2025 IEEE International Symposium on High Performance Computer Architecture (HPCA) | 979-8-3315-0647-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/HPCA61900.2025.00089

# Anaheim: Architecture and Algorithms for Processing Fully Homomorphic Encryption in Memory

Jongmin Kim, Sungmin Yun, Hyesung Ji, Wonseok Choi, Sangpyo Kim, and Jung Ho Ahn
Seoul National University
{jongmin.kim, sungmin.yun, kevin5188, wonseok.choi, vnb987, gajh}@snu.ac.kr

***Abstract*—Fully homomorphic encryption (FHE) is a promising solution for privacy-preserving cloud computing as it enables computations on sensitive data without any risk of data leakage. Despite the significant attention FHE has received, substantial computation and memory demands make it hardly practical for real-world applications. We propose a readily available and practical hardware solution to tackle this problem by using GPUs. GPUs have adequate computational and memory resources to handle complex operations in FHE, including number-theoretic transform (NTT), on which most prior work has deeply focused. However, through detailed analyses, we discover that the performance bottleneck on GPUs is primarily due to simpler element-wise operations, which are limited by off-chip memory (DRAM) bandwidth. Motivated by these observations, we develop Anaheim, a processing-in-memory (PIM) architecture for FHE. We develop optimized FHE execution flows and an end-to-end software framework for using PIM with GPUs. Also, we design a versatile PIM unit that handles various modular integer arithmetic PIM instructions, along with an efficient data mapping and associated PIM execution algorithms that minimize data access overhead by leveraging the internal structure of DRAM. Our concerted efforts substantially enhance the performance and energy efficiency of various FHE workloads on GPUs.**

## I. Introduction

Fully homomorphic encryption (FHE) is a disruptive technology in cloud computing that fundamentally blocks the possibility of infringement on user confidentiality and privacy. As data encrypted with FHE can be processed directly without decryption, cloud servers can provide useful services without ever identifying the data contents. A number of FHE variants exist but CKKS [18] is unique in its ability to handle real (or complex) numbers, which is adequate for private machine learning tasks. CKKS also offers the highest computational throughput among FHE schemes [6].

However, significant acceleration efforts must precede to render FHE more practical. Even with CKKS, processing encrypted data is orders of magnitude more expensive in terms of computational and memory demands than processing plain data [39], which become major hurdles in its adoption. For example, an FHE implementation of ResNet20 inference [49] takes over 37 minutes to process on a single CPU thread, which is over 500,000× slower than its unencrypted counterpart (< 5ms). Also, it requires dozens of GBs of memory space to perform inference with the 1MB (FP32) model.

Thus, we explore hardware acceleration opportunities for CKKS, which can be easily extended to other popular FHE variants that share similar structures, including BGV [10] and BFV [11], [27], as exemplified in [42].

The high degree of parallelism inherent in FHE operations (ops) [47] make GPUs an attractive solution for hardware acceleration. A recently developed GPU library, Cheddar [44], reduced the ResNet20 inference latency to 1.5 seconds with a single GPU, which is over 1,500× faster than the original CPU implementation [49].

Detailed analyses of FHE workload characteristics on GPUs uncover a critical source of performance bottleneck that prior work has overlooked. While prior architecture studies [3], [45]–[47], [68], [69], [71], [72], [78] have focused deeply on the compute-intensive ops of FHE CKKS, such as number-theoretic transform (NTT), we identify that FHE workloads on GPUs are severely bottlenecked rather by simple element-wise ops (§IV). They exhibit extremely low arithmetic intensity and are bottlenecked by limited off-chip DRAM bandwidth, drawing a hard line for hardware acceleration of FHE.

To break through this limitation, we propose **Anaheim**, an in-DRAM processing-in-memory (PIM) architecture and algorithms to handle various element-wise CKKS ops fast and efficiently by utilizing the high internal bandwidth of DRAM chips. First, we carefully organize FHE execution flows to make them adequate for PIM offloading and develop an end-to-end software framework to handle the collaborative execution of the GPU and PIM devices. We fuse and reorder ops such that we can offload large computational blocks entirely composed of element-wise ops to PIM and quantitatively show potential reductions in execution time and GPU-side DRAM access attainable from PIM offloading (§V).

We design a specialized PIM unit and associated PIM execution methods to fully exploit bank-level parallelism inside DRAM devices (§VI) and propose two microarchitecture variants for practical PIM realization. First, near-bank PIM places the PIM units next to each DRAM bank following the approach of real PIM devices, such as HBM-PIM [54] and GDDR6-AiM [53]. We also propose an alternative design based on custom-HBM [14], which exploit increased bandwidth from the DRAM dies to the logic die of HBM [80] by adopting advanced packaging technologies [75] available for future HBMs. We place the PIM units on the logic die of HBM for this variant, which we find especially practical as it avoids using highly restrictive DRAM process nodes. As shown with real GPUs in MI100-PIM [43], which integrates an AMD MI100 GPU with HBM-PIM [54] stacks, Anaheim can augment existing GPUs in a modular manner, which requires only manageable hardware modifications to the GPU.

979-8-3315-0647-6/25/$31.00 ©2025 IEEE
DOI 10.1109/HPCA61900.2025.00089

Anaheim significantly boosts the end-to-end performance and energy efficiency of diverse CKKS workloads, leading to 1.62–3.14× improvements in energy-delay product (EDP) across various PIM architectures. At the level of individual PIM instructions, Anaheim achieves 1.65–10.3× speedups and 2.63–17.4× energy efficiency improvements. Our key contributions are as follows.

- We debunk a common fallacy that the performance of FHE operations on GPUs suffers primarily from compute-intensive kernels, such as NTT.
- We provide an in-depth analysis of how the architectural characteristics of GPUs lead to different algorithm choices compared to custom ASIC designs. We discover that this eventually results in element-wise ops becoming crucial sources of DRAM bandwidth bottleneck on GPUs.
- We develop a software framework that facilitates PIM integration with GPUs and enhance the applicability of PIM on FHE op sequences through extensive reordering and kernel fusion.
- We develop an efficient PIM unit microarchitecture, data layout, and PIM execution methods tailored to various element-wise ops in FHE CKKS.

## II. Background

Polynomials are denoted in lowercase (e.g., $a$), while capital letters (e.g., $L$) represent constant numbers, except for $\alpha$ and iterators (e.g., $i$). Bold letters (e.g., $\mathbf{u}$) represent vectors. We summarize key symbols and notations in Table I.

### A. CKKS data objects and basic functions

CKKS operations (ops) are performed on polynomials in a polynomial ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ of degree $N$ (typically $2^{16}$) and modulus $Q$ (typically $\sim 2^{1300}$). The residue number system (RNS) [16] is employed to efficiently manage the large modulus $Q$; $L$ primes $Q_0, \cdots, Q_{L-1}$ satisfying $Q = \prod_{i=0}^{L-1} Q_i$ are selected and $a \in \mathcal{R}_Q$ is represented by a list of $L$ limbs $(a[0], \cdots, a[L-1])$, where $a[i] = a \bmod Q_i$. Then, a polynomial op can be performed in a limb-wise manner; e.g., polynomial multiplication (mult) can be performed as

$$a*b = (a[0]*b[0] \bmod Q_0, \cdots, a[L-1]*b[L-1] \bmod Q_{L-1}).$$

With RNS, we can view a polynomial as an $L \times N$ matrix of coefficients, each row of which corresponds to a limb.

A complex vector message $\mathbf{u} \in \mathbb{C}^{N/2}$ is first converted into an unencrypted polynomial $\langle \mathbf{u} \rangle \in \mathcal{R}_Q$, referred to as plaintext, through encoding. Then, we encrypt $\langle \mathbf{u} \rangle$ by using a random polynomial $a$, a secret $s$, and an error term $e$ to obtain the ciphertext $[\![\mathbf{u}]\!] = (b, a) \leftarrow (-a*s + \langle \mathbf{u} \rangle + e, a) \in \mathcal{R}_Q^2$, a pair of polynomials in $\mathcal{R}_Q$.

Various functions can be evaluated in the encrypted state. We introduce a few basic functions. $[\![\mathbf{u}]\!] = (b_1, a_1)$, $[\![\mathbf{v}]\!] = (b_2, a_2)$, and $\langle \mathbf{p} \rangle = p$. $\odot$ represents element-wise mult.

- HAdd$([\![\mathbf{u}]\!], [\![\mathbf{v}]\!]) \rightarrow [\![\mathbf{u}+\mathbf{v}]\!] = (b_1+b_2, a_1+a_2)$.
- PMult$([\![\mathbf{u}]\!], \langle \mathbf{p} \rangle) \rightarrow [\![\mathbf{u} \odot \mathbf{p}]\!] = (b_1 \cdot p, a_1 \cdot p)$.
- HMult$([\![\mathbf{u}]\!], [\![\mathbf{v}]\!], \mathsf{evk}_\odot) \rightarrow [\![\mathbf{u} \odot \mathbf{v}]\!]$.
- HRot$([\![\mathbf{u}]\!], R, \mathsf{evk}_R) \rightarrow [\![\mathbf{u} \ll R]\!]$ ($\ll$: cyclic rotation).

TABLE I
Notations and symbols

| Notation | Explanation |
|---|---|
| $N$ | Polynomial ring degree. Typically $2^{16}$. |
| $Q, Q_i, L$ | Modulus, modulus primes, and # of $Q_i$. $Q = \prod_{i=0}^{L-1} Q_i$. |
| $P, P_i, \alpha$ | Additional modulus, additional modulus primes, and # of $P_i$. Used for extended modulus $PQ$. $P = \prod_{i=0}^{\alpha-1} P_i$. |
| $D$ | Decomposition number [34]. $D = L/\alpha$. |
| $\mathcal{R}_Q$ | Cyclotomic polynomial ring $\mathbb{Z}_Q[X]/(X^N+1)$. |
| $\langle \mathbf{u} \rangle$ | Plaintext. Encoding of $\mathbf{u} \in \mathbb{C}^{N/2}$, which is not encrypted yet. |
| $[\![\mathbf{u}]\!]$ | Ciphertext. Encryption of $\mathbf{u} \in \mathbb{C}^{N/2}$. $[\![\mathbf{u}]\!] = (b, a) \in \mathcal{R}_Q^2$. |
| evk | Evaluation key. Composed of $2 \cdot D$ polynomials in $\mathcal{R}_{PQ}$. |
| $L_{\text{eff}}$ | # of mult ops applicable to $[\![\mathbf{u}]\!]$ in between bootstrapping. |

While inter-ciphertext addition (HAdd) and plaintext-ciphertext multiplication (PMult) can be performed with relatively low computational complexity, inter-ciphertext multiplication (HMult) and ciphertext rotation (HRot) involve more complex op sequences and even require additional data objects called evaluation keys (evks). An evk uses additional modulus $P = \prod_{i=0}^{\alpha-1} P_i$ and comprises $2 \cdot D$ polynomials ($D$: decomposition number [34]) in the extended modulus $PQ$; each polynomial has $L + \alpha$ limbs. We refer a more detailed description of CKKS functions to prior work [16], [18], [34].

### B. Primary polynomial operations

We can compose any CKKS function with element-wise ops between polynomials, `ModSwitch`, and automorphism. `ModSwitch` is further broken down into a sequence of inverse NTT (INTT), basis conversion (BConv), and NTT.

**NTT**, an integer variant of the Fourier transform, facilitates polynomial mult. A naïve mult between two degree-$N$ polynomials in $\mathcal{R}_Q$ is equivalent to a (negacyclic) convolution incurring $\mathcal{O}(N^2)$ integer ops per limb. Performing NTT, which has a complexity of $\mathcal{O}(N \log N)$ exploiting the fast Fourier transform (FFT), prior to the mult converts it into $\mathcal{O}(N)$ element-wise mult. Henceforth, we assume that all polynomials are already NTT-applied unless otherwise specified.

**BConv** switches between modulus $Q$ and $PQ$ by generating the $P$-part ($\alpha$ limbs) from the $Q$-part ($L$ limbs) of a polynomial. Computing BConv is mostly equivalent to a matrix-matrix mult between a predefined $\alpha \times L$ BConv matrix and the $L \times N$ input ($L$ limbs of $N$ coefficients) to produce an $\alpha \times N$ output ($\alpha$ limbs). Polynomials must not be NTT-applied for BConv, resulting in a sequence of (1) INTT on $L$ input limbs, (2) BConv, and (3) NTT on $\alpha$ output limbs, which we refer to as `ModSwitch`. `ModSwitch` is used for a number of other combinations of input and output moduli.

**Automorphism** ($\phi_R$) is performed specifically for HRot, one of the most frequent basic functions. The data order of coefficients in each limb is permuted during automorphism. The permutation pattern is consistent among the limbs and is determined by the rotation distance $R$ in HRot.

Fig. 1 shows how these ops are used to evaluate HRot$([\![\mathbf{u}]\!], R, \mathsf{evk}_R)$, where $[\![\mathbf{u}]\!] = (b, a)$. HRot consists of five subsequences: `ModUp` → `KeyMult` → constant mult and

add (MAC) $\rightarrow$ automorphism $\rightarrow$ ModDown. Most of the ops are performed to multiply $a \in \mathcal{R}_Q$ with evk $\in \mathcal{R}_{PQ}^{2\cdot D}$ for the extended modulus $PQ$. Computation is attributed mostly to ModUp and ModDown, which are variants of ModSwitch for changing the modulus from $Q$ to $PQ$ and from $PQ$ back to $Q$. The other ops, element-wise ops and automorphism, show low computational complexity of $\mathcal{O}(N)$ per limb.

### C. Bootstrapping

Evaluating functions on a ciphertext reduces $L$ but $L$ cannot decrease indefinitely, necessitating bootstrapping, a defining feature of FHE, to restore $L$. At a high level, bootstrapping comprises homomorphic linear transforms and high-degree polynomial function evaluations. Linear transforms are especially costly, each involving dozens of evks and plaintexts. Thus, we discuss linear transforms in detail in the following sections. Despite numerous algorithmic enhancements [8], [9], [34], [51], [55], bootstrapping still accounts for a majority of the computation time in practical FHE applications [35], [41], [49], [52]. Thus, we can exploit bootstrapping performance as a proxy for FHE performance and adopt the following metric from prior work [45]:

$$T_{\text{boot,eff}} = (\text{Bootstrapping time})/L_{\text{eff}}.$$

$L_{\text{eff}}$ is the number of multiplicative ops applicable to a ciphertext just after bootstrapping before it requires another bootstrapping, which is used to account for bootstrapping frequency. We can increase $L_{\text{eff}}$ by increasing $D$. However, the evk size and the amount of computation increase for larger $D$ values, creating a trade-off with $L_{\text{eff}}$. Prior work [38], [45]–[47], [72] and FHE libraries [5], [26] typically utilize $D$ values smaller than eight. We use $D = 4$ by default.

### D. DRAM microarchitecture & processing-in-memory (PIM)

DRAM dies are organized as channels, ranks, bank groups, and banks (see Fig. 6). A DRAM bank consists of DRAM cell arrays, row and column decoders, write drivers, and I/O sense amplifiers (IOSAs). To access data in a row of a bank, we copy the entire row into the IOSAs (ACT), access a chunk of data (typically 256 bits) from the IOSAs using the column decoder (RD/WR), and restore the voltage level of the IOSAs for subsequent row access (PRE). Multiple chunks of data in a row can be accessed in between ACT and PRE commands for faster execution. DRAM devices handle requests on different banks concurrently to keep the shared I/O channel busy.

By exploiting this internal structure of DRAM, PIM aims to accelerate memory-bound ops by placing computational units inside DRAM devices and utilizing their high internal bandwidth. While the computational units can be placed at various locations, realized PIM products, such as UPMEM PIM [24], HBM-PIM [54], and GDDR6-AiM [53], placed them alongside DRAM banks.

## III. Hardware Limitations & Algorithm Choices

Prior FHE architectural studies have focused mostly on custom hardware designs. ASIC proposals [45]–[47], [71], [72], in particular, achieved remarkable performance enhancements.

However, ASIC solutions lack adaptability to evolving algorithms and are extremely costly to realize. As FHE is an emerging field, its algorithms and implementations continually evolve. Static ASIC architectures, tailored to specific parameter settings rather than general-purpose computation, are difficult to modify for new advancements. Also, the rapidly growing diversity of FHE accelerator architectures complicates the timely development of comprehensive software stacks, including compilers and software libraries, which are essential for integrating new algorithmic advancements with ASICs.

Instead, our work focuses on enhancing existing GPUs, which offer a readily available hardware solution for accelerating FHE with its high computational capability and well-established programming ecosystem.

### A. Limited GPU memory capacity & bandwidth

The key differences between prior ASIC proposals and GPUs (illustrated using NVIDIA A100 80GB) are as follows.

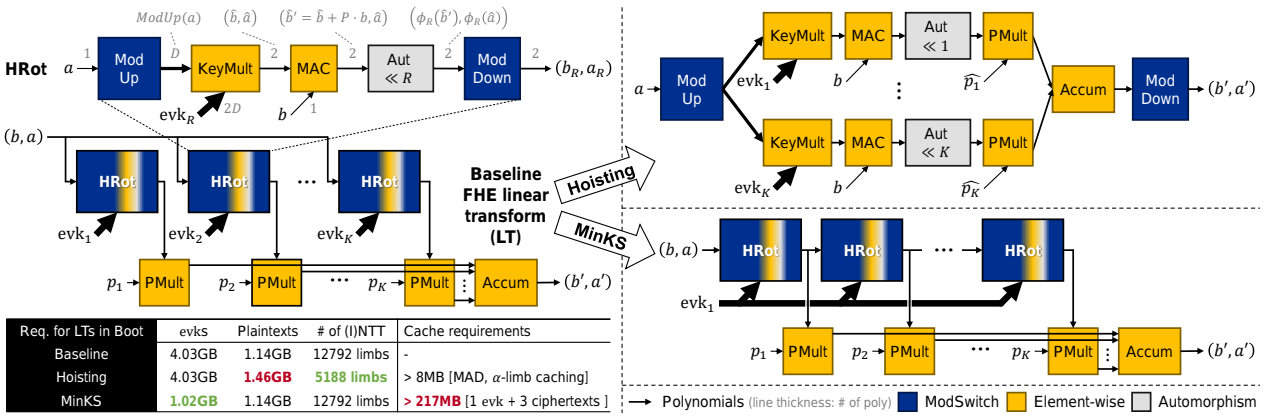
- **D1** 180–512MB of on-chip cache vs. 40MB L2 cache. As a polynomial can be as large as 17MB and an evk 136MB, GPUs cannot achieve the same level of cache data reuse as ASIC proposals. Thus, GPUs fall back to off-chip DRAM as their main source of memory access, which offer less than 2TB/s of bandwidth. In contrast, prior ASIC proposals benefited from massive caches and extreme cache bandwidth assumptions of 72–92TB/s.
- **D2** 25–170 TOPS of modular mult throughput vs. 19.5 TOPS of general integer mult throughput. As one modular mult involves a handful of instructions on GPUs, the effective throughput differences can reach $> 40\times$.
- **D3** ASIC proposals feature specialized datapath for complex data exchange patterns required for (I)NTT and BConv.

Although these ASIC proposals remain unrealized, recent advancements in GPU technology are narrowing the gap. For example, RTX 4090 features 41.3 TOPS of integer mult throughput (**D2**) and H100 includes a new network-on-chip (NoC) structure adequate for FFT data exchange patterns (**D3**).

In contrast to the consistent computational capability advancements, memory capacity and bandwidth of GPUs have plateaued due to technological constraints. For example, TSMC reported a mere 5.2% improvement in SRAM cell area from 5nm ($0.021\mu\text{m}^2$) [13] to 3nm ($0.0199\mu\text{m}^2$) [12] fabrication node. DRAM bandwidth also faces power limitations [62]. Thus, even the latest GPUs have at most dozens of MBs of L2 cache and several TB/s of DRAM bandwidth, leading to large effective bandwidth differences (**D1**) with the ASIC proposals.

### B. FHE linear transform: Hoisting vs. MinKS

Homomorphic evaluation of linear transform is a crucial component of FHE CKKS that accounts for over 60% of the bootstrapping time [38]. While there exist various linear transform methods [4], [37], we focus on the method based on diagonal packing [32] used for bootstrapping. Similar computational patterns are also commonly found in private deep neural network (DNN) workloads [37], [41], [49], [50]. Thus, we delve into linear transform algorithms in FHE CKKS.




| Req. for LTs in Boot | evks | Plaintexts | # of (I)NTT | Cache requirements |
|---|---|---|---|---|
| Baseline | 4.03GB | 1.14GB | 12792 limbs | - |
| Hoisting | 4.03GB | **1.46GB** | **5188 limbs** | > 8MB [MAD, $\alpha$-limb caching] |
| MinKS | **1.02GB** | 1.14GB | 12792 limbs | **> 217MB** [1 evk + 3 ciphertexts ] |

Fig. 1. HROT and linear transform op sequences (left). Optimized linear transform op sequences with hoisting and MinKS, which are incompatible with each other (right). The table compares the amount of evks/plaintexts and the number of (I)NTT ops required for a collection of linear transforms (CoeffToSlot [17]) in bootstrapping depending on the algorithms. The cache capacity requirements for hoisting is derived based on the $\alpha$-limb caching method from MAD [2].

We prepare $K$ plaintexts $\langle \mathbf{p}_i \rangle = p_i$ $(i \in [1, K])$ containing the data elements of the linear transform matrix, and perform $K$ PMULT and $K$ HROT evaluations[1] as the following:

$$[\![\sum_{i=1}^{K} (\mathbf{u} \ll i) \odot \mathbf{p}_i]\!] \leftarrow \sum_{i=1}^{K} \text{PMULT}(\text{HROT}([\![\mathbf{u}]\!], i, \text{evk}_i), \langle \mathbf{p}_i \rangle)$$

In this paper, we use a linear transform with $D = 4$ and $K = 8$ as a running example. Two optimizations, hoisting [8], [32] and minimum key-switching (MinKS) [32], [46], are widely used for efficient FHE linear transform but we cannot adopt both of them together [32]. The op sequences with and without these optimizations are shown in Fig. 1.

**Hoisting** extracts common ops in parallel HROT evaluations by reordering the op sequence to reduce the computational burden. As the HROT evaluations all start with the same `ModUp`, we can perform `ModUp` only once instead of $K$ times. We also apply hoisting to `ModDown` by performing PMULT and accumulation ops before `ModDown` to perform `ModDown` only once for the accumulated ciphertext. The reordering causes the PMULT and accumulation ops to be performed for the extended modulus $PQ$, increasing the size of each plaintext.

**MinKS** attempts to relieve the memory capacity and bandwidth burdens imposed by evks in linear transforms. Instead of using $K$ evks to obtain $\text{HROT}([\![\mathbf{u}]\!], 1, \text{evk}_1)$, …, $\text{HROT}([\![\mathbf{u}]\!], K, \text{evk}_K)$, using the property that $\text{HROT}([\![\mathbf{u}]\!], 2, \text{evk}_2) = \text{HROT}(\text{HROT}([\![\mathbf{u}]\!], 1, \text{evk}_1), 1, \text{evk}_1)$, MinKS uses only a single $\text{evk}_1$ to iteratively evaluate them.

The table in Fig. 1 shows the benefits of these algorithms when applied to a collection of linear transforms (CoeffToSlot [17]) inside CKKS bootstrapping. Although hoisting requires slightly larger plaintexts, it significantly reduces the number of `ModSwitch` ops, implied by a 2.47× reduction in the (I)NTT op count. In contrast, MinKS does not alter the amount of computation but requires 4× fewer evks.

[1]We do not discuss the baby-step giant-step linear transform algorithm [32] in the paper for conciseness despite using it whenever applicable.

### C. Algorithm choices based on hardware differences

While MinKS increases the reusability of evks, hundreds of MBs of on-chip cache (e.g., 217MB, Fig. 1), which is large enough to hold at least an evk (136MB) and several ciphertexts ($3 \times 27$MB), is required for an actual reuse. Prior ASIC solutions [45], [46] deployed up to 512MB of on-chip cache to utilize MinKS. Although MinKS requires more computation than hoisting, ASIC solutions have sufficiently high computational capabilities (**D2**, **D3**) such that the performance gains from reduced DRAM access for evks are much larger.

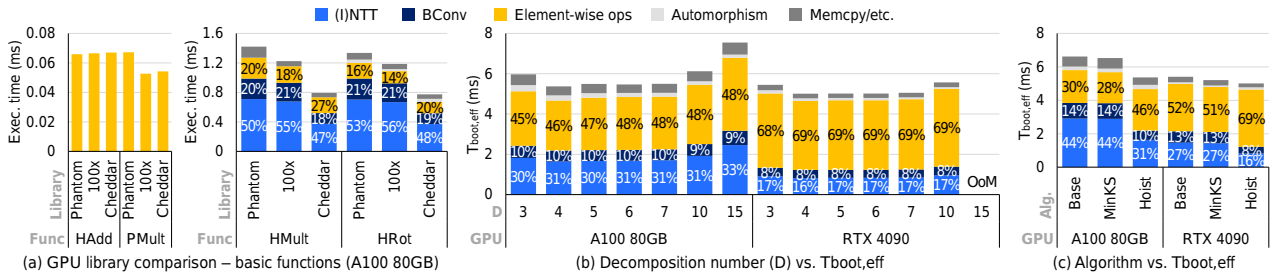
Due to **D1**, GPUs (or other conventional hardware) cannot adopt the same strategy. Instead, GPU implementations, such as 100× [38] and Cheddar [44], utilized hoisting to prioritize reducing the computational burden, which shows better performance compared to MinKS on GPUs (see Fig. 2c). *Thus, hereafter, we focus on the hoisting-based linear transform method, which is also suitable for Anaheim.*

## IV. DRAM BANDWIDTH BOTTLENECK ON GPUS

### A. GPU library enhancement

We employed Cheddar [44], a recently developed GPU library demonstrating 2.9–25.6× superior FHE workload performance compared to [28], [38], [63], to renew the workload characterization studies on GPUs. Fig. 2a shows the execution times of basic CKKS functions we measured on A100 80GB using Cheddar and open-source libraries, Phantom [77] and 100× [38]. Cheddar[2] shows 1.79× (resp., 1.73×) and 1.54× (1.53×) higher HMULT (HROT) performance compared to Phantom and 100×. These speedups come from the acceleration of compute-intensive (I)NTT and BConv, which get 1.80–1.81× and 1.73–1.75× faster by using Cheddar (vs. 100×). We also tested other open-source CKKS libraries, including Liberate.FHE [23] and Troy-Nova [56], but they show over 10× slower HMULT performance compared to Cheddar.

[2]As Cheddar adopts 32 bits of word size while the others 64 bits, we used 2× higher $L$ and $\alpha$ with Cheddar considering double-prime scaling [1], [45].



(a) GPU library comparison – basic functions (A100 80GB) (b) Decomposition number (D) vs. Tboot,eff (c) Algorithm vs. Tboot,eff

Fig. 2. (a) Execution time breakdown of basic CKKS functions (HADD, PMULT, HMULT, and HROT) measured on an A100 80GB GPU with three GPU libraries: Phantom [77], 100× [38], and Cheddar [44]. (b) $T_{\text{boot,eff}}$ breakdown using Cheddar, where we vary the decomposition number [34]. OoM refers to out-of-memory failure. (c) $T_{\text{boot,eff}}$ breakdown for $D = 4$ when using MinKS, hoisting (Hoist), or none of both (Base).

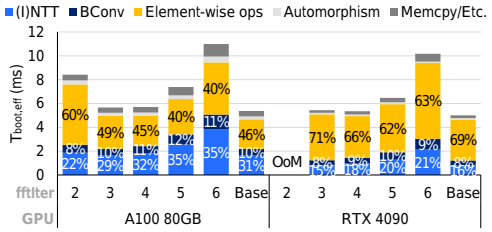



Fig. 3. $T_{\text{boot,eff}}$ breakdown using Cheddar [44], where we vary fftIter [2]. We use $D = 4$ and an fftIter mix of three and four by default (Base). OoM refers to out-of-memory failure.

### B. Element-wise operations dominate the bootstrapping time

The disparities in hardware and corresponding algorithm choices yield observations that challenge prevailing assumptions in FHE architectural research. We observe that, as hoisting and recent GPU hardware/software improvements lead to notable reductions in `ModSwitch` time, simple element-wise ops become the main source of bottleneck on GPUs, which have limited cache capacity, rather than (I)NTT or BConv.

We measured $T_{\text{boot,eff}}$ (§II-C) on A100 80GB and RTX 4090 with Cheddar. To account for different parameters, we varied the decomposition number ($D$) for the analysis by modifying $L$ and $\alpha$ values while keeping $N = 2^{16}$ and $\log PQ < 1623$ for the standard 128-bit security [19]. Fig. 2b shows that element-wise ops account for surprisingly huge portions in bootstrapping, reaching 45–48% on A100 80GB and even 68–69% on RTX 4090, regardless of the parameter choice.

Hoisting is the main reason behind these trends. If we apply MinKS instead of hoisting, which hardly results in speedups on GPUs, the portion of element-wise ops (28%) becomes similar to that in HMULT/HROT (20–27%) on A100 80GB (see Fig. 2c). As discussed in §III, hoisting (1) significantly reduces the number of `ModSwitch` ops and (2) performs more element-wise ops in the extended modulus (more limbs) to result in substantial increases in the portion of element-wise ops. Other algorithms, such as skipping [40], [55] and merging [2] `ModSwitch` evaluations, intensify these trends.

### C. Limitations of prior mitigation methods

**Higher fftIter**: We may reduce the cost of linear transforms inside bootstrapping by decomposing each linear transform matrix into many sparse matrices [15]. For example, MAD [2] defined the number of decomposed matrices as fftIter and used fftIter $= 6$. However, for each fftIter increase, $L_{\text{eff}}$ also drops, possibly resulting in worse $T_{\text{boot,eff}}$. Fig. 3 shows this trade-off well; increasing fftIter slightly reduces the portion of element-wise ops in bootstrapping, but the reduced $L_{\text{eff}}$ degrades the performance for fftIter $> 4$. We use an fftIter mix of three and four by default, which achieves the best $T_{\text{boot,eff}}$.

**Kernel fusion**: To reduce the time spent on element-wise ops, we can fuse multiple GPU kernels to perform as many ops as possible before storing the results back to DRAM. However, in practice, we continually load data, including evks, $b$, and $\hat{p}_i$'s (see Fig. 1), which limit the benefits of kernel fusion. Cheddar already includes state-of-the-art kernel fusion methods [38], which are clearly not sufficient to shift the needle.

### D. Potential acceleration opportunities for the operations

Accelerating these element-wise ops proves challenging as they have straightforward structures and are severely bottlenecked by limited DRAM bandwidth. Element-wise ops show less than 2 ops/byte of arithmetic intensity, while GPUs are best suited for 10–40 ops/byte based on Table III. Cheddar also failed to improve them as shown in Fig. 2a (HADD/PMULT).

Compared to A100 80GB, RTX 4090 with 2.1× higher peak integer throughput (41.3 vs. 19.5 TOPS) shows 2.0× and 1.4× higher performance for (I)NTT and BConv, implying that (I)NTT and BConv are compute-bound. Likewise, future hardware and software developments will further improve the performance of (I)NTT and BConv; however, element-wise ops remain bottlenecked without solving **D1**.

## V. ANAHEIM ALGORITHMS AND SOFTWARE STACK FOR PIM-FRIENDLY FHE

To break through this memory bandwidth wall GPU-based FHE acceleration faces, we turn to PIM to exploit the high internal bandwidth of DRAM as a solution.

### A. Offloading element-wise operations to PIM

A naïve solution to **D1** could involve utilizing higher DRAM bandwidth. We measured the performance of an optimized linear transform with hoisting (Fig. 5, $K = 8$) on an A100 GPU and estimated the performance when 4× higher

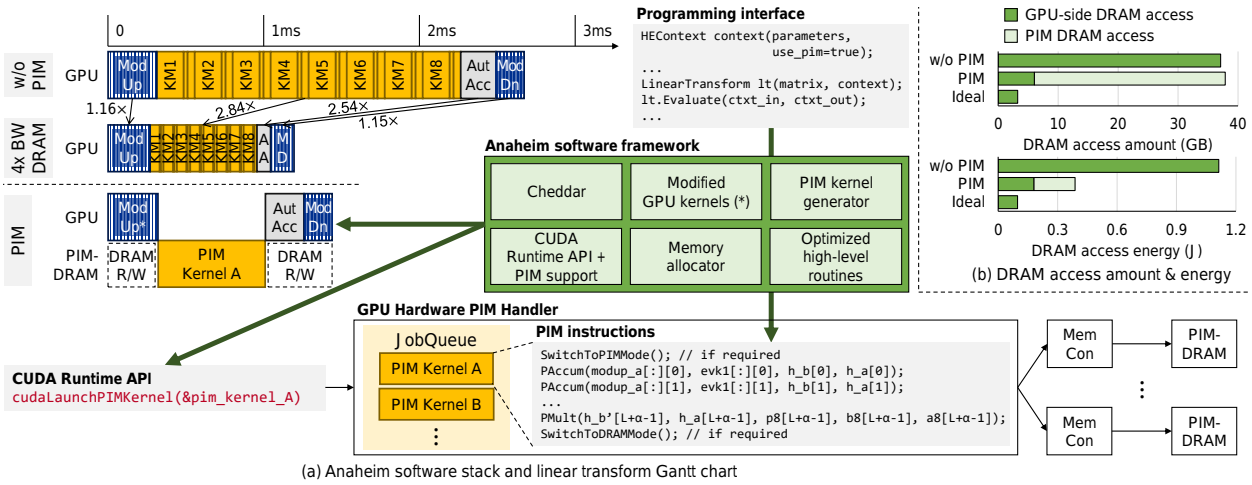



Fig. 4. (a) Anaheim software framework and GPU hardware PIM handler translating programmer-generated FHE code into PIM instructions and delivering them to PIM-enabled DRAM devices. Resulting Gantt charts for a linear transform (see Fig. 5, $K = 8$) when using only GPU (w/o PIM), when GPU DRAM bandwidth is quadrupled (4× BW DRAM), and when using PIM for element-wise ops (PIM) are also shown. For coherence, `ModUp` kernels are modified (*) when using PIM. (b) DRAM access amount and energy with and without PIM for bootstrapping derived by simulation.

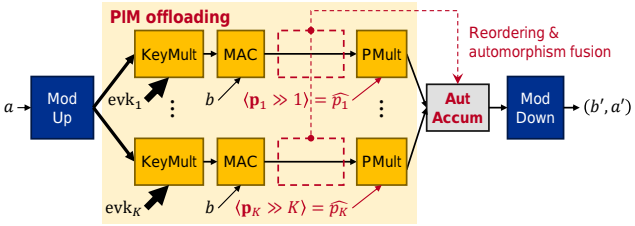



Fig. 5. Linear transform with hoisting, op sequence reordering, automorphism fusion, and PIM offloading. Differences compared to Fig. 1 are colored red.

DRAM bandwidth is available. For the estimation, we reduced the GPU clock frequency to ¼, which reduces its core throughput and cache bandwidth to ¼, but maintained the DRAM clock. Then, we divided the resulting execution times by four, which should effectively produce results for 4× higher DRAM bandwidth. Fig. 4a shows the results, where element-wise ops and automorphism get 2.84× and 2.54× faster but `ModSwitch` variants barely improve. Although effective, this naïve solution is highly unrealistic due to power constraints [62].

Instead, by utilizing PIM, we can effectively increase the memory bandwidth by exploiting the high internal bandwidth of DRAM, obtaining similar performance boosts without actually increasing the external DRAM bandwidth. *Further, we suggest using PIM exclusively for element-wise ops based on §IV.* We cannot expect much performance improvement from PIM for compute-bound (I)NTT and BConv ops inside `ModSwitch`. Also, complex data movements required for automorphism are challenging to handle for PIM. Still, the dominance of element-wise ops in FHE CKKS execution time guarantees decent speedups as shown in Fig. 4a.

Besides their low arithmetic intensity, element-wise ops are particularly amenable to PIM because they are embarrassingly parallel; an element-wise op with polynomials each containing $L$ limbs can be divided into $L \times N$ individual ops for each data point. This inherent parallelism allows us to design a highly parallel PIM device (§VI) without complex measures for data communication inside DRAM dies.

### B. Extensive reordering and automorphism fusion

PIM offloading may still require additional data transfers as relevant data needs to be written back to DRAM from the GPU cache. Nevertheless, we identify that FHE op sequences include PIM-friendly computational blocks that entirely consist of numerous element-wise ops and that require only a small amount of preparatory DRAM write-backs. Such blocks are especially noticeable in a linear transform (see Fig. 5), where a huge computational block can be offloaded to PIM after writing only $D = 4$ polynomials ($\texttt{ModUp}(a)$) back to DRAM. Then, we can replace the GPU kernels for this computational block with a large PIM kernel as in Fig. 4a.

We further reduce the amount of GPU-side DRAM access by enlarging the blocks as much as possible; we aggressively reorder op sequences such that a bulk of element-wise ops can be executed together. Fig. 5 already includes such a reordering, where we swap the order of automorphism and PMULT ops by preprocessing the plaintexts, using the property that

$$[\![(\mathbf{m} \ll R) \odot \mathbf{p}]\!] = [\![(\mathbf{m} \odot (\mathbf{p} \gg R)) \ll R]\!].$$

The original position of automorphism, which we decide to perform on the GPU, between MAC and PMULT in a linear transform incurs extra $2K$ DRAM reads and $2K$ DRAM writes, which are eliminated by the reordering.

Furthermore, we extend kernel fusion, which has previously been applied only for element-wise ops [38], to automorphism. In linear transforms, we fuse the relocated automorphism (Aut) ops with the accumulation to form a single AutAccum kernel.

We apply these techniques to various FHE CKKS op sequences. We also adequately change the position of automorphism by using different evk structures [8]. We checked that all these optimizations do not damage the precision.

### C. Anaheim software framework for seamless PIM integration

To run the PIM kernels easily and efficiently, we develop the Anaheim software framework on top of Cheddar (see Fig. 4a). We set the following objectives, which are crucial for simple and practical integration of PIM with GPUs:

- We minimize GPU hardware modifications.
- GPU chip designers are expected to utilize an abstract view of the PIM device, requiring minimal knowledge of its microarchitectural details. This abstraction layer facilitates the integration of diverse PIM devices.
- We minimize the overhead of transitions between PIM computation and regular GPU computation.
- We want the programmers to use PIM effortlessly.

**Launching PIM kernels**: We introduce an additional CUDA Runtime API for launching PIM kernels, which are handled asynchronously by an independent hardware unit. The hardware unit stores PIM kernels in a queue and dispatches the PIM instructions to the PIM devices. GPU kernels and PIM kernels are handled in an already-present stream queue, such that the end of each kernel will trigger the initiation of the next. Such design is similar to how asynchronous stream-ordered memory APIs are handled in NVIDIA GPUs and we estimate the transition overhead between GPU and PIM kernels based on them. It only takes a couple of microseconds for each transition, which is negligible considering the large granularity (hundreds of microseconds) of PIM kernels.

**Memory allocation**: As the computational flow of FHE is entirely static and all polynomials are handled in a large and fixed granularity (a limb with $N = 2^{16}$ data elements), we can predetermine the memory addresses before execution and concisely express addresses for complex memory layouts. The memory layout maximizing the efficiency of PIM is dependent on the microarchitecture, which we detail in §VI-B.

**Coherence**: We restrict PIM instructions to always write to new memory locations such that they do not affect GPU-side cache status. Also, we introduce user-controlled DRAM write-backs from the L2 cache, which is essential to make sure that data inside DRAM banks are up to date. To the best of our knowledge, NVIDIA or AMD GPUs do not provide an efficient method for this. The required functionality is similar to `clflush` in x86 architectures, but without cache invalidation for low overhead. Our software framework inserts these operations into the GPU kernels that produces data set to be processed by the PIM devices. In our experiments, we simulate the overhead of write-backs by putting additional global memory access commands, which incur the same amount of data transfers as writing back to DRAM.

**High-level programming interface**: It would be challenging for the programmers to develop an FHE application while considering various requirements of FHE and PIM. Thus, we provide a full-fledged library which supports all basic functions in FHE and includes optimized routines for advanced features, such as linear algebra, arbitrary polynomial evaluation, and DNN support. Programmers can write a simple high-level code, which will be translated into appropriate GPU kernels, API calls, and PIM kernels as shown in Fig. 4a.

**(No) pipelining**: The PIM kernels and GPU kernels never overlap but it may be possible to enhance performance with pipelining. However, pipelining requires significant GPU hardware modifications to handle cache coherence and cause frequent transitions between PIM and DRAM modes [54]. Later, we show that the portion of element-wise ops in FHE workloads can be sufficiently reduced by Anaheim such that further gains from pipelining would be marginal (see Fig. 10).

### D. GPU-side memory access reduction by PIM

PIM significantly reduces the amount of GPU-side DRAM access. For our running example in Fig. 5 ($D = 4$, $K = 8$), we can eliminate 1.20GB of DRAM access for evks and plaintexts ($\hat{p_i}$'s) alone, which are one-time-use data that never leaves DRAM. Considering the large size of evks and plaintexts in bootstrapping (see the table in Fig. 1), this benefit applies to general FHE workloads. Meanwhile, we write back only up to 68MB more for $\texttt{ModUp}(a)$ in the preceding GPU kernels. Moreover, as GPUs often do not have enough cache to hold $\texttt{ModUp}(a)$, most of it will reside in DRAM, which, without PIM, would incur multiple DRAM loads for the `KeyMult` ops.

As a quantitative analysis, we measured how much DRAM access PIM can reduce during bootstrapping through simulation. We discover op sequences similar to Fig. 5, that can substantially benefit from PIM across FHE workloads. We used efficient caching methods from MAD [2] and derived DRAM access energy using per-bit access energy values estimated based on [62]. For PIM, we assumed that computational units are placed next to each DRAM bank. We also made an estimation for an ideal case without PIM assuming unlimited cache space, such that no data write-backs occur and only compulsory cache misses occur for evks and plaintexts. The ideal case uses MinKS to reduce the total number of evks.

Fig. 4b shows that PIM significantly reduces GPU-side DRAM access to 6.03GB, 6.15× lower than the baseline without PIM. This amount is only 1.86× higher than that of the ideal case. GPU-side DRAM access for element-wise ops, which accounts for 83.7% of the total DRAM access in the baseline, is converted into PIM-side access and slightly increases in amount. However, as the physical distances data travel are reduced when using PIM, PIM overall results in a 2.87× energy reduction for accessing DRAM.

## VI. Anaheim Microarchitecture

Thus far, we have maintained an abstract view of PIM devices. We now introduce a concrete realization of a PIM microarchitecture. We exploit the high degree of parallelism in element-wise ops by distributing data elements evenly across thousands of DRAM banks and placing a PIM unit nearby each bank to process them in parallel. As in GDDR6-AiM [53], we

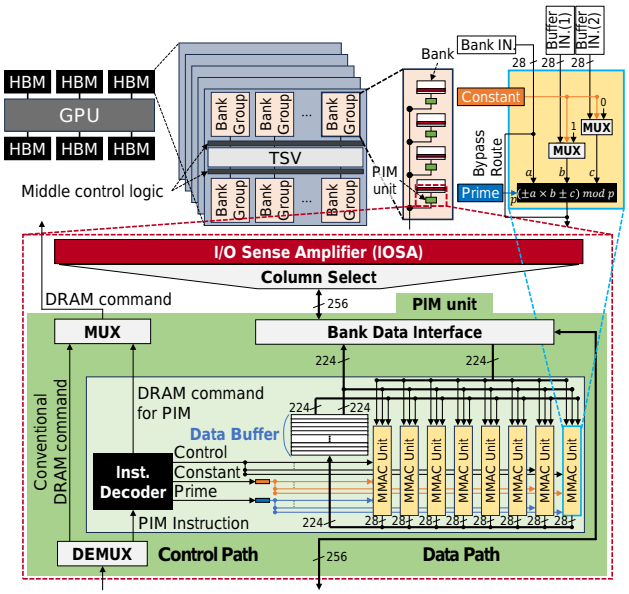



Fig. 6. Microarchitecture of an HBM stack and the Anaheim PIM unit. Anaheim can also be used for other DRAM technologies, such as GDDR.

exploit all-bank operations such that all banks in a DRAM die operate simultaneously during PIM execution.

### A. Anaheim PIM unit microarchitecture

The PIM unit, depicted in Fig. 6, consists of an instruction decoder, eight modular multiply-accumulate (MMAC) units, a data buffer, and control. The instruction decoder interprets PIM instructions, defined on top of existing DRAM command interface as in prior PIM studies [53], [54], to control other components in the PIM unit and generate DRAM commands to fetch data elements from the adjacent DRAM bank.

Given the high cost associated with implementing logic units using DRAM process nodes [24], we design the MMAC units for general-purpose modular arithmetic operations, rather than employing dedicated logic units for each specific op. We also utilize small 28-bit primes ($Q_i < 2^{28}$) to reduce the area and energy overhead based on prior work [72]. We always use double-prime scaling [1], [45] to maintain high precision despite using small primes. Data are still stored in the 32-bit granularity inside DRAM banks to facilitate GPU-side use and are truncated to 28 bits when being passed to a PIM unit. We additionally utilize the property that the eligible primes satisfy $Q_i = 1 \bmod 2N$, which is a necessary condition to enable (I)NTT, to build an efficient Montgomery modular reduction circuit [58] for the MMAC units. Each PIM unit has eight MMAC units that work in tandem, matching the 256-bit DRAM global I/O datapath size. By multiplexing the input and adjusting the positive and negative flags, we can choose between various unary, binary, and ternary modular ops. Table II summarizes the list of element-wise ops.

A 224-bit-wide $B$-entry data buffer is placed for each PIM unit. Inputs from the DRAM banks have to go through the MMAC units, even when they are not immediately used, to reduce the number of ports in the data buffer, which has two read and one write ports. Prime and constant numbers are broadcast from the instruction decoder to prevent occupying data buffer ports or redundant space.

TABLE II
ANAHEIM PIM INSTRUCTION SET ARCHITECTURE

| Instruction | Dest / Src | Functionality (modular operations) |
|---|---|---|
| Basic instructions | | |
| Move/Neg | $x$ / $a$ | $x = \pm a$ |
| Add/Sub | $x$ / $a, b$ | $x = a \pm b$ |
| Mult | $x$ / $a, b$ | $x = a * b$ |
| MAC | $x$ / $a, b, c$ | $x = a * b + c$ |
| PMult | $x, y$ / $a, b, p$ | $x = a * p, y = b * p$ |
| PMAC | $x, y$ / $a, b, p, c, d$ | $x = a * p + c, y = b * p + d$ |
| Constant instructions (constant $C$ embedded in instructions) | | |
| CAdd/CSub | $x$ / $a$ | $x = a \pm C$ |
| CMult | $x$ / $a$ | $x = C * a$ |
| CMAC | $x$ / $a, b$ | $x = C * a + b$ |
| Compound instructions ($i \in \{1, 2, \cdots, K\}$) | | |
| Tensor | $x, y, z$ / $a, b, c, d$ | $x = a * c, y = a * d + b * c, z = b * d$ |
| TensorSq | $x, y, z$ / $a, b$ | $x = a * a, y = 2 * a * b, z = b * b$ |
| ModDownEp | $x$ / $a, b$ | $x = C * (a - b)$ |
| PAccum$\langle K \rangle$ | $x, y$ / $\{a_i, b_i, p_i\}$ ($i \in [0, K-1]$) | $x = \sum_{i=0}^{K-1} a_i * p_i$, $y = \sum_{i=0}^{K-1} b_i * p_i$ |
| CAccum$\langle K \rangle$ | $x, y$ / $\{a_i, b_i\}$ ($i \in [1, K]$) | $x = C_0 + \sum_{i=1}^{K} C_i * a_i$, $y = C_0 + \sum_{i=1}^{K} C_i * b_i$ |

### B. Column partitioning data layout and PolyGroup

We design an efficient and versatile data layout friendly for PIM execution. We first partition the DRAM dies in a GPU into a few die groups. For A100 80GB with five 8-Hi HBM stacks, we determine each stack as a separate die group, which will handle $L/5$ limbs of a polynomial. This guarantees that all the banks in the same DRAM die operate with the same prime or constant when handling a PIM instruction, enabling us to embed prime and constant values inside PIM instructions. By utilizing both limb-level and coefficient-level parallelism, we maximally exploit the data parallelism and achieve high scalability for systems with many DRAM dies.

How we place the data elements inside each DRAM bank has profound impacts on the performance and efficiency of PIM instructions. Data in a row of a DRAM bank must be copied into the IOSAs before use. For regular DRAM reads and writes, each bank can hide the ACT/PRE overhead; when one bank is performing ACT or PRE, other banks will occupy the global I/O datapath to keep the shared DRAM channel busy. However, as all the banks work simultaneously for PIM, ACT/PRE times are directly exposed.

To mitigate this overhead, we devise a column partitioning data layout to place relevant data in the same row such that many chunks of data can be fetched from the IOSAs, amortizing the ACT/PRE overhead. A DRAM bank in an HBM can be abstracted as having many 8Kb-wide rows, which is accessed in the granularity of a 256-bit chunk; a row

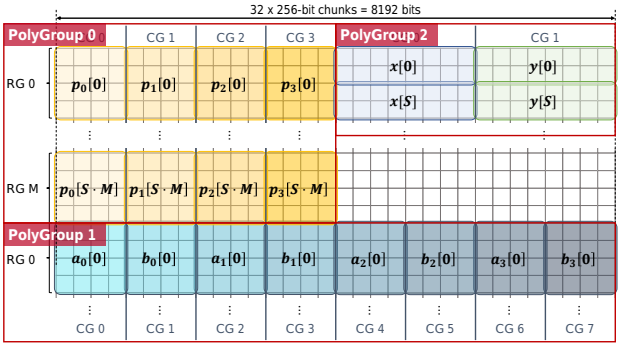



Fig. 7. Exemplar data layout inside a DRAM bank when 128 data elements (16 chunks) are allocated to a bank per limb, $S$ HBM stacks are used, and $S \cdot M < L \leq S \cdot (M+1)$. This data layout is used for Alg. 1. Although we depict a PolyGroup as being composed of adjacent row groups (RGs) and column groups (CGs) for visual clarity, nonadjacent RGs/CGs can be used.

contains 32 chunks. We take an example where 16 chunks (128 elements) are allocated per bank for a limb. In the runtime, we partition each row into 4/8/16 column groups (CGs) each holding 8/4/2 chunks. Each limb is placed in a column group in a wrapped manner, forming a row group (RG) composed of adjacent 2/4/8 rows. Fig. 7 depicts an exemplar data layout.

Programmers can request memory space, referred to as PolyGroup, comprising multiple row and column groups. We place relevant polynomials in a PolyGroup, where the limbs of a polynomial (e.g., $x[0], x[S], \dots$) are allocated to different row groups and different polynomials (e.g., $x[i]$ and $y[i]$ in PolyGroup 2) are allocated to different column groups. This preallocation strategy is feasible due to the static nature of FHE workloads, which does not show control divergence.

If we access the polynomials in adequate chunk granularities (e.g., 8) for performing element-wise PIM ops between polynomials in the same PolyGroup (e.g., $x + y$), it is guaranteed that accessing data elements from different polynomials will not incur extra DRAM bank ACT or PRE.

### C. Efficient PIM kernel fusion with PolyGroups

We extend kernel fusion to PIM kernels and develop efficient methods to handle fused compound instructions (see Table II). As an example, we describe how Anaheim handles `KeyMult` with the PAccum$\langle$D$\rangle$ PIM instruction in Alg. 1 for $D = 4$. We first determine the chunk granularity $G$ based on the buffer size $B$. For PAccum$\langle 4\rangle$, we set $G = \lfloor B/6 \rfloor$ to let the data buffer temporarily store $G$ chunks each of $p_0, p_1, p_2, p_3, x, y$. We repeat the process of (1) loading $G$ chunks of $p_k$ to the buffer, (2) loading $G$ chunks of $a_k$ and $G$ chunks of $b_k$ for immediate MMAC computation with $p_k$ and temporary accumulation results loaded from the buffer, and (3) storing $G$ chunks of the results $(x, y)$ back to the banks.

Our PIM unit design, combined with the effective use of PolyGroups, minimizes ACT/PRE overhead during execution. The ACT/PRE cost is amortized across $4G$, $8G$, and $2G$ chunk reads/writes for (1), (2), and (3), respectively. In contrast, a naïve contiguous allocation placing the polynomials all in separate DRAM rows would require $4\times$, $8\times$, and $2\times$ more ACT/PRE for (1), (2), and (3). In fact, it becomes better to skip PIM kernel fusion and just repeat PMAC instructions in the naïve case. We implement efficient execution methods for all PIM instructions in a similar way, whose efficiency increases with the data buffer size ($B$).

**Algorithm 1** Optimized PIM execution of PAccum$\langle 4\rangle$

**Require:** $\{p_0, \cdots, p_3\}$ in PolyGroup0, $\{(a_0, b_0), \cdots, (a_3, b_3)\}$ in PolyGroup1, $C$ chunks allocated per DRAM bank for each limb, and data buffer `buf` with $B$ entries
**Ensure:** $\{x = \sum_{k=0}^{3} a_k * p_k, y = \sum_{k=0}^{3} b_k * p_k\}$ in PolyGroup2
Chunk granularity $G = \lfloor B/6 \rfloor$ ▷ Assume $G$ divides 4
For each $\ell$-th limb, compute the following at each PIM unit
**for** $i \leftarrow 0$ **to** $C/G - 1$ **do**
  (1) PRE and ACT the corresponding row in PolyGroup0
  **for** $k \leftarrow 0$ **to** 3 **do**
    **for** $j \leftarrow 0$ **to** $G - 1$ **do** ▷ Bypass MMAC
      `buf`$[kG + j] \leftarrow p_k[\ell]$.chunks$[iG + j]$
  (2) PRE and ACT the corresponding row in PolyGroup1
  **for** $k \leftarrow 0$ **to** 3 **do**
    **for** $j \leftarrow 0$ **to** $G - 1$ **do** ▷ Immediately compute MMAC
      `buf`$[4G + j] \mathrel{+}= a_k[\ell]$.chunks$[iG + j] *$ `buf`$[kG + j]$
      `buf`$[5G + j] \mathrel{+}= b_k[\ell]$.chunks$[iG + j] *$ `buf`$[kG + j]$
  (3) PRE and ACT the corresponding row in PolyGroup2
  **for** $j \leftarrow 0$ **to** $G - 1$ **do** ▷ Write back
    $x[\ell]$.chunks$[iG + j] \leftarrow$ `buf`$[4G + j]$
    $y[\ell]$.chunks$[iG + j] \leftarrow$ `buf`$[5G + j]$

### D. Anaheim supporting various DRAM/PIM architectures

Anaheim is not confined to specific DRAM or PIM architectures. While we have used HBM as an example, Anaheim can be applied to DDR, GDDR, and LPDDR memories. We evaluate both HBM-based (A100 80GB) and GDDR-based (RTX 4090) Anaheim in §VII. Also, while we have introduced a custom near-bank PIM unit microarchitecture for FHE in §VI-A, we can also utilize other PIM device types, such as general-purpose ones [24], to which the other contributions of ours (e.g., software stack and data layout) still applies. In this way, Anaheim enables hardware designers to adopt PIM for FHE according to their practical needs.

Finally, we propose an alternative Anaheim design based on custom-HBM [14] and advanced packaging technologies, such as hybrid bonding [75]. Although practical integration of GPU+PIM has already been demonstrated by Samsung's MI100-PIM [43], modifying the microarchitecture of DRAM dies may still be demanding, especially for those other than the DRAM manufacturers. To tackle this problem, we propose placing all the PIM units inside a custom logic die of HBMs. Each PIM unit is connected to multiple banks in a DRAM die by through-silicon vias (TSVs). We still refrain from increasing the external DRAM bandwidth. However, we enable higher internal bandwidth between the logic die and the DRAM dies by placing more TSVs, following the approach in [80]. Advanced packaging technologies reduce the area overhead of TSVs to allow this modification. This alternative design only needs significant modifications to the logic die, for which we can utilize the logic process node (from HBM4)

TABLE III
TESTED GPUS AND ANAHEIM CONFIGURATIONS.

| GPU | A100 80GB | | RTX 4090 |
|---|---|---|---|
| PIM | Near-bank PIM | Custom-HBM | Near-bank PIM |
| Compute† | 19.5 TOPS | | 41.3 TOPS |
| DRAM | 5 HBM2E stacks | | 12 GDDR6X dies |
| ⌞ bandwidth | 1802GB/s | | 939GB/s |
| ⌞ capacity | 80GB | | 24GB |
| DRAM banks | 64 per die | | 32 per die |
| PIM die group | 1 stack (8 dies) | | 4 dies |
| PIM MMAC | 0.194 TOPS per die (378MHz) | 0.388 TOPS per stack (756MHz) | 0.168 TOPS per die (656MHz) |
| Buffer size ($B$) | 16 | 16 | 32 |
| BW incr.‡ | 16× | 4× | 8× |
| Area overhead (portion) | 10.7mm$^2$ per die (9.69%) | 10.9mm$^2$ per die (9.94%) | 7.26mm$^2$ per die (7.58%) |

† Theoretical peak 32-bit integer multiply-and-add throughput.
‡ Theoretical peak effective DRAM bandwidth increase attainable from utilizing the DRAM internal bandwidth with PIM.

TABLE IV
DEFAULT PARAMETERS USED IN THE PAPER FOR EVALUATION. $\Delta$ IS THE SCALING FACTOR [18], $H_d$ AND $H_s$ ARE DENSE AND SPARSE SECRET HAMMING WEIGHTS [9], AND $\lambda$ IS THE IND-CPA SECURITY LEVEL [7].

| Param | $N$ | Primes | $L$ | $\alpha$ | $D$ | $\Delta$ | $H_d$ | $H_s$ | $\lambda$ |
|---|---|---|---|---|---|---|---|---|---|
| Value | $2^{16}$ | $< 2^{28}$ | $\leq 54$ | $\leq 14$ | 4 | $2^{48}$–$2^{55}$ | $2^8$–$2^{15}$ | $2^5$ | $\geq 128$ |

for easier PIM unit design and enhanced performance and efficiency.

## VII. EVALUATION

### A. Experimental setup

Our PIM simulator is implemented on top of the state-of-the-art DRAM simulator, Ramulator 2.0 [57]. We used prior work [62] to estimate DRAM energy, while approximating the data movement energy within a DRAM die based on the datapath length inside DRAM dies [65], [70], [73]. Area and energy of a PIM unit were estimated by first synthesizing it in RTL using a 7nm predicitive process design kit, ASAP7 [22], applying voltage and 7nm-to-14nm process node scalings based on [59], and applying conservative 10× area compensations for the DRAM process node overhead [24].

We used real GPUs, summaried in Table III, to estimate time and energy. We measured energy using the NVIDIA Management Library (NVML) [61]. Using our software framework (see §V-C), we replaced the GPU kernels with PIM kernels as well as modified GPU kernels. Our simulation framework separately runs GPU codes and PIM simulation codes, later combining the results to get the total time and energy. We configured our PIM units to incur area overhead within 10% of the DRAM dies [21], [48] for the default settings (see Table III). For the custom-HBM design, the PIM units, for which we utilized ASAP7, and the additional TSVs incur 9.94% of area overhead for the logic die. We assumed a TSV pitch of 22$\mu$m [76].

The parameters we used, which satisfy the standard 128-bit security constraints [7], are summarized in Table IV. We always used double-prime scaling [1], [45] to use high scaling factors ($\Delta$) to maintain the precision of data. We used six FHE workloads, most of which are for private DNN, which is the main target application of FHE CKKS:

- **Boot**: Full-slot ($2^{15}$ complex numbers) bootstrapping with sparse-secret encapsulation [9]. $L$ changes as $2 \rightarrow 54 \rightarrow 24$ during bootstrapping. $L_{\text{eff}} = 11$.
- **HELR** [33]: Logistic regression for binary classification. We trained the model for 32 iterations, each with a 1024-batch of $14 \times 14$ MNIST images. We report per-iteration training time and energy. $L_{\text{eff}} = 10$.
- **Sort** [35]: Two-way sorting of $2^{14}$ real numbers. $L_{\text{eff}} = 9$.
- **RNN** [67]: Recurrent neural network inference. We packed a 32-batch of 128-long embeddings in a ciphertext and performed 200 iterations of evaluating an RNN cell. $L_{\text{eff}} = 10$.
- **ResNet20** [49]: ResNet20 convolutional neural network (CNN) inference with a $32 \times 32 \times 3$ CIFAR-10 image. $L_{\text{eff}} = 8$.
- **ResNet18-AESPA** [37]: ResNet18 CNN inference with a $224 \times 224 \times 3$ ImageNet image based on NeuJeans [37]. AESPA [64] was used for activation. $L_{\text{eff}} = 7$.

We used Cheddar [44] as the baseline GPU implementation.

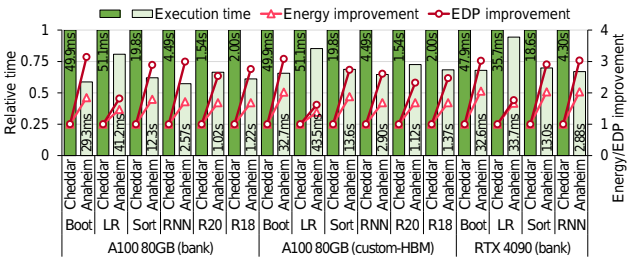
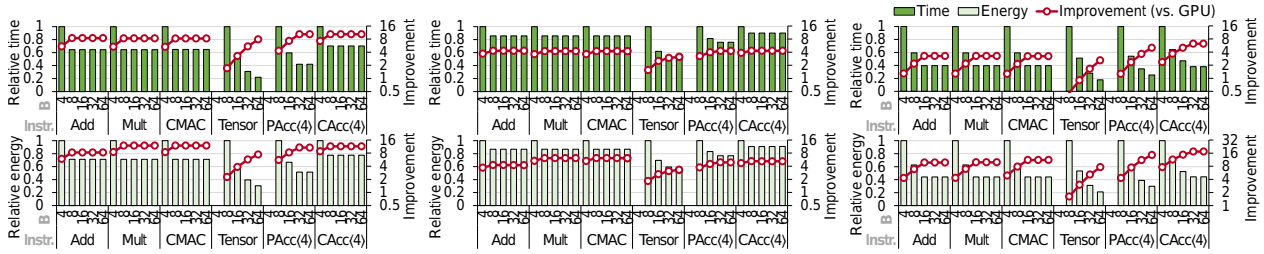



Fig. 8. Workload execution time, energy efficiency, and energy-delay product (EDP) improvements from using Anaheim. ResNet20 (R20) and ResNet18-AESPA (R18) faced out-of-memory errors on RTX 4090.

### B. Workload performance & energy efficiency improvements

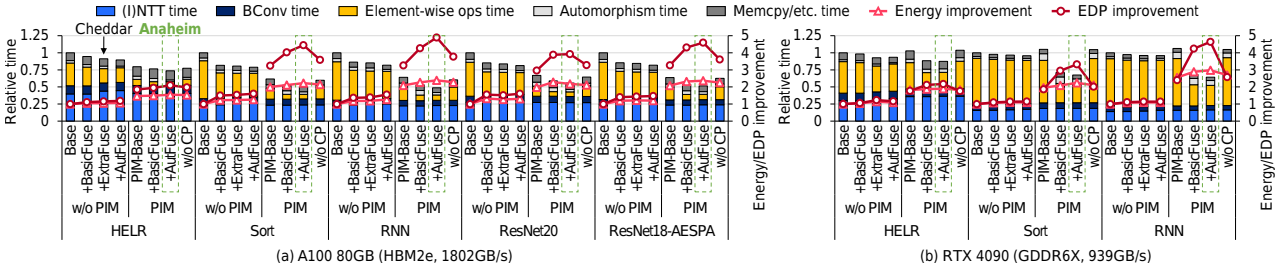
Anaheim significantly enhances the workload performance and energy efficiency on GPUs as shown in Fig. 8. Anaheim results in speedups of 1.24–1.74× on A100 80GB with near-bank PIM, 1.17–1.55× on A100 80GB with custom-HBM, and 1.06–1.49× on RTX 4090 with near-bank PIM. Although element-wise ops account for higher portions of execution times on RTX 4090 (§IV), the effective bandwidth increase achievable from PIM is lower (8× vs. 16×, see Table III) for its DRAM configuration, which leads to relatively lower speedups. Although we set even lower 4× bandwidth increase for the custom-HBM solution, A100 80GB with custom-HBM shows just slightly lower speedups because we can better hide the overhead for accessing DRAM banks as each PIM unit is in charge of multiple banks for custom-HBM.

Anaheim also reduces the energy consumption by 1.38–2.05×, overall resulting in energy-delay product (EDP) improvements (reductions) of 1.62–3.14×. While relatively modest improvements of 1.62–1.82× are achieved for HELR, the




(a) A100 80GB (HBM2E) with near-bank PIM (b) A100 80GB (HBM2E) with custom-HBM (c) RTX 4090 (GDDR6X) with near-bank PIM

Fig. 9. Microbenchmark of PIM instructions depending on the number of data buffer entries ($B$) when using Anaheim on A100 80GB or on RTX 4090.




(a) A100 80GB (HBM2e, 1802GB/s) (b) RTX 4090 (GDDR6X, 939GB/s)

Fig. 10. Performance of the baseline GPU (w/o PIM) and Anaheim using near-bank PIM when we incrementally introduced basic kernel fusions (+BasicFuse) and automorphism fusion (+AutFuse). For the baseine, we also applied extra kernel fusions (+ExtraFuse) that are instead handled by PIM in Anaheim. For Anaheim (PIM), we also show the results for the case where all the algorithms are applied but without the column partitioning data layout (w/o CP).

other workloads are improved by 2.47–3.14×. This is due to the small scale of the HELR workload; instead of performing bootstrapping for the maximum $N/2 = 2^{15}$ number of data elements, HELR only requires bootstrapping for its $14 \times 14 = 196$ weights. In this case, bootstrapping becomes much cheaper and the significance of linear transforms declines. Thus, the dominance of `ModSwitch` in HELR (see Fig. 10) results in less significant improvements when using Anaheim. However, more practical workloads, such as ResNet18-AESPA, involve much more data elements and are free from this limitation.

### C. PIM kernel microbenchmark

We also conducted a microbenchmark of PIM instructions while modifying the number of data buffer entries ($B$) from four to 64, whose results are shown in Fig. 9. Some compound PIM instructions (e.g., Tensor and PAccum$\langle 4 \rangle$) are not supported when using a small $B$. Higher $B$ enables using a larger chunk granularity ($G$) for the PIM instructions to reduce the ACT/PRE overhead. However, as we increase $B$, performance and energy efficiency improvements eventually saturate. The saturation is faster for custom-HBM, which suffers less from the ACT/PRE overhead. For our default configurations with $B$ = 16–32 (see Table III), Anaheim shows 1.65–10.33× speedups and 2.63–17.39× energy efficiency improvements.

Especially high speedups are achieved for PAccum (7.26×/3.98×/3.63×) and CAccum (10.33×/4.31×/6.20×). Our optimized PIM execution methods for compound PIM instructions using PolyGroups allow further amortization of the ACT/PRE overhead, resulting in higher speedups for them.

### D. Sensitivity study

We performed a sensitivity study to assess our algorithmic contributions. We focused on Anaheim using near-bank PIM. The baselines (Base and PIM-Base) already include GPU kernel fusion methods embedding constant-polynomial element-wise ops into (I)NTT kernels [38]. First, we additionally introduced PAcuum and CAccum (+BasicFuse). For the GPU implementation without PIM, we also applied additional GPU kernel fusions (+ExtraFuse), such as `ModDown` fusion in [38], which are better handled by PIM offloading on Anaheim. Without PIM, these fusions result in modest 27–37% (resp., 5–18%) reductions in the times spent on element-wise ops on A100 80GB (RTX 4090); with Anaheim, they incur greater reductions of 40–57% (38–55%). This signifies the importance of reducing the ACT/PRE overhead by PIM kernel fusion. Automorphism fusion (+AutFuse) results in additional 1.01–1.09× and 1.01–1.15× improvements (reductions) in workload execution times and EDP values.

The column partitioning data layout (§VI-B) is crucial for the effectiveness of the algorithms. If we contiguously allocate data elements in a polynomial inside DRAM banks (w/o CP) instead, the times spent on element-wise ops get respectively 2.24× and 2.11× slower on A100 80GB and RTX 4090 in terms of geometric mean, nullifying the aforementioned performance benefits.

Meanwhile, the GPU-side memory access instructions we put into GPU kernels considering extra DRAM write-backs before PIM execution (§V-C) result in the total (I)NTT time and the total automorphism time to increase by 1.03–1.11×

and 2.39–2.89× on RTX 4090. Still, the time reductions in element-wise ops outweigh the increases. A100 80GB is not affected much by the extra instructions because of its higher DRAM bandwidth and lower computational throughput.

## VIII. Discussion

### A. Practical FHE Execution — vs. GPU, FPGA, and ASIC

Table V shows workload-level performance comparison of Anaheim with prior acceleration studies. Although fair comparison is difficult due to the large variance in parameter choices and workload implementations among these studies, it is clear that Anaheim outperforms prior GPU and FPGA solutions by a large margin. We mainly used Cheddar for fair comparison in §VII. While Anaheim also shows comparable performance to GME [74] (modified GPU) and BTS [47] (ASIC), several other ASIC proposals achieve much higher performance compared to them. For example, SHARP [45] shows 8.9–17.2× faster execution times than Anaheim.

The large performance gaps are expected considering the hardware differences discussed in §III-A, especially **D1**. When using MinKS with hundreds of MBs of on-chip cache, DRAM bandwidth is no longer the source of bottleneck and the performance of an ASIC design is determined by the amount of computational resources inside the chip [45]. However, considering that FHE is still evolving, the tremendous design costs required for realizing these large (hundreds of $mm^2$) and complex ASIC designs are yet to be justified.

Anaheim takes a different approach. We mostly reuse existing designs with small hardware modifications to the DRAM die (around $10mm^2$ in area) and the GPU chip, which is enabled by our software support (§V). We find our custom-HBM variant especially practical as it requires significant modifications only to the logic die of HBM. Despite minimal hardware modifications, Anaheim demonstrates the potential for practical FHE in executing complex real-world applications, as evidenced by achieving ImageNet CNN inference (ResNet18-AESPA) in just 1.22–1.37 seconds (see Fig. 8).

### B. DRAM capacity

While prior architectural studies did not elaborate on DRAM capacity, we expect it to become one of the major limiting factors as more complex FHE workloads are developed. Our evaluation on RTX 4090 with 24GB DRAM faced out-of-memory (OoM) failures even for ResNet20. Also, ResNet18-AESPA requires over 40GB of memory. Prior ASIC designs [45]–[47], [72], each of which deploy two HBM stacks, and FPGA studies [3], [78], which utilize an FPGA board with 8GB HBM2, are not free from these problems.

### C. Future work

The proposed Anaheim architecture is highly general; thus, we expect it to be applicable to other related domains (especially in security) that require high-throughput parallel modular arithmetic. A direct extension for other FHE schemes would be feasible. For example, BGV [10] and BFV [11], [27] include the same `KeyMult` ops, and FHEW [25] and TFHE [20] also require similar parallel mult process for their evks. However, thorough analyses for these schemes as in §III and §IV must precede, which we leave as future work.

TABLE V
Reported Boot, HELR, ResNet20 (R20), and Sort execution time on prior GPU implementations, custom accelerators (GME [74], FPGA, and ASIC), and Anaheim (GPU+PIM)

| Proposal | Cache | DRAM | Boot | HELR | R20 | Sort |
|---|---|---|---|---|---|---|
| 100× [38] (V100) | 6MB | 835GB/s | 328ms | 775ms | - | - |
| TensorFHE [28] (A100) | 40MB | 1.4TB/s | 250ms | 1007ms | 4.94s | - |
| [63] (A100) | 40MB | 1.8TB/s | 171ms | - | 8.57s | - |
| GME [74] (MI100†) | 15.5MB | 1.1TB/s | 33.6ms | 54.5ms | 0.98s | - |
| FAB [3] (FPGA) | 43MB | 460GB/s | 477ms | 103ms | - | - |
| Poseidon [78] (FPGA) | 8.6MB | 460GB/s | 128ms | 72.9ms | 2.66s | - |
| CraterLake [72] (ASIC) | 256MB | 1TB/s | 6.33ms | 3.81ms | 0.32s | - |
| BTS [47] (ASIC) | 512MB | 1TB/s | 28.6ms | 28.4ms | 1.91s | 15.6s |
| ARK [46] (ASIC) | 512MB | 1TB/s | 3.52ms | 7.42ms | 0.13s | 1.99s |
| SHARP [45] (ASIC) | 180MB | 1TB/s | 3.12ms | 2.53ms | 0.10s | 1.38s |
| Anaheim (A100) | 40MB | 1.8TB/s | 29.3ms | 41.2ms | 1.02s | 12.3s |
| ⌞ custom-HBM | 40MB | 1.8TB/s | 32.7ms | 43.5ms | 1.12s | 13.6s |
| Anaheim (RTX 4090) | 72MB | 939GB/s | 32.6ms | 33.7ms | OoM‡ | 13.0s |

† GME extensively modifies the AMD MI100 GPU architecture for FHE.
‡ Failure due to the out-of-memory (OoM) error.

## IX. Related Work

**GPU**: 100× [38], an early GPU acceleration work for FHE CKKS, tried to alleviate the overhead of element-wise ops by applying kernel fusion. Phantom [77] is a more recent open-source GPU work supporting various FHE schemes (BGV, BFV, and CKKS) but it does not include CKKS bootstrapping. Cheddar [44] improved upon 100× by fine-tuning GPU kernels and applying state-of-the-art algorithms. TensorFHE [28] attempted to use tensor cores in NVIDIA GPUs to accelerate (I)NTT. However, the matrix-mult-based (I)NTT method in TensorFHE requires $2N\sqrt{N} + N$ mults, which is much higher than $\frac{1}{2}N \log N$ for FFT-based methods. Meanwhile, GME [74] proposed adding new networks and computational units to an AMD MI100 GPU to better support the compute-intensive ops in CKKS. However, substantial hardware modifications are required to implement GME. Some studies focused on accelerating specific tasks. [63] accelerated FHE CNNs by employing an efficient packing method [41] and the AESPA [64] activation method. BoostCom [79] accelerated comparison operations for the BFV scheme. Liberate.FHE [23] and REDcuFHE [29] leveraged multiple GPUs to accelerate CKKS and TFHE, respectively.

**FPGA**: HEAX [68] and Roy et al. [69] presented FPGA implementations supporting major functions in FHE, but not supporting CKKS bootstrapping. Later, more capable FPGA designs supporting bootstrapping, FAB [3] and Poseidon [78], were proposed. However, their performance gains are modest as they reported slower execution times than a GPU [44].

**ASIC**: F1 [71] proposed a high-throughput vector architecture for CKKS; however, as F1 targeted small parameters ($N = 2^{14}$), its performance severely degrades for large FHE

parameters. CraterLake [72] improved upon F1 by adapting to the changes in larger parameters ($N = 2^{16}$) and adding new computational units, such as a unit for BConv. BTS [47] discovered compelling parameter choices for CKKS accelerators ($N = 2^{17}$) under the assumption that the performance of an accelerator cannot go beyond the point where the latency of a basic function (e.g., HROT) can be fully hidden within the DRAM read time of an evk. However, ARK [46] showed that accelerators can achieve higher performance by applying MinKS to reuse evks with a massive 512MB cache. SHARP [45] later improved upon ARK by finding an efficient word length choice and reducing the cache size to 180MB.

**PIM**: Several preliminary attempts have been made to apply PIM to homomorphic encryption, but most stayed at demonstrating the potential benefits for simple ops. [30], [36] utilized UPMEM PIM [24] to evaluate polynomial mult, but their performance enhancements stay at modest levels even compared to CPUs. [36] discovered that performing (I)NTT with PIM requires data communication between computational units at different DRAM banks, which incurs severe data communication bottleneck due to the low global NoC bandwidth in DRAM dies. Meanwhile, other PIM studies targeted (I)NTT [60], [66] or other FHE schemes [31].

## X. Conclusion

While GPUs offer a practical solution for accelerating FHE, they suffer from severe off-chip DRAM bandwidth bottleneck caused by element-wise ops. To address this bottleneck, we have proposed Anaheim, which combines an in-DRAM PIM architecture and algorithms for efficient PIM offloading. We developed a software framework with optimized execution flows enhancing the applicability of PIM for FHE, versatile PIM functional units, fused PIM instructions, and data layout focusing on eliminating redundant DRAM commands. By these means, Anaheim enhances the performance and energy efficiency of FHE CKKS workloads on GPUs, achieving 1.62–3.14× energy-delay product reductions.

## Acknowledgement

This work was supported in part by Samsung Electronics Co., Ltd (IO201207-07812-01) and by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) [RS-2024-00402898, RS-2021-II211343, RS-2023-00256081]. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. The Institute of Computer Technology (ICT) at Seoul National University (SNU) provides research facilities for this study. Jongmin Kim, Sungmin Yun, and Hyesung Ji are with the Interdisciplinary Program in Artificial Intelligence (IPAI), SNU. Wonseok Choi is with the Department of Intelligence and Information and with the SNU Graduate School of AI Semiconductor (SGAIS), SNU. Sangpyo Kim is with the Department of Intelligence and Information, SNU. Jung Ho Ahn, the corresponding author, is with the Department of Intelligence and Information, Inter-university Semiconductor Research Center, IPAI, and SGAIS, SNU.

## References

[1] R. Agrawal, J. Ahn, F. Bergamaschi, R. Cammarota, J. H. Cheon, F. D. M. de Souza, H. Gong, M. Kang, D. Kim, J. Kim, H. de Lassus, J. H. Park, M. Steiner, and W. Wang, "High-Precision RNS-CKKS on Fixed but Smaller Word-Size Architectures: Theory and Application," in *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2023, https://doi.org/10.1145/3605759.3625257.

[2] 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 *MICRO*, 2023, https://doi.org/10.1145/3613424.3614302.

[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 *HPCA*, 2023, https://doi.org/10.1109/HPCA56546.2023.10070953.

[4] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data," in *Privacy Enhancing Technologies Symposium*, 2023, https://doi.org/10.56553/popets-2023-0020.

[5] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-Source Fully Homomorphic Encryption Library," in *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2022, https://doi.org/10.1145/3560827.3563379.

[6] A. A. Badawi and Y. Polyakov, "Demystifying Bootstrapping in Fully Homomorphic Encryption," *IACR Cryptology ePrint Archive*, no. 149, 2023, https://eprint.iacr.org/2023/149.

[7] J.-P. Bossuat, R. Cammarota, J. H. Cheon, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara, C. Lee, X. Lu, C. Maple, A. Pedrouzo-Ulloa, R. Player, L. A. R. Lopez, Y. Song, D. Yhee, and B. Yildiz, "Security Guidelines for Implementing Homomorphic Encryption," *IACR Cryptology ePrint Archive*, no. 463, 2024, https://eprint.iacr.org/2024/463.

[8] J. Bossuat, C. Mouchet, J. R. Troncoso-Pastoriza, and J. Hubaux, "Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2021, https://doi.org/10.1007/978-3-030-77870-5_21.

[9] J. Bossuat, J. Troncoso-Pastoriza, and J. Hubaux, "Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation," in *Applied Cryptography and Network Security*, 2022, https://doi.org/10.1007/978-3-031-09234-3_26.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," *ACM Transactions on Computing Theory*, vol. 6, no. 3, pp. 1–36, 2014, https://doi.org/10.1145/2633600.

[11] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014, https://doi.org/10.1137/120868669.

[12] C.-H. Chang, V. Chang, K. Pan, K. Lai, J. H. Lu, J. Ng, C. Chen, B. Wu, C. Lin, C. Liang, C. Tsao, Y. Mor, C. Li, T. Lin, C. Hsieh, P. Chen, H. Hsu, J. Chen, H. Chen, J. Yeh, M. Chiang, C. Lin, J. Liaw, C. Wang, S. Lee, C. Chen, H. Lin, R. Chen, K. Chen, C. Chui, Y. Yeo, K. Huang, T. Lee, M. Tsai, K. Chen, Y. Lu, S. Jang, and S.-Y. Wu, "Critical Process Features Enabling Aggressive Contacted Gate Pitch Scaling for 3nm CMOS Technology and Beyond," in *International Electron Devices Meeting*, 2022, https://doi.org/10.1109/IEDM45625.2022.10019565.

[13] J. Chang, Y. Chen, G. Chan, H. Cheng, P. Wang, Y. Lin, H. Fujiwara, R. Lee, H. Liao, P. Wang, G. Yeap, and Q. Li, "A 5nm 135Mb SRAM in EUV and High-Mobility-Channel FinFET Technology with Metal Coupling and Charge-Sharing Write-Assist Circuitry Schemes for High-Density and Low-VMIN Applications," in *IEEE International Solid-State Circuits Conference*, 2020, https://doi.org/10.1109/ISSCC19947.2020.9062967.

[14] K. Chatterjee, Y. Li, H. Chang, M. Damadam, P. Asrar, J. Kim, G. Jeong, and W. Kim, "Thermal and Mechanical Simulations of 3D Packages with Custom High Bandwidth Memory (HBM)," in *IEEE Electronic Components and Technology Conference*, 2024, pp. 1054–1059, https://doi.org/10.1109/ECTC51529.2024.00169.

[15] H. Chen, I. Chillotti, and Y. Song, "Improved Bootstrapping for Approximate Homomorphic Encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2019, https://doi.org/10.1007/978-3-030-17656-3_2.

[16] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A Full RNS Variant of Approximate Homomorphic Encryption," in *Selected Areas in Cryptography*, 2018, https://doi.org/10.1007/978-3-030-10970-7_16.

[17] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for Approximate Homomorphic Encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2018, https://doi.org/10.1007/978-3-319-78381-9_14.

[18] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *International Conference on the Theory and Applications of Cryptology and Information Security*, 2017, https://doi.org/10.1007/978-3-319-70694-8_15.

[19] J. H. Cheon, Y. Son, and D. Yhee, "Practical FHE Parameters against Lattice Attacks," *Journal of the Korean Mathematical Society*, vol. 59, no. 1, pp. 35–51, 2022, https://doi.org/10.4134/JKMS.j200650.

[20] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption Over the Torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020, https://doi.org/10.1007/s00145-019-09319-x.

[21] K. C. Chun, Y. K. Kim, Y. Ryu, J. Park, C. S. Oh, Y. Y. Byun, S. Y. Kim, D. H. Shin, J. G. Lee, B.-K. Ho, M.-S. Park, S.-J. Cho, S. Woo, B. M. Moon, B. Kil, S. Ahn, J. H. Lee, S. Y. Kim, S.-K. Choi, J.-S. Jeong, S.-G. Ahn, J. Kim, J. J. Kong, K. Sohn, N. S. Kim, and J.-B. Lee, "A 16-GB 640-GB/s HBM2E DRAM With a Data-Bus Window Extension Technique and a Synergetic On-Die ECC Scheme," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 199–211, 2021, https://doi.org/10.1109/JSSC.2020.3027360.

[22] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016, https://doi.org/10.1016/j.mejo.2016.04.006.

[23] DESILO, "Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy," 2023, https://github.com/Desilo/liberate-fhe.

[24] F. Devaux, "The True Processing In Memory Accelerator," in *IEEE Hot Chips 31 Symposium*, 2019, https://doi.org/10.1109/HOTCHIPS.2019.8875680.

[25] L. Ducas and D. Micciancio, "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015, https://doi.org/10.1007/978-3-662-46800-5_24.

[26] EPFL-LDS and Tune Insight SA, "Lattigo v4," Aug 2022, https://github.com/tuneinsight/lattigo.

[27] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *IACR Cryptology ePrint Archive*, no. 144, 2012, https://eprint.iacr.org/2012/144.

[28] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU," in *HPCA*, 2023, https://doi.org/10.1109/HPCA56546.2023.10071017.

[29] L. Folkerts, C. Gouert, and N. G. Tsoutsos, "REDsec: Running Encrypted Discretized Neural Networks in Seconds," in *Network and Distributed System Security Symposium*, 2023, pp. 1–17, https://doi.org/10.14722/ndss.2023.24034.

[30] H. Gupta, M. Kabra, J. Gómez-Luna, K. Kanellopoulos, and O. Mutlu, "Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System," in *IEEE International Symposium on Workload Characterization*, 2023, https://doi.org/10.1109/IISWC59245.2023.00030.

[31] S. Gupta, R. Cammarota, and T. Šimunić, "MemFHE: End-to-end Computing with Fully Homomorphic Encryption in Memory," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 2, 2024, https://doi.org/10.1145/3569955.

[32] S. Halevi and V. Shoup, "Faster Homomorphic Linear Transformations in HElib," in *Annual International Cryptology Conference*, 2018, https://doi.org/10.1007/978-3-319-96884-1_4.

[33] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic Regression on Homomorphic Encrypted Data at Scale," in *AAAI Conference on Artificial Intelligence*, 2019, https://doi.org/10.1609/aaai.v33i01.33019466.

[34] K. Han and D. Ki, "Better Bootstrapping for Approximate Homomorphic Encryption," in *Cryptographers' Track at the RSA Conference*, 2020, https://doi.org/10.1007/978-3-030-40186-3_16.

[35] S. Hong, S. Kim, J. Choi, Y. Lee, and J. H. Cheon, "Efficient Sorting of Homomorphic Encrypted Data With k-Way Sorting Network," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4389–4404, 2021, https://doi.org/10.1109/TIFS.2021.3106167.

[36] G. Jonatan, H. Cho, H. Son, X. Wu, N. Livesay, E. Mora, K. Shivdikar, J. L. Abellán, A. Joshi, D. Kaeli, and J. Kim, "Scalability Limitations of Processing-in-Memory using Real System Evaluations," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 8, no. 1, 2024, https://doi.org/10.1145/3639046.

[37] J. H. Ju, J. Park, J. Kim, M. Kang, D. Kim, J. H. Cheon, and J. Ahn, "NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping," in *ACM Conference on Computer and Communications Security*, 2024, https://doi.org/10.1145/3658644.3690375.

[38] W. Jung, S. Kim, J. Ahn, J. H. Cheon, and Y. Lee, "Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 114–148, 2021, https://doi.org/10.46586/tches.v2021.i4.114-148.

[39] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. Ahn, "Accelerating Fully Homomorphic Encryption Through Architecture-Centric Analysis and Optimization," *IEEE Access*, vol. 9, pp. 98 772–98 789, 2021, https://doi.org/10.1109/ACCESS.2021.3096189.

[40] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate Homomorphic Encryption with Reduced Approximation Error," in *Cryptographers' Track at the RSA Conference*, 2022, https://doi.org/10.1007/978-3-030-95312-6_6.

[41] D. Kim, J. Park, J. Kim, S. Kim, and J. Ahn, "HyPHEN: A Hybrid Packing Method and Its Optimizations for Homomorphic Encryption-Based Neural Networks," *IEEE Access*, vol. 12, pp. 3024–3038, 2024, https://doi.org/10.1109/ACCESS.2023.3348170.

[42] J. Kim, J. Seo, and Y. Song, "Simpler and Faster BFV Bootstrapping for Arbitrary Plaintext Modulus from CKKS," in *ACM Conference on Computer and Communications Security*, 2024, https://doi.org/10.1145/3658644.3670302.

[43] J. H. Kim, Y. Ro, J. So, S. Lee, S.-h. Kang, Y. Cho, H. Kim, B. Kim, K. Kim, S. Park, J.-S. Kim, S. Cha, W.-J. Lee, J. Jung, J.-G. Lee, J. Lee, J. Song, S. Lee, J. Cho, J. Yu, and K. Sohn, "Samsung PIM/PNM for Transfmer Based AI : Energy Efficiency on PIM/PNM Cluster," in *IEEE Hot Chips 35 Symposium*, 2023, https://doi.org/10.1109/HCS59251.2023.10254711.

[44] J. Kim, W. Choi, and J. Ahn, "Cheddar: A Swift Fully Homomorphic Encryption Library for CUDA GPUs," 2024, https://arxiv.org/abs/2407.13055.

[45] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. Ahn, "SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption," in *ISCA*, 2023, https://doi.org/10.1145/3579371.3589053.

[46] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. Ahn, "ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse," in *MICRO*, 2022, https://doi.org/10.1109/MICRO56248.2022.00086.

[47] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. Ahn, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *ISCA*, 2022, https://doi.org/10.1145/3470496.3527415.

[48] Y.-J. Kim, H.-J. Kwon, S.-Y. Doo, M. Ahn, Y.-H. Kim, Y.-J. Lee, D.-S. Kang, S.-G. Do, C.-Y. Lee, G.-H. Cho, J.-K. Park, J.-S. Kim, K. Park, S. Oh, S.-Y. Lee, J.-H. Yu, K. Yu, C. Jeon, S.-S. Kim, H.-S. Park, J.-W. Lee, S.-H. Cho, K.-W. Park, Y. Kim, Y.-H. Seo, C.-H. Shin, C.-Y. Lee, S.-Y. Bang, Y. Park, S.-K. Choi, B.-C. Kim, G.-H. Han, S.-J. Bae, H.-J. Kwon, J.-H. Choi, Y.-S. Sohn, K.-I. Park, S.-J. Jang, and G. Jin, "A 16-Gb, 18-Gb/s/pin GDDR6 DRAM With Per-Bit Trainable Single-Ended DFE and PLL-Less Clocking," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 197–209, 2019, https://doi.org/10.1109/JSSC.2018.2883395.

[49] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions," in *International Conference on Machine Learning*, 2022, https://proceedings.mlr.press/v162/lee22e.html.

[50] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Net-

work," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022, https://doi.org/10.1109/ACCESS.2022.3159694.

[51] J. Lee, E. Lee, Y. Lee, Y. Kim, and J. No, "High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2021, https://doi.org/10.1007/978-3-030-77870-5_22.

[52] S. Lee, G. Lee, J. W. Kim, J. Shin, and M.-K. Lee, "HETAL: Efficient Privacy-preserving Transfer Learning with Homomorphic Encryption," in *International Conference on Machine Learning*, 2023, https://proceedings.mlr.press/v202/lee23m.html.

[53] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications," in *IEEE International Solid-State Circuits Conference*, 2022, https://doi.org/10.1109/ISSCC42614.2022.9731711.

[54] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, S. O, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product," in *ISCA*, 2021, https://doi.org/10.1109/ISCA52012.2021.00013.

[55] Y. Lee, J.-W. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and H. Kang, "High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2022, https://doi.org/10.1007/978-3-031-06944-4_19.

[56] lightbulb128, "Troy-Nova," 2024, https://github.com/lightbulb128/troy-nova.

[57] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," *IEEE Computer Architecture Letters*, pp. 1–4, 2023, https://doi.org/10.1109/LCA.2023.3333759.

[58] A. C. Mert, E. Öztürk, and E. Savaş, "Design and Implementation of a Fast and Scalable NTT-Based Polynomial Multiplier Architecture," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, https://doi.org/10.1109/DSD.2019.00045.

[59] S. Narasimha, B. Jagannathan, A. Ogino, D. Jaeger, B. Greene, C. Sheraw, K. Zhao, B. Haran, U. Kwon, A. K. M. Mahalingam, B. Kannan, B. Morganfeld, J. Dechene, C. Radens, A. Tessier, A. Hassan, H. Narisetty, I. Ahsan, M. Aminpur, C. An, M. Aquilino, A. Arya, R. Augur, N. Baliga, R. Bhelkar, G. Biery, A. Blauberg, N. Borjemscaia, A. Bryant, L. Cao, V. Chauhan, M. Chen, L. Cheng, J. Choo, C. Christiansen, T. Chu, B. Cohen, R. Coleman, D. Conklin, S. Crown, A. da Silva, D. Dechene, G. Derderian, S. Deshpande, G. Dilliway, K. Donegan, M. Eller, Y. Fan, Q. Fang, A. Gassaria, R. Gauthier, S. Ghosh, G. Gifford, T. Gordon, M. Gribelyuk, G. Han, J. Han, K. Han, M. Hasan, J. Higman, J. Holt, L. Hu, L. Huang, C. Huang, T. Hung, Y. Jin, J. Johnson, S. Johnson, V. Joshi, M. Joshi, P. Justison, S. Kalaga, T. Kim, W. Kim, R. Krishnan, B. Krishnan, K. Anil, M. Kumar, J. Lee, R. Lee, J. Lemon, S. Liew, P. Lindo, M. Lingalugari, M. Lipinski, P. Liu, J. Liu, S. Lucarini, W. Ma, E. Maciejewski, S. Madisetti, A. Malinowski, J. Mehta, C. Meng, S. Mitra, C. Montgomery, H. Nayfeh, T. Nigam, G. Northrop, K. Onishi, C. Ordonio, M. Ozbek, R. Pal, S. Parihar, O. Patterson, E. Ramanathan, I. Ramirez, R. Ranjan, J. Sarad, V. Sardesai, S. Saudari, C. Schiller, B. Senapati, C. Serrau, N. Shah, T. Shen, H. Sheng, J. Shepard, Y. Shi, M. Silvestre, D. Singh, Z. Song, J. Sporre, P. Srinivasan, Z. Sun, A. Sutton, R. Sweeney, K. Tabakman, M. Tan, X. Wang, E. Woodard, G. Xu, D. Xu, T. Xuan, Y. Yan, J. Yang, K. Yeap, M. Yu, A. Zainuddin, J. Zeng, K. Zhang, M. Zhao, Y. Zhong, R. Carter, C. Lin, S. Grunow, C. Child, M. Lagus, R. Fox, E. Kaste, G. Gomba, S. Samavedam, P. Agnello, and D. K. Sohn, "A 7nm CMOS Technology Platform for Mobile and High Performance Compute Application," in *IEEE International Electron Devices Meeting*, 2017, https://doi.org/10.1109/IEDM.2017.8268476.

[60] H. Nejatollahi, S. Gupta, M. Imani, T. S. Rosing, R. Cammarota, and N. Dutt, "CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware," in *ACM/IEEE Design Automation Conference*, 2020, https://doi.org/10.1109/DAC18072.2020.9218730.

[61] NVIDIA Corporation, "NVML API Reference," Mar 2024, https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html.

[62] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *MICRO*, 2017, https://doi.org/10.1145/3123939.3124545.

[63] J. Park, D. Kim, J. Kim, S. Kim, W. Jung, J. H. Cheon, and J. Ahn, "Toward Practical Privacy-Preserving Convolutional Neural Networks Exploiting Fully Homomorphic Encryption," in *Workshop on Data Integrity and Secure Cloud Computing*, 2023, https://disccworkshop.org/assets/pdfs/DISCC_2023_paper_4.pdf.

[64] J. Park, M. J. Kim, W. Jung, and J. Ahn, "AESPA: Accuracy Preserving Low-Degree Polynomial Activation for Fast Private Inference," 2022, https://arxiv.org/abs/2201.06699.

[65] M.-J. Park, H. S. Cho, T.-S. Yun, S. Byeon, Y. J. Koo, S. Yoon, D. U. Lee, S. Choi, J. Park, J. Lee, K. Cho, J. Moon, B.-K. Yoon, Y.-J. Park, S.-m. Oh, C. K. Lee, T.-K. Kim, S.-H. Lee, H.-W. Kim, Y. Ju, S.-K. Lim, S. G. Baek, K. Y. Lee, S. H. Lee, W. S. We, S. Kim, Y. Choi, S.-H. Lee, S. M. Yang, G. Lee, I.-K. Kim, Y. Jeon, J.-H. Park, J. C. Yun, C. Park, S.-Y. Kim, S. Kim, D.-Y. Lee, S.-H. Oh, T. Hwang, J. Shin, Y. Lee, H. Kim, J. Lee, Y. Hur, S. Lee, J. Jang, J. Chun, and J. Cho, "A 192-Gb 12-High 896-GB/s HBM3 DRAM with a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization," in *IEEE International Solid-State Circuits Conference*, 2022, https://doi.org/10.1109/ISSCC42614.2022.9731562.

[66] Y. Park, S. Pal, A. Amarnath, K. Swaminathan, W. D. Lu, A. Buyuktosunoglu, and P. Bose, "DRAMATON: A Near-DRAM Accelerator for Large Number Theoretic Transforms," *IEEE Computer Architecture Letters*, pp. 1–4, 2024, https://doi.org/10.1109/LCA.2024.3381452.

[67] R. Podschwadt and D. Takabi, "Classification of Encrypted Word Embeddings using Recurrent Neural Networks," in *Workshop on Privacy in Natural Language Processing*, 2020, https://ceur-ws.org/Vol-2573/PrivateNLP_Paper3.pdf.

[68] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An Architecture for Computing on Encrypted Data," in *ASPLOS*, 2020, https://doi.org/10.1145/3373376.3378523.

[69] S. S. Roy, F. Turan, K. Järvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data," in *HPCA*, 2019, https://doi.org/10.1109/HPCA.2019.00052.

[70] Y. Ryu, S.-G. Ahn, J. H. Lee, J. Park, Y. K. Kim, H. Kim, Y. G. Song, H.-W. Cho, S. Cho, S. H. Song, H. Lee, U. Shin, J. Ahn, J.-M. Ryu, S. Lee, K.-H. Lim, J. Lee, J. H. Park, J.-S. Jeong, S. Joo, D. Cho, S. Y. Kim, M. Lee, H. Kim, M. Kim, J.-S. Kim, J. Kim, H. G. Kang, M.-K. Lee, S.-R. Kim, Y.-C. Kwon, Y. Y. Byun, K. Lee, S. Park, J. Youn, M.-O. Kim, K. Sohn, S.-J. Hwang, and J. Lee, "A 16 GB 1024 GB/s HBM3 DRAM With Source-Synchronized Bus Design and On-Die Error Control Scheme for Enhanced RAS Features," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1051–1061, 2023, https://doi.org/10.1109/JSSC.2022.3232096.

[71] 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*, 2021, https://doi.org/10.1145/3466752.3480070.

[72] 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, https://doi.org/10.1145/3470496.3527393.

[73] Samsung Electronics Co., Ltd., "8Gb GDDR6 SGRAM C-die," Tech. Rep. K4Z80325BC, 2020, https://www.lcsc.com/datasheet/lcsc_datasheet_2204251615_Samsung-K4Z80325BC-HC14_C2920181.pdf.

[74] K. Shivdikar, Y. Bao, R. Agrawal, M. Shen, G. Jonatan, E. Mora, A. Ingare, N. Livesay, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, "GME: GPU-based Microarchitectural Extensions to Accelerate Homomorphic Encryption," in *MICRO*, 2023, https://doi.org/10.1145/3613424.3614279.

[75] S. Sinha, S. Hung, D. Fisher, X. Xu, C. Chao, P. Chandupatla, F. Frederick, H. Perry, D. Smith, A. Cestero, J. Safran, V. Ayyavu, M. Bhargava, R. Mathur, D. Prasad, R. Katz, A. Kinsbruner, J. Garant, J. Lubguban, S. Knickerbocker, V. Soler, B. Cline, R. Christy, T. McLaurin, N. Robson, and D. Berger, "A High-Density Logic-on-Logic 3DIC Design Using Face-to-Face Hybrid Wafer-Bonding on 12nm FinFET Process," in *IEEE International Electron Devices Meeting*, 2020, https://doi.org/10.1109/IEDM13553.2020.9372120.

[76] SK hynix, "Advanced Packaging Technology for Beyond Memory," p. 29, 2023, https://www.theise.org/wp-content/uploads/2023/10/

Tutorial1-4_%EC%86%90%ED%98%B8%EC%98%81%EC%88%98%EC%84%9D%EB%8B%98_SK%ED%95%98%EC%9D%B4%EB%8B%89%EC%8A%A4.pdf.

[77] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A CUDA-Accelerated Word-Wise Homomorphic Encryption Library," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–12, 2024, https://doi.org/10.1109/TDSC.2024.3363900.

[78] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical Homomorphic Encryption Accelerator," in *HPCA*, 2023, https://doi.org/10.1109/HPCA56546.2023.10070984.

[79] A. W. B. Yudha, J. Xue, Q. Lou, H. Zhou, and Y. Solihin, "BoostCom: Towards Efficient Universal Fully Homomorphic Encryption by Boosting the Word-wise Comparisons," in *International Conference on Parallel Architectures and Compilation Techniques*, 2024, https://doi.org/10.1145/3656019.3676893.

[80] S. Yun, K. Kyung, J. Cho, J. Choi, J. Kim, B. Kim, S. Lee, K. Sohn, and J. Ahn, "Duplex: A Device for Large Language Models with Mixture of Experts, Grouped Query Attention, and Continuous Batching," in *MICRO*, 2024, https://doi.org/10.1109/MICRO61859.2024.00105.