Towards GPU Accelerated FHE Computations

Orion Papadakis, Michail Papadimitriou, Athanasios Stratikopoulos, Maria Xekalaki,
Juan Fumero, Nikos Foutris and Christos Kotselidis
Department of Computer Science
The University of Manchester
Manchester, United Kingdom
{first}.{last}@manchester.ac.uk

***Abstract*—Fully homomorphic encryption (FHE) enables processing encrypted data without revealing sensitive information, making it applicable in fields like healthcare, finance, and legal. Despite its benefits, FHE has high computational complexity and performance overhead. To address this, researchers have explored hardware acceleration using Field-Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs). FPGAs are suitable for low-latency computations, while GPUs excel in parallel, high-throughput tasks. However, widespread FHE adoption remains elusive due to unresolved performance issues.**

This paper explores the challenges of offloading FHE computations to hardware accelerators, focusing on the OpenFHE library and the Brakerski-Gentry-Vaikuntanathan (BGV) scheme. It is the first study on adapting this scheme for GPU acceleration. We profile OpenFHE to identify computational bottlenecks and propose integrating parallelized CUDA computations within OpenFHE. Our solution, tested with varying numbers of multiplicative depth, shows up to 26x performance improvement over non-accelerated implementations, proving the effectiveness of GPUs for FHE. However, the end-to-end performance is still up to 2x slower due to the overhead of marshaling and moving data between the CPU and GPU, accounting for over 97% of execution time.

***Index Terms*—data privacy, fully homomorphic encryption, hardware acceleration, GPUs**

I. Introduction

The continuous growth of high volumes of data has posed significant challenges regarding the fast transferring and processing of the information. Nonetheless, they are not the only crucial factors. Security and data privacy are factors of paramount importance, especially in industrial domains in which the information may be sensitive, such as healthcare, finance [1], intellectual property, legal, and more.

Fully homomorphic encryption (FHE) [2] is a privacy-preserving mechanism that enables the processing of encrypted data without exposing sensitive information [3]. Hence, in a scenario where data must be given to a third-party software for analysis, the data owner can receive the result of the analysis without sharing the actual data or the decryption key.

However, the enhanced security and data privacy come at the cost of increased computational complexity, thereby resulting in a high-performance overhead. To tackle this inefficiency, prior work has employed hardware acceleration platforms, such as Field-Programmable Gate Arrays (FPGAs) [4]–[6] and Graphics Processing Units (GPUs) [7]–[11]. Although both hardware platforms are suitable for increasing the performance of parallel software implementations, it is evident that there is still no definite solution for enabling the widespread adoption of FHE in industry. The reason is that there are several challenges when offloading FHE computations on hardware acceleration platforms, and ultimately, they can impact the trade-off between data privacy, performance and scalability.

In this paper, we discuss these challenges and aim to rethink how they can be alleviated in order to enable a harmonic coexistence between hardware acceleration, and more specifically GPUs, and FHE computations. To comprehend these challenges better, we employ OpenFHE [12], an open-source FHE library that does not offer GPU acceleration support by default. Thus, OpenFHE is an ideal candidate library to study the integration of GPU acceleration capabilities. This paper makes the following contributions:

- We discuss the advantages and disadvantages of the FHE libraries and the hardware acceleration platforms in order to comprehend the source of the research challenges.
- We perform a profiling analysis of OpenFHE that reveals the computationally expensive parts of the encrypted computations.
- We propose our work-in-progress solution that includes the automatic integration of a parallelized computation implemented in CUDA within OpenFHE.
- Finally, we present an experimental performance analysis that shows our preliminary results, which highlight the current obstacles towards accelerating FHE libraries.

II. Background

A. Fully Homomorphic Encryption

FHE is a novel privacy-preserving mechanism designed to allow the computation of encrypted data without sharing any confidential information regarding the actual data or the decryption key. Figure 1 presents the process for performing a computation over encrypted data with FHE. This process consists of four steps: Step ❶ : The generation of a pair of keys, a public key used for encryption (Step 2) and a private key used for decryption (Step 4). Step ❷ : The encryption of data which utilizes the public key generated by Step 1 and produces a ciphertext. Step ❸ : The performance of homomorphic operations over the encrypted data (i.e., ciphertext). The main concept behind that step is that the computation uses as input a ciphertext (from Step 2) and the outcome is also a ciphertext. Common homomorphic operations include

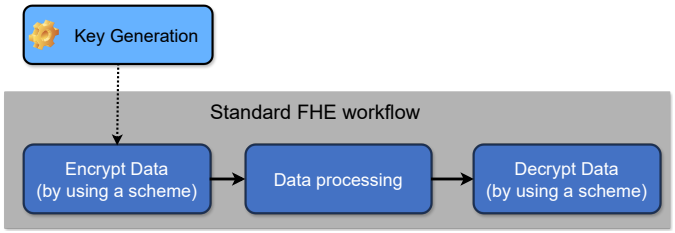

Fig. 1. The standard process for performing FHE computations.

addition and multiplication, which can be combined to perform more complex computations. Step ❹ : The decryption of the produced ciphertext that holds the result of the processing (Step 3). After decryption, the result matches the result of the same computation as performed on the input plaintext.

In the first step, the keys are generated by using complex mathematical formulas, typically based on lattice-based cryptography [13], [14], which is resistant to quantum attacks [15]. The formulas are related to the applied encryption/decryption schemes (or encryption systems), and they mathematically define all steps described above. Brakerski/Fan-Vercauteren (BFV) [16], Brakerski-Gentry-Vaikuntanathan (BGV) [17], Cheon-Kim-Kim-Song (CKKS) [18], Fast Fully Homomorphic Encryption over Torus (TFHE) [19] and Gentry-Sahai-Waters (GSW) are the state-of-the-art schemes supported by many FHE libraries. Each scheme has unique characteristics that determine its suitability in particular use cases. For instance, BFV is designed for integer arithmetic operations, CKKS is suitable for arithmetic on real numbers, TFHE is good for binary and boolean operations, and finally BGV is ideal for polynomial operations. Most of these schemes aim to find the balance between the efficiency of the arithmetic homomorphic operations and the management of the accumulated noise. Every homomorphic operation adds noise to the generated ciphertext. As more operations are performed, the noise level increases, thereby requiring special care to ensure that the generated ciphertext can be decrypted successfully. To address this challenge, many schemes support bootstrapping, a highly computationally intensive process that reduces the noise from the ciphertext and enables more homomorphic operations to be processed [13].

B. Hardware Acceleration

To program hardware accelerators, many programming models (e.g., OpenCL, CUDA, oneAPI) have been provided by hardware vendors. A programming model offers a defined process that programmers must follow to simplify programming for hardware accelerators (e.g., a GPU, an FPGA) [20]. This process is common between most programming models and consists of three steps (Figure 2), as follows: Step ❶ : The transferring of the data to be processed from the program (i.e., CPU main memory) into the device memory (i.e., DRAM). Step ❷ : The parallel processing of the data. The executed computation is usually defined in source code (e.g., CUDA C, OpenCL C, DPC++, VHDL) or binary code (e.g., SPIR-V) that is being compiled by the device driver prior to the execution.

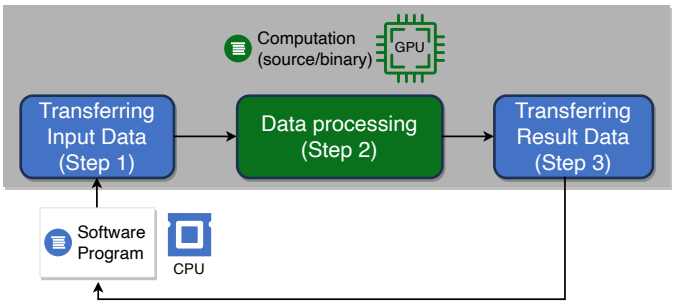

Fig. 2. The workflow of heterogeneous programming models.

Step ❸ : The transferring of the result of the computation from the device memory back to the CPU main memory.

The performance of different workloads and algorithmic patterns can vary depending on the device type (GPU or FPGA). For instance, GPUs excel in high-throughput computations with thousands of threads running the same instructions on different data. FPGAs can deploy custom hardware blocks and allow the reconfiguration of their on-device resources like memory blocks and logic slices at runtime.

C. Research Challenges

This section discusses the primary challenges that must be addressed to enable the smooth integration of modern hardware acceleration technologies within state-of-the-art FHE libraries. In a nutshell, these challenges are attributed to the incompatibility of data types between FHE libraries and the hardware acceleration programming models (Section II-C2), the difficulty in parallelizing FHE algorithms (Section II-C1), the lack of scalability for large input data sizes (Section II-C3), and the management of the increase noise in the ciphertext (Section II-C4).

1) Difficulty in Parallelizing FHE Algorithms: This challenge emerges due to the sequential nature of the FHE algorithms (i.e., schemes) since they have been designed mostly to increase the data privacy and enhance the security of the computations, rather than considering the offloading of computationally expensive parts on hardware accelerators. Hence, the adaptation of the computationally expensive algorithmic parts into parallel implementations that can harness the power of hardware accelerators is not straightforward and requires a high engineering cost due to the consideration of numerous parallel programming concepts, such as *load balancing* and *data synchronization*.

2) Incompatible Data Types: Besides the difficulties in parallelizing FHE algorithms, a second challenge emerges due to incompatible data types. The majority of FHE libraries employ common data structures for the representation of the *ciphertext*, the *plaintext*, the *public key*, the *secret key*, and more. Each data structure is composed of many types of information that is useful for its entity. For instance, the *ciphertext* contains information regarding the FHE scheme information that is being used, the encryption parameters (e.g., the polynomial modulus degree, the coefficient modulus), as well as the level of noise that exists in the ciphertext.

To represent all the information stored in a *ciphertext* into a program (known also as kernel) that is going to be executed on a hardware accelerator device, it is necessary to map each information type into a data type that is supported by the hardware acceleration programming models (e.g., OpenCL, CUDA). These programming models offer primarily support for scalar data types (e.g., *int*, *float*, *long*, *double*), vectors of scalar values (e.g., *int4, int8*), matrices and arrays (e.g., *floatnxm*) and secondarily for more specialized data types, such as for image processing (e.g., *image2d_t*) [21].

Thus, the incompatibility between the data types that are supported by the FHE libraries and the heterogeneous programming models hinders the productivity of fast parallel implementations that could potentially be offloaded on hardware accelerators (e.g., GPUs, FPGAs). It can also impact the overall performance of the FHE library due to the necessity of *marshaling/unmarshaling*, a process responsible for converting data types from one format to another, and vice-versa.

3) Lack of Scalability: The memory footprint of the keys and the ciphertexts can impact the memory latency, the noise level of the ciphertext and the storage requirements of modern FHE libraries. Additionally, the ability to scale up the size of input encrypted data without sacrificing performance is of high importance. To achieve this goal, it is useful to optimize the memory access pattern of the FHE algorithms by considering CPU optimization techniques (e.g., caching, pre-fetching), and hardware acceleration optimization techniques, such as the unified memory between CPU and accelerators [22].

In the scope of this paper, we will emphasize to the latter class of optimization techniques. It has been evidential by prior work that hardware accelerators can increase the performance of the computation performed by the FHE schemes [7], [23], as will be discussed in Section IV-B. Despite the performance improvement by offloading the execution of a kernel on a GPU or the FPGA, there is an equally crucial part related to: i) the data transfers (Steps 1 and 3, discussed in Section II-B), and ii) how often the computation described in the kernel accesses the DRAM memory. To address the first part, some heterogeneous programming models, such as CUDA and Intel oneAPI LevelZero, have provided support for enabling a unified address space between the CPU and GPUs [24]. Regarding the second part, the parallel implementations of the kernels in OpenCL, CUDA or DPC++ can utilize the allocation of local memory [7], [25] and data synchronization mechanisms (i.e., barriers) [20]. Thus, the memory access pattern of the FHE algorithms along with the cost for transferring data can both impact the performance of FHE computations.

4) Satisfaction of Noise Management: Finally, the level of noise that is accumulated as the encrypted data are being computed can play a very important role to ensure that the resulted ciphertext is decryptable. Hence, bootstrapping and other mechanisms for reducing the noise of ciphertexts can be computationally expensive and can potentially harness hardware acceleration [26].

TABLE I
PROFILING RESULTS OF THE SIMPLE-INTEGERS-BGVRNS OPENFHE EXAMPLE, OBTAINED WITH THE CLION PROFILER.

Classification	Function	%	Invoc. Count
crypto context	genCryptoContextBGVRNSInternal	5.5	1
key generation	EvalAtIndexKeyGen	34.6	1
key generation	EvalMultKeyGen	8.7	1
key generation	KeyGen	4.6	1
encryption	MakePackedPlaintext	1.2	1
encryption	Encrypt	8.0	3
compute	EvalRotate	17.5	5
compute	EvalMult	12.1	2
compute	EvalAdd	<1	2
decryption	Decrypt	6.9	6

III. GPU ACCELERATION SUPPORT FOR OPENFHE

The work that we present in this paper aims to study the first three challenges discussed in the previous section (Section II-C). Most importantly, in this section we employ OpenFHE as the primary FHE library for our study. In particular, we describe a profiling analysis of the computationally expensive parts in OpenFHE for the execution of homomorphic computations with the BGV scheme (Section III-A). Then, we present the design and implementation of a parallel implementation of the computationally heavy parts in CUDA (Section III-B). Finally, we discuss our experimental findings in Section III-C.

A. Identifying Opportunities for Acceleration

1) An Example of FHE Computations: We used an example from the OpenFHE GitHub repository [27] that encrypts three input vectors and performs fully homomorphic computations: i) two additions on the same ciphertext, ii) two multiplications with a different ciphertext, and iii) five rotations. Each vector has twelve integer values. After computation, the results are decrypted and compared to expected values.

2) Profiling Analysis: Using the CLion CPU profiler based on *perf* and *DTrace*, we obtained the metrics in Table I. Functions are classified by role (key generation, encryption, decryption, compute), and the table shows the percentage of the execution time for each function and invocation count (Invoc. Count). The most computationally expensive functions are *EvalAtIndexKeyGen*, *EvalRotate* and *EvalMult* which consume in total 64.2% of the total execution time of the program. *EvalAtIndexKeyGen* is invoked once during initialization, thus its impact decreases as computation complexity increases. In the compute phase, *EvalRotate* accounts for 17.5%, *EvalMult* for 12.1%, and *EvalAdd* for less than 1%. Considering invocation count, *EvalMult* is the most expensive per invocation. Consequently, we analyzed *EvalMult* and identified *ApproxSwitchCRTBasis* as a key function for GPU acceleration.

B. Design & Implementation

The identified *ApproxSwitchCRTBasis* function performs a modular multiplication, multiplication (*mul128* function), and a barrett reduction. It has iterative data processing (loops) with no data dependencies, allowing parallel execution on a

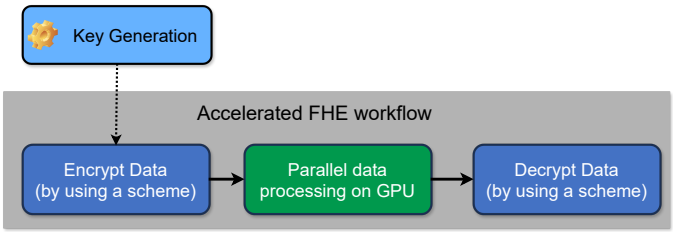

Fig. 3. The GPU accelerated flow for performing parallel FHE computations.

TABLE II
EXPERIMENTAL TESTBED.

Hardware	
Processor	Intel Core i7-13700 @ 5.20GHz
Cores	12 (24 HyperThreads)
RAM	64GB
GPU	NVIDIA GeForce RTX 3070 (Ampere) 8GB GDDR6, 5888 CUDA Cores
Software	
Operating System	PopOS 22.04 LTS (Kernel 6.9.3-76-generic)
CUDA Driver	550.67 (CUDA 12.4)
OpenFHE	v1.0.3

GPU. Figure 3 shows the parallel implementation flow of a fully homomorphic multiplication in OpenFHE, substituting the computation part with a GPU-offloadable parallel implementation. We began by defining appropriate data types supported by heterogeneous hardware models. Using 128-bit integers, supported by CUDA since version 11.5, we transformed `for` loops into thread indices for GPU threads. The size of GPU threads and workgroups of our implementation will be presented in the experimental analysis (Section III-C).

C. Experimental Evaluation

To evaluate the performance of our parallel implementation, we conducted experiments on a testbed that contains both CPU and GPU hardware. The hardware and software specifications of the testbed are presented in Table II. All the reported measurements discussed in this section are the average of one hundred executions for the accelerated and the baseline implementations. For the CPU, we used the `std::chrono` C++ library to obtain precise timing data, while GPU performance metrics were collected by utilizing the NVIDIA Nsight Systems[1] tool.

Regarding the configuration of the OpenFHE parameters, Table III presents the configuration of the multiplicative input depth, modulus, cyclotomic, and ring dimensions for our experiments. Additionally, it reports the GPU block and thread allocations that we evaluated for each depth. These parameters were chosen to showcase the performance of homomorphic encryption computations as well as the required parallel threads deployed onto the GPU across various depth values.

1) Performance Analysis: In this paragraph, we compare the GPU kernel execution time and the end-to-end execution time of the accelerated implementation (including kernel execution, data transfers, and data handling) against the baseline

[1]https://developer.nvidia.com/nsight-systems

TABLE III
CONFIGURATION PARAMETERS FOR OPENFHE AND THEIR CORRESPONDING GPU BLOCK AND THREAD ALLOCATIONS FOR VARIOUS MULTIPLICATIVE DEPTHS.

Input Depth	Modulus	Cyclotomic	Ring Dimension	Blocks	Threads
1	65,537	16,384	8,192	8	1,024
5	65,537	32,768	16,384	16	1,024
12	65,537	65,536	32,768	32	1,024
24	786,433	131,072	65,536	64	1,024

TABLE IV
THE EXECUTION TIME OF THE GPU KERNEL, THE END-TO-END ACCELERATED IMPLEMENTATION AND THE BASELINE NON-ACCELERATED IMPLEMENTATION OF OPENFHE ACROSS DIFFERENT INPUT MULTIPLICATION DEPTHS.

Input Depth	GPU Time (ms)	End-to-End Accel. Impl. (ms)	End-to-End Baseline Impl. (ms)
1	0.05	531	0.68
5	0.5	603	13
12	5	836	123
24	60	2273	1182

implementation of the *EvalMult* function. Table IV reports all the execution times as measured in our experimental testbed (Table II) in milliseconds. Figure 4 presents the speedup of the GPU kernel and the end-to-end accelerated implementation versus the baseline implementation. As shown, the GPU kernel outperforms the baseline implementation for all the input depths with a 26x maximum performance achieved for depth 5. On the other hand, the end-to-end accelerated implementation presents a significant performance slowdown ranging from 0.0013x (input depth 1) to 0.52x (input depth 24).

To understand better the reason behind the observed performance penalty shown in the end-to-end accelerated implementation we performed a breakdown analysis of the overall executed time in three main parts: i) the GPU kernel execution time (i.e., the portion that shows performance speedup in Figure 4), ii) the data transfers, which include the time spent for transferring data from the CPU to the GPU memory, and backwards; and iii) the rest of the time which is spent for the *marshaling/unmarshaling* of the data and the allocation of the memory buffers in the GPU memory. As shown in Figure 5, for small input depths which present the biggest performance slowdown the time for marshaling/unmarshaling and allocating the GPU buffers dominates the overall time by 99.97%. This indicates the significance of investing effort to converge the incompatibility of the data types between FHE libraries and heterogeneous programming models (Section II-C2).

Finally, it is shown that if the input depth increases the percentage of that portion decreases up to 67.40% (input depth 24), due to the fact that the arithmetic complexity of the computation performed on the GPU and the input data sizes also increase. Thus, GPUs can be utilized to increase performance of FHE computations for big input depths (large data sizes).

IV. RELATED WORK

This section reviews the state-of-the-art FHE libraries and their hardware acceleration integration.

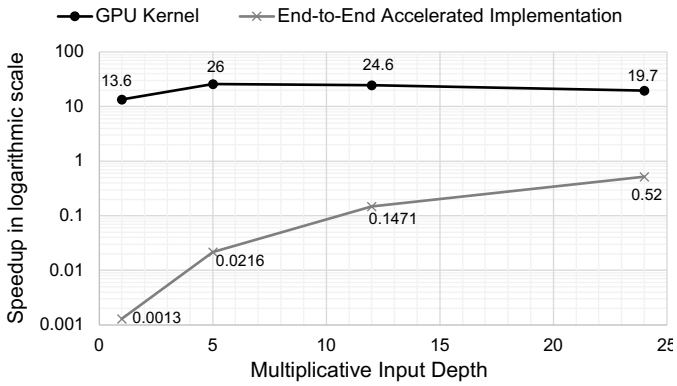

Fig. 4. Performance speedup of the GPU kernel and the end-to-end accelerated implementation against the baseline.

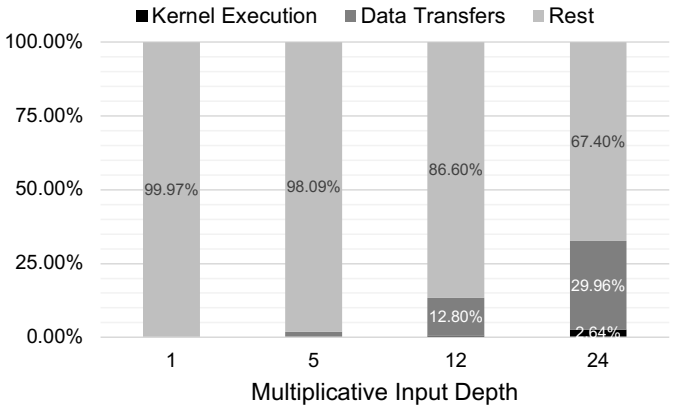

Fig. 5. Breakdown of the end-to-end time of the accelerated implementation into three main parts: the kernel execution, the data transfers and the rest of the time.

A. State-of-the-art FHE Libraries

Table V summarizes the FHE libraries and their supported encryption schemes and hardware acceleration. All libraries are open-source, fostering community-driven development. Microsoft SEAL [7], Lattigo [28], and PALISADE support BFV, BGV, and CKKS schemes. TenSEAL [29], is built on SEAL and focuses on tensor operations. TenSEAL and HEaaN support BFV and CKKS, while HElib [30] supports BGV and CKKS. TFHE [19] and Concrete support TFHE. OpenFHE [12] is designed by creators of PALISADE, HElib, HEaaN, and FHEW [31] to offer high usability and performance.

B. Hardware Acceleration Support

OpenFHE includes a hardware abstraction layer (HAL) to enhance homomorphic operations efficiency. The HAL layer currently supports the Intel HEXL library utilizing the Intel Advanced Vector Extensions (AVX) [12]. Microsoft SEAL and PALISADE also use Intel HEXL for homomorphic encryption acceleration [32].

Research has explored GPU efficiency for homomorphic operations in FHE libraries such as Microsoft SEAL [7] and TFHE [23]. Badawi et al. [8] proposed a parallel GPU implementation of the BFV scheme outperforming Microsoft SEAL and NFLlib-FV. Another study [9] analyzed multi-threaded CPU and GPU implementations of the BFV scheme for PALISADE. Wang et al. [10] presented the first GPU implementation of the Gentry-Halevi FHE scheme [14]. Wang et al. [11] also proposed a parallel GPU implementation of the levelled FHE scheme. FPGA support has also been extensively studied. Intel provided an open-source homomorphic encryption acceleration library for FPGAs (Intel Hexl FPGA), including various kernel implementations and API calls for third-party FHE libraries. However, the repository was archived in December 2023. Agrawal et al. [4] designed an FPGA-based architecture for accelerating RLWE-based somewhat homomorphic encryption. Riazi et al. [5] introduced HEAX, a high-performance architecture for fast modular arithmetic on encrypted data, built on a parallelizable Number-Theoretic Transform (NTT) engine. Research has also examined shared memory conflicts and thread divergence in cuHE NTT kernels [25]. Sinha et al. [6] proposed an FPGA-based architecture for the BFV scheme, emphasizing the importance of fast external memory for high performance.

Our work is the first, to our knowledge, to provide profiling analysis and a parallel implementation for the BGV scheme in full FHE mode.

TABLE V
OVERVIEW OF THE FHE LIBRARIES. THIS TABLE PRESENTS ALSO THE SUPPORTED ENCRYPTION SCHEMES OF EACH LIBRARY AND WHETHER HARDWARE ACCELERATION (HA) IS SUPPORTED.

Libraries	Scheme	HA Support
Microsoft SEAL	BFV, BGV, CKKS	AVX
TenSEAL	BFV, CKKS	No
Lattigo	BFV, BGV, CKKS	No
Concrete	TFHE	No
TFHE	TFHE	No
PALISADE	BFV, BGV, CKKS	AVX
HElib	BGV, CKKS	No
HEaaN	BFV, CKKS	GPU
OpenFHE	BFV, BGV, CKKS, DM (FHEW), CGGI (TFHE)	AVX

V. CONCLUSIONS

This paper has discussed the challenges of enabling hardware acceleration for FHE computations, while also using the OpenFHE library for experimental insights. The preliminary results of our work show that notable performance improvements can be achieved from integrating parallelized CUDA computations for the BGV scheme into OpenFHE, thereby highlighting a promising direction for overcoming existing barriers to FHE widespread adoption in industry.

In the future, we plan to investigate the impact of unified memory and other optimizations in order to tackle the identified performance overhead.

ACKNOWLEDGMENT

This work is supported by the European Union's Horizon Europe programme under grant agreement No 101070670 (ENCRYPT). In addition, this work is funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee (10039809).

REFERENCES

[1] J. Ménétrey, M. Pasin, P. Felber, V. Schiavoni, G. Mazzeo, A. Hollum, and D. Vaydia, "A Comprehensive Trusted Runtime for WebAssembly with Intel SGX," IEEE Transactions on Dependable and Secure Computing, pp. 1–18, 2023.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, ser. STOC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440

[3] L. Coppolino, S. D'Antonio, G. Mazzeo, L. Romano, and L. Sgaglione, "PriSIEM: Enabling privacy-preserving Managed Security Services," Journal of Network and Computer Applications, vol. 203, p. 103397, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S108480452200056X

[4] R. Agrawal, L. Bu, and M. A. Kinsy, "Fast Arithmetic Hardware Library For RLWE-Based Homomorphic Encryption," in 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2020, pp. 206–206.

[5] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An Architecture for Computing on Encrypted Data," in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1295–1309. [Online]. Available: https://doi.org/10.1145/3373376.3378523

[6] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-Based Multicore Processor for FV Somewhat Homomorphic Function Evaluation," IEEE Transactions on Computers, vol. 67, no. 11, pp. 1637–1650, 2018.

[7] Ö. Özerk, C. Elgezen, A. C. Mert, E. Öztürk, and E. Savaş, "Efficient number theoretic transform implementation on GPU for homomorphic encryption," The Journal of Supercomputing, vol. 78, no. 2, pp. 2840–2872, 2022. [Online]. Available: https://doi.org/10.1007/s11227-021-03980-5

[8] A. A. Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-Performance FV Somewhat Homomorphic Encryption on GPUs: An Implementation using CUDA," IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 2, p. 70–95, May 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/875

[9] A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and Performance Evaluation of RNS Variants of the BFV Homomorphic Encryption Scheme," IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 2, pp. 941–956, 2021.

[10] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in 2012 IEEE Conference on High Performance Extreme Computing, 2012, pp. 1–5.

[11] W. Wang, Z. Chen, and X. Huang, "Accelerating leveled fully homomorphic encryption using GPU," in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014, pp. 2800–2803.

[12] A. A. Badawi, A. Alexandru, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, C. Pascoe, Y. Polyakov, I. Quah, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-Source Fully Homomorphic Encryption Library," Cryptology ePrint Archive, Paper 2022/915, 2022, https://eprint.iacr.org/2022/915. [Online]. Available: https://eprint.iacr.org/2022/915

[13] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford, CA, USA, 2009, aAI3382729.

[14] C. Gentry and S. Halevi, "Implementing Gentry's Fully-Homomorphic Encryption Scheme," in Advances in Cryptology – EUROCRYPT 2011, K. G. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–148.

[15] C. Gentry, "Toward Basing Fully Homomorphic Encryption on Worst-Case Hardness," in Advances in Cryptology – CRYPTO 2010, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 116–137.

[16] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[17] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 309–325. [Online]. Available: https://doi.org/10.1145/2090236.2090262

[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in Advances in Cryptology – ASIACRYPT 2017, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster Fully Homomorphic Encryption: Bootstrapping in less than 0.1 Seconds," Cryptology ePrint Archive, Paper 2016/870, 2016, https://eprint.iacr.org/2016/870. [Online]. Available: https://eprint.iacr.org/2016/870

[20] J. Fumero, A. Stratikopoulos, and C. Kotselidis, Heterogeneous Programming Models. Cham: Springer International Publishing, 2024, pp. 37–56. [Online]. Available: https://doi.org/10.1007/978-3-031-49559-5_3

[21] K. O. W. Group, "The OpenCL C Specification," Online: https://www.khronos.org/registry/OpenCL/specs/3.0-unified/html/OpenCL_C.html, Last Access: June 2024.

[22] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards high performance paged memory for gpus," in 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016, pp. 345–357.

[23] T. Morshed, M. M. A. Aziz, and N. Mohammed, "Cpu and gpu accelerated fully homomorphic encryption," in 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2020, pp. 142–153.

[24] J. Fumero, F. Blanaru, A. Stratikopoulos, S. Dohrmann, S. Viswanathan, and C. Kotselidis, "Unified shared memory: Friend or foe? understanding the implications of unified memory on managed heaps," in Proceedings of the 20th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes, ser. MPLR 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 143–157. [Online]. Available: https://doi.org/10.1145/3617651.3622984

[25] A. A. Badawi, B. Veeravalli, and K. M. Mi Aung, "Faster number theoretic transform on graphics processors for ring learning with errors based cryptography," in 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), 2018, pp. 26–31.

[26] M. S. Lee, Y. Lee, J. H. Cheon, and Y. Paek, "Accelerating bootstrapping in FHEW using GPUs," in 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2015, pp. 128–135.

[27] "The source code of the Simple Integers BGVrns example from the OpenFHE GitHub repository." Online: https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/simple-integers-bgvrns.cpp, Last Access: June 2024.

[28] C. Mouchet, J.-P. Bossuat, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Lattigo: a Multiparty Homomorphic Encryption Library in Go," in Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC). HomomorphicEncryption.org, 2020. [Online]. Available: https://homomorphicencryption.org/wp-content/uploads/2020/12/wahc20_demo_christian.pdf

[29] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," 2021.

[30] S. Halevi and V. Shoup, "Design and implementation of HElib: a homomorphic encryption library," Cryptology ePrint Archive, Paper 2020/1481, 2020, https://eprint.iacr.org/2020/1481. [Online]. Available: https://eprint.iacr.org/2020/1481

[31] L. Ducas and D. Micciancio, "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second," in Advances in Cryptology – EUROCRYPT 2015, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 617–640.

[32] F. Boemer, S. Kim, G. Seifu, F. D. M. de Souza, and V. Gopal, "Intel HEXL: accelerating homomorphic encryption with intel AVX512-IFMA52," CoRR, vol. abs/2103.16400, 2021. [Online]. Available: https://arxiv.org/abs/2103.16400