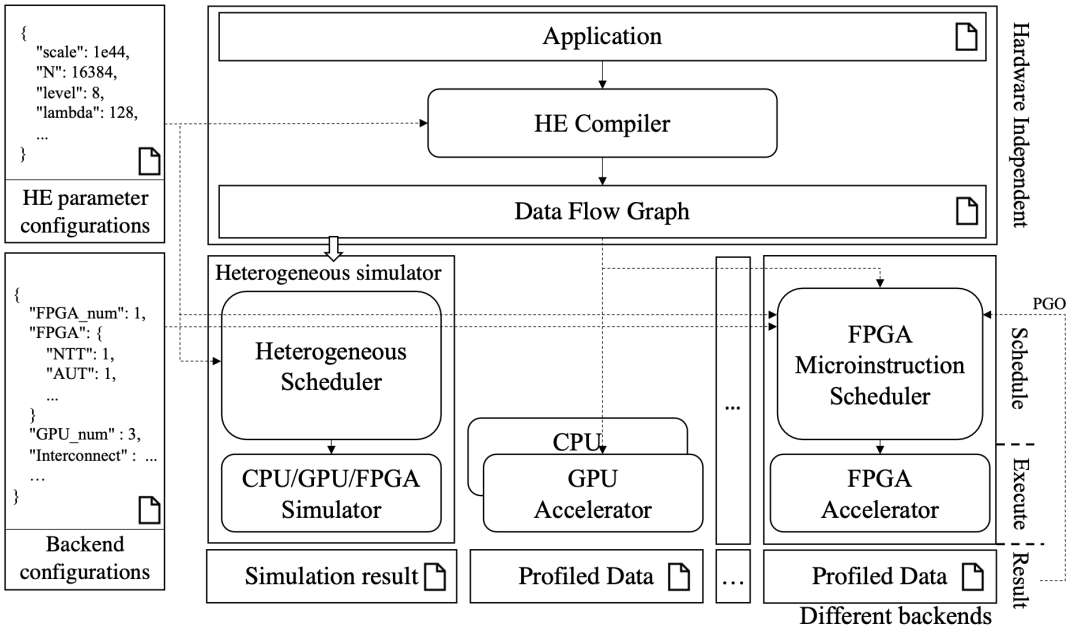


Figure 3: Proposed framework

The major components in the framework are:

1. **HE Compiler:** This component transforms the arithmetic operations of an HE application into a data flow graph, which is a directional acyclic graph (DAG) for HE operations. This involves efficiently packing data into vectors and conducting homomorphic operations on these vectors with correctness and security. The framework integrates this compiler as a plug-gable layer, drawing from various works such as EVA [29], CHET [30], and nGraph-HE [31] to facilitate the generation of the data flow graph.

2. **Heterogeneous Scheduler:** The heterogeneous scheduler divides the HE operations into different partitions with the overall objective of minimizing total execution time across the heterogeneous system. It distributes the workload considering the cost of different accelerators and their interconnects, providing efficient coarse-grained scheduling. More insights on the Heterogeneous Scheduler can be found in Section 3.1.

3. **HE Microinstruction Scheduler:** This scheduler further expands the HE operations into HE microinstructions and schedules them considering the hardware. This step opti-

mizes the program through fine-grained scheduling, taking into account different system architectures. Section 3.2 elaborates on the HE Microinstruction Scheduler.

4. **Accelerator Backends:** These components are responsible for executing HE operations on designated systems. We have developed HE accelerators for FPGA and adapted modified codes from [32] and [13] for the integration of CPU and GPU systems, respectively. Section 3.3 provides detailed information about each platform.

5. **Profile-guided optimization**: The framework includes profiling facilities to extract performance-related events and metrics to support profile-guided optimization on microinstruction scheduling and system design. The profiled data are fed back to the scheduler to optimize the schedule, as discussed in Section 3.2 and are also used to calibrate the timing models in the simulator to improve the simulation results.

## 3.1 Heterogeneous Scheduler

Fig 4 shows the organization of the heterogeneous system. This system comprises a typical node on a server host, equipped with a CPU and four PCIe slots. Each slot can accommodate either an FPGA or a GPU accelerator.

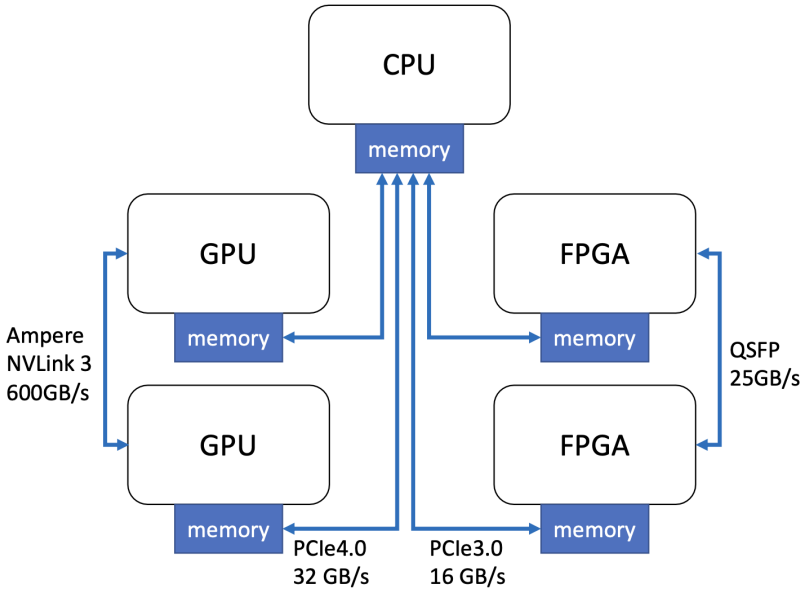


Figure 4: The heterogeneous system

The scheduler divides the data flow graph across the heterogeneous systems, as depicted in Fig 4. To achieve this, the scheduler requires:

1. *Task Cost on Different Systems*: We determine this cost through direct measurements of real systems. For various applications, we create tables for different levels and operations.

2. *Weighted DAG*: The weighted Directed Acyclic Graph (DAG) is derived from the data flow graph. The edges of this graph are calculated by dividing the ciphertext size of the node by a reference bandwidth.

3. *Communication Bandwidth*: This represents a two-dimensional matrix detailing the bandwidth between two different systems. As an illustration, in Fig 4, the GPU-to-GPU bandwidth is 600GB/s, while the FPGA-to-CPU bandwidth is 16GB/s. If a direct link is absent between two systems, we choose the minimal bandwidth from all available routes connecting the two nodes. For instance, the bandwidth between FPGA and GPU is set at 16GB/s. After filling up the matrix, it is then normalized by the reference bandwidth.

In Fig. 5, we illustrate an example of the required inputs. It's important to note that we did not include the latency matrix since its impact is negligible compared to data transfer time and computation time. Once these inputs are established, we apply a scheduling algorithm to them. The algorithm is modular, and for our evaluation, we utilize the basic HEFT [27]. The algorithm determines the start and end times for each task across the heterogeneous systems. The system with the latest end time then dictates the overall execution time for the entire heterogeneous system.

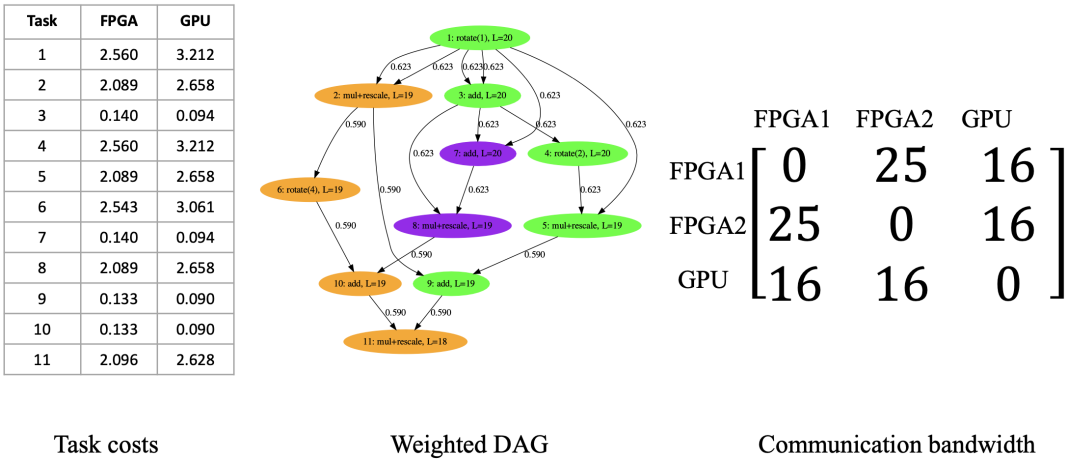


Figure 5: Inputs of the scheduling algorithm. The example depicts results after applying the HEFT algorithm to a setup of 2 FPGAs and 1 GPU. Green and orange tasks are scheduled on the FPGAs, while the purple task is scheduled on the GPU.

## 3.2 Generating and Scheduling HE Microinstructions

We reference the state-of-the-art CKKS algorithm [25] to implement HE operations. Each homomorphic operation is composed of combinations of HE microinstructions, which are essentially polynomial operations, such as the number theoretic transform (NTT), and include polynomial multiplication and addition. On systems like CPUs and GPUs, these polynomial operations are executed using either the AVX core or the CUDA core collaboratively. Consequently, they cannot process two distinct microinstructions simultaneously. As depicted in Figure 3, neither the CPU nor the GPU is equipped with a scheduler. On the other hand, platforms like FPGAs or ASICs have distinct functional units for different polynomial operations. This design enables the simultaneous execution of microinstructions. Additionally, the FPGA or ASIC requires additional scheduling for load/store operations to and from the scratchpad, as illustrated in Figure 7.

Precisely scheduling microinstructions could yield millions of such instructions and would be time-consuming. As an alternative, we leverage static controls to manage the data flow between various functional units, carrying out the execution of HE operations, such as `HMul`. This strategy might simultaneously engage multiple functional units (e.g., `polyadd` and `polymul`). The criteria for scheduling are straightforward: an operation is issued if (1) all its input data is stored on-chip and there is available space for the output, and (2) the functional units are unoccupied. Regarding the eviction policy, each operation receives an ID based on the user-defined sequence of HE operations. The operation node with the highest ID among its children is selected for eviction.

Obviously, the aforementioned scheduling is highly dependent on the flow of tasks and data. Thus, we place probes in the framework to profile the target programs and extract the control and data flows during the run time, in case the compiler cannot accurately predict the program behavior. The added profiling facility enables profile-guided optimization (PGO), allowing the developer and the compiler to identifies hotspots in the execution path and fine-tune the schedule with program traces from previous runs. In addition, the profile is also useful to the exploration of the design space for FPGA-based FHE accelerators as well as the entire system architecture.

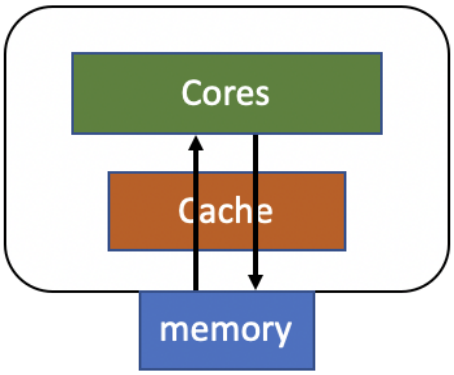


Figure 6: CPU/GPU FHE system.

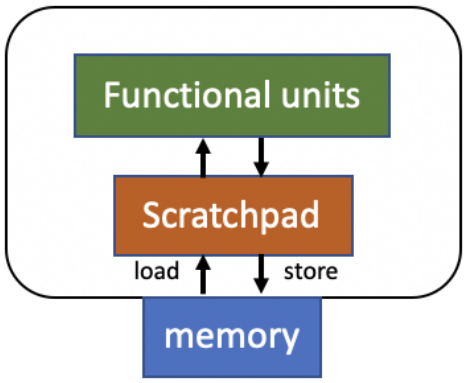


Figure 7: FPGA FHE system.

### 3.3 Accelerator Backends

For the CPU/GPU FHE system, We build upon the CPU [32] and GPU [13] works. Given the challenges of managing cache in CPU and GPU architectures, as illustrated in Fig 6, results are consistently stored to global memory following each homomorphic operation. However, operations might still benefit from cache usage.

Fig 7 illustrates the organization of the FPGA system. The system features a global memory that stores the initial data. Different from the CPU/GPU FHE system, there are microinstructions loading/storing data to/from on-chip scratchpad. On-chip data can be streamed into functional units for HE microinstruction execution. Below, we provide a brief overview of the functional units required for carrying out HE operations.

1. *Addition and Multiplication Unit*: These units handle element-wise polynomial addition and multiplication, functioning as basic arithmetic circuits without the need for buffers.

2. *Number Theoretic Transform (NTT) Unit*: This unit uses a computational method similar to the Fast Fourier Transform (FFT) but operates under modular arithmetic. We adapted the design from [33] and further decomposed the NTT into three recursive steps. This strategy balances the FPGA's resource constraints while targeting optimal performance.

3. *Automorphism Unit*: Designed primarily for data rearrangement, this unit is used for the `Rotate` operation. We've allocated large buffers to store an entire polynomial on-chip, enabling rearrangement without off-chip accesses.

4. *Residue Number System (RNS) Unit*: During HE computations, polynomial multiplication-and-accumulation operations are common, particularly in keyswitching. In this unit, we've integrated the multiplication and addition units and paired them with a buffer for temporary storage.

# 4 Evaluation

We evaluate the proposed framework in this section. First, we constructed an experimental heterogeneous computing environment with multiple GPUs and FPGAs, as described in Section 4.1 to validate that the proposed framework is capable of exploiting all the processor units in the system. Then, we measure the performance of the proposed framework with a set of HE application benchmarks to evaluate the effects of optimized scheduling and explore the system design space in Section 4.2.

## 4.1 Environmental Setup

In our FPGA design, we employ Verilog and utilize Xilinx Vivado 2022.2 for synthesis, achieving an operational frequency of 300 MHz. We deploy our design on the Xilinx Alveo U280 FPGA card, equipped with 8 GB of HBM2, offering a bandwidth of up to 460 GB/s.

For our CPU and GPU configuration, the host runs Ubuntu 20.04 with an AMD Ryzen Threadripper 3970X 32-Core Processor operating at 2.20 GHz. This setup is equipped with an RTX3090 GPU using the CUDA 11.3 library.

In terms of system interconnects, we base our architecture on the following links:

1. A PCIe x16 Gen 4 link between the CPU and FPGA.

2. A PCIe x16 Gen 3 link connecting the CPU and GPU.

3. NVLink 3rd Gen for inter-GPU communication [34].

4. QSFP28 x2 links for inter-FPGA communication [35].

Our evaluations encompass various combinations of FPGAs and GPUs; consequently, bandwidths are adjusted according to the configurations. We acquire results from single FPGA, GPU, and CPU setups and simulate the heterogeneous configurations.

## 4.2 Resulted Performance and Discussions

We used three types of applications, i.e. inference, logistic regression, and bootstrapping, to benchmark the performance of the proposed framework. For homomorphic inference, we report performance inferencing the MNIST and Cifar10 described in LoLa [36]. We further

evaluate homomorphic logistic regression as outlined in [37], employing L=38 and conducting 4 iterations plus one bootstrapping, while presenting the time required per batch for a single iteration. Lastly, we evaluate the bootstrapping process, refreshing a ciphertext at the lowest level using parameters with N=65536 and L=38.

We first evaluate the efficacy of the scheduling algorithm, as depicted in Table 1. The experimental setup involved 2 FPGAs and GPUs with a host CPU. Compared to the HEFT scheduling algorithm, the naive method randomly selects an unoccupied device to execute the task, transferring all necessary inputs for that task to the device before execution. We notice that for low-depth tasks without bootstrapping, like MNIST and Cifar10, the naive method works comparably to HEFT. However, for programs including bootstrapping (long-depth DAG), its performance is hindered by unbalanced workloads across the heterogeneous system, as child tasks must await the completion of all their parent tasks, which may include tasks from slower devices.

Table 1: Latency (ms) and speedup of FHE applications using different scheduling. The experimental system contains 2 FPGAs and GPUs.

| **Scheduling** | **MNIST** | **Cifar10** | **LR** | **Bootstrapping** |
|---|---|---|---|---|
| Naive | 8.7 (1.00X) | 9123.5 (1.00X) | 316.2 (1.00X) | 436.2 (1.00X) |
| HEFT | 8.2 (1.06X) | 8767.8 (1.04X) | 235.1 (1.35X) | 185.9 (2.35X) |

We then report the latency of FHE applications on different system configurations in Table 2. There is a performance difference of 2x-5x between FPGA and GPU across different applications. For applications without bootstrapping, like MNIST and Cifar10, the FPGA tends to exhibit superior performance. Conversely, for logistic regression and bootstrapping, the excessive off-chip memory access for bootstrapping data and limited on-chip memory on FPGA cause the wait time for memory to dominate, hence the performance difference is not significant.

Table 2: Latency (ms) of FHE applications on different system configurations

| | **Design** | **MNIST** | **Cifar10** | **LR** | **Bootstrapping** |
|---|---|---|---|---|---|
| | 1-FPGA | 19.3 | 26524.5 | 837.0 | 688.3 |
| | 1-GPU | 81.5 | 59569.4 | 1150.7 | 769.1 |
| With host CPU | 2-FPGA | 9.6 | 12697.2 | 411.4 | 338.9 |
| | 4-FPGA | 5.2 | 6487.5 | 208.2 | 181.5 |
| | 2-GPU | 39.9 | 28677.8 | 569.8 | 381.6 |
| | 4-GPU | 20.7 | 14542.1 | 291.9 | 207.4 |
| | 1-FPGA & 3-GPU | 11.1 | 10952.2 | 256.4 | 188.7 |
| | 2-FPGA & 2-GPU | 8.2 | 8767.8 | 235.1 | 185.9 |
| | 3-FPGA & 1-GPU | 6.4 | 7362.9 | 219.3 | 176.5 |

In multi-FPGA and GPU settings, the speedup is almost 2x or 4x (with additional aid of CPU) when the number of FPGAs or GPUs is doubled or quadrupled. This is due to each application being divisible into thousands of smaller tasks, with their differences being minimal, leading to a balanced workload in the data flow graph. Additionally, the communication to computation ratio is also low, around 1 to 10, resulting in low overhead from data transfer. In the heterogeneous settings, it's apparent that additional hardware contributes to reducing the

program's latency. Through the scheduling results, we found that every device nearly ends at the same time, and with high utilization across all devices. This demonstrates the effectiveness of the scheduling algorithm and its capacity to balance the workloads across heterogeneous systems.

# 5 Conclusion

We have developed a heterogeneous computing framework aimed at accelerating fully homomorphic encryption. This framework encompasses extensive FHE functionalities across various platforms, including CPU, GPU, and FPGA. Additionally, it features a heterogeneous system simulator, complemented by a heterogeneous scheduler. Our evaluation across several applications demonstrates that the system effectively reduces latency with the integration of additional computational platforms. This allows various users to deploy a heterogeneous system for FHE tailored to their specific conditions. While GPUs are more accessible and cost-effective, FPGAs, although offering superior performance, require a steeper learning and utilization curve. Even utilizing a single CPU, the application can still operate effectively. Overall, this provides a robust solution that affords users the flexibility to navigate between cost, ease of use, and computational power, ensuring that secure computations via FHE can be effectively achieved under a variety of system configurations and user expertise levels.

# Acknowledgements

This work was supported in part by the National Science and Technology Council, Taiwan, under Grant NSTC 112-2634-F-002-002-MBK and 110-2221-E-002-099-MY2, and by a Wistron research grant.

# References

[1] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[2] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[3] Brandon Reagen, Wooseok Choi, Yeongil Ko, Vincent T. Lee, Gu-Yeon Wei, Hsien-Hsin S. Lee, and David M. Brooks. Cheetah: Optimizations and methods for privacypreserving inference via homomorphic encryption. *ArXiv*, abs/2006.00505, 2020.

[4] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 238–252, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 173–187, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. Bts: An accelerator for bootstrappable fully homomorphic encryption. In

*Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 711–725, New York, NY, USA, 2022. Association for Computing Machinery.

[7] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1295–1309, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 882–895, 2023.

[9] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. Poseidon: Practical homomorphic encryption accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 870–881, 2023.

[10] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. *IEEE Access*, 9:98772–98789, 2021.

[11] Pedro Geraldo M. R. Alves, Jheyne N. Ortiz, and Diego F. Aranha. Faster homomorphic encryption over gpgpus via hierarchical dgt. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 520–540, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[12] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):70–95, May 2018.

[13] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):114–148, Aug. 2021.

[14] Ahmad Al Badawi, Chao Jin, Jie Lin, Chan Mun, Sim Jie, Benjamin Tan, Xiao Nan, Aung Khin, and Vijay Chandrasekhar. Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. *IEEE Transactions on Emerging Topics in Computing*, PP:1–1, 08 2020.

[15] Ahmad Al Badawi, Bharadwaj Veeravalli, Jie Lin, Nan Xiao, Matsumura Kazuaki, and Aung Khin. Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, PP:1–1, 09 2020.

[16] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[17] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[18] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[19] Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33:34–91, 2019.

[21] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23,

Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[22] Jung Cheon, Han Kyoohyung, Andrey Kim, Miran Kim, and Yongsoo Song. *A Full RNS Variant of Approximate Homomorphic Encryption: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers*, pages 347–368. 01 2019.

[23] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384, Cham, 2018. Springer International Publishing.

[24] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 34–54, Cham, 2019. Springer International Publishing.

[25] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings*, page 364–390, Berlin, Heidelberg, 2020. Springer-Verlag.

[26] Charanjit S. Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 491–520, Cham, 2022. Springer International Publishing.

[27] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[28] Hamid Arabnejad and Jorge G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.*, 25(3):682–694, mar 2014.

[29] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 546–561, New York, NY, USA, 2020. Association for Computing Machinery.

[30] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 142–156, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, page 3–13, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Lattigo v4. Online: https://github.com/tuneinsight/lattigo, August 2022. EPFL-LDS, Tune Insight SA.

[33] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[34] Nvlink and nvswitch performance. Online: https://www.nvidia.com/en-us/data-center/nvlink/.

[35] Alveo u280 data center accelerator card specifications. Online: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html#specifications.

[36] Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. Low latency privacy preserving inference. In *International Conference on Machine Learning*, 2019.

[37] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):9466–9471, Jul. 2019.