SHORT-PAPER

MLIR-Based Homomorphic Encryption Compiler for GPU

AI NOZAKI, The University of Tokyo, Tokyo, Japan

TAKUYA KOJIMA, The University of Tokyo, Tokyo, Japan

HIROSHI NAKAMURA, The University of Tokyo, Tokyo, Japan

HIDEKI TAKASE, The University of Tokyo, Tokyo, Japan

Open Access Support provided by:

The University of Tokyo

Published: 19 June 2024

Citation in BibTeX format

HEART 2024: 14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies
June 19 - 21, 2024
Porto, Portugal

https://doi.org/10.1145/3665283.3665343
ISBN: 9798400717277

MLIR-Based Homomorphic Encryption Compiler for GPU

Ai NOZAKI
nozaki@hal.ipc.i.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

Takuya KOJIMA
The University of Tokyo
Japan

Hiroshi NAKAMURA
The University of Tokyo
Japan

Hideki TAKASE
The University of Tokyo
Japan

ABSTRACT

Homomorphic Encryption (HE) enables computing over encrypted data and has attracted attention as an approach to data protection with the spread of cloud computing. To tackle the expensive computational cost of HE, hardware accelerators have been studied. However, developing HE applications and accelerating them with hardware is challenging because it requires a deep understanding of both HE and hardware. Previous work proposed compilers to automate the development of HE applications, but most of them target CPUs, so automation for accelerators is not well established. This work proposes an MLIR-based HE compiler that automates the transformation of applications into HE and accelerates the process using GPU. This compiler enables users to perform GPU acceleration without requiring knowledge of GPU and its tools.

ACM Reference Format:
Ai NOZAKI, Takuya KOJIMA, Hiroshi NAKAMURA, and Hideki TAKASE. 2024. MLIR-Based Homomorphic Encryption Compiler for GPU. In *14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART'24)) (HEART '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3665283.3665343

1 INTRODUCTION

In recent years, the use of cloud computing has become widespread. Since data in the cloud server is processed in a decrypted state, data confidentiality cannot be guaranteed. HE has attracted attention as an approach to address this problem. HE allows arithmetic or binary operations on encrypted data, eliminating the need to decrypt data on an untrusted server. Although HE is an ideal means of data protection, it is computationally expensive. The execution time of one multiplication is $10^4 - 10^5$ times longer than that of unencrypted multiplication. Therefore, hardware accelerators have been used to compute HE operations efficiently. It is useful to take advantage of the massively parallel nature of GPUs and to design architectures specialized for HE as FPGAs and ASICs. Using accelerators, applications such as ResNet can be executed in a few microseconds per multiplication, with realistic execution times [1].

HEART '24, June 19–21, 2024, Porto, Portugal

ACM ISBN 979-8-4007-1727-7/24/06
https://doi.org/10.1145/3665283.3665343

However, it is tough to develop HE applications for these accelerators. Developing HE applications requires knowledge of cryptography, such as noise management and selecting parameters. Furthermore, porting them to accelerators requires knowledge of hardware, which increases complexity. For example, developers have to understand hardware architecture and use proprietary tools. Previous work proposed compilers [2-4] to facilitate the development of HE applications, but most of them target CPUs. Automation for porting HE applications to accelerators is not well established.

This work proposes an MLIR-based HE compiler that automatically transforms applications into HE and accelerates using GPU. This compiler enables users to accelerate applications on GPU without knowledge of GPU and its tools.

2 BACKGROUND

2.1 Homomorphic Encryption

HE enables operations to be performed on encrypted data. CKKS[2] scheme, proposed in 2019, can perform operations such as addition and multiplication on complex numbers. The ciphertext is represented as a pair of polynomials over residue rings, where the dimension N of the polynomials is usually $N > 2^{12}$. Since the size of a ciphertext can reach several tens of MB, the computational cost of HE operations is very high.

Hardware accelerators have been used to efficiently compute HE operations. 100x[6], cuHE[3], and TensorFHE[5] utilize a large number of threads on GPUs. BTS[8], F1[11], and ARK[7] are ASIC-based accelerators. They use a large number of PEs (8192-18432) and on-chip memory (64-512MB) and design specialized architecture for HE operations. F1 reports a 17,000x speedup in execution time on MNIST workloads compared to execution on CPUs[11].

2.2 Difficulties in developping HE applications

Developing applications on HE is challenging in that it requires a great deal of cryptographic knowledge. For example, the noise added to the ciphertext should be managed to prevent failing decryption. The noise is added during operations and can be reduced by a process called bootstrapping, but the computational cost of bootstrapping is very high, so it is necessary to consider where to insert the bootstrapping. It is also necessary to set parameters large enough to meet security requirements, but this involves a trade-off with the computational costs. Furthermore, the application must be composed of primitive operations that can be computed on the HE scheme.

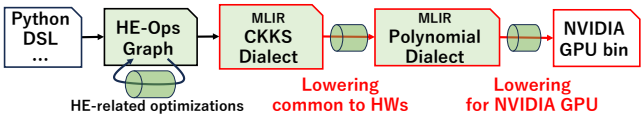

Figure 1: Overview of the proposed compiler

The complexity increases when applications on HE are accelerated by hardware. Libraries for the target accelerators are mostly written in proprietary languages like CUDA, so developers have to use those languages and tools. In addition, it is crucial to manage the data transfer between the device and the host. Thus, developing efficient and recoverable HE applications is challenging, requiring knowledge of both cryptography and hardware.

2.3 Related Works

Previous work proposed compilers to automate the development of HE applications. Recently proposed compilers such as HECO[13], EVA[4], and HECATE[10] support the CKKS scheme and automate optimizations that require cryptographic knowledge, such as the conversion to operations on HE primitive operations, noise management, and batching by extracting parallelism for the input application. Most of them leave the internal implementation part of HE to CPU-targeted OSS libraries like SEAL[12]. Thus, constraints specific to hardware accelerators, such as the data transfer to the device and the hiding of proprietary tools, have not yet been fully achieved.

2.4 MLIR

MLIR(Multi-Level Intermediate Representations)[9] is a compiler platform developed by the LLVM project. MLIR is designed to compile an input program with a lowering process, which converts higher-level intermediate representations(IR) to lower-level IR. The IR is represented using multiple Dialects, which is a mechanism for defining types, functions, and attributes in MLIR. Users easily define domain-specific Dialects and import or export them. Since HE involves multi-stage optimization, many of the recently proposed HE compilers are based on MLIR.

3 COMPILER DESIGN

Figure1 is an overview of the proposed compiler. The part shown in red is proposed in this paper. The compiler takes a Python or DSL program as an input, ports it to HE, and accelerate it using NVIDIA GPU. This is realized via multi stage compilation: (i) Program Transformations and HE-related optimizations, which transforms an input program into HE-Ops (e.g., Homomorphic Add, Homomorphic Mult) compurational graph and optimizing the graph using the knowledge of underlying scheme, (ii) Lowering HE-Ops, which describe the algorithms of each HE-Ops using polynomial operations, and (iii) Code generation for NVIDIA GPU, which generates executable code based on polynomial operations. This multi stage compilation is realized via MLIR and we define CKKS Dialect and Polynomial Dialect. The compilation step (i) can be realized by using an existing compiler such as HECO.

Table 1: Execution Time

	SEAL(CPU)		100x(GPU)		This Work	
logN	14	15	14	15	14	15
L	32	44	32	44	32	44
logQ	438	881	1576	2379	1751	2406
HADD(ms)	0.123	0.873	0.005	0.006	0.008	0.009
HMULT(ms)	18.0	108.3	1.37	2.01	3.89	7.48

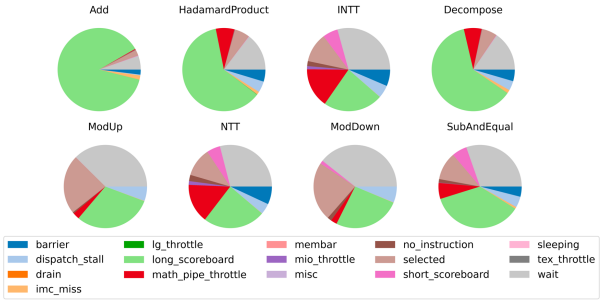

Figure 2: Profiled stall reason for each kernel

Design of Polynomial Dialect: Polynomial Dialect is defined as a boundary for common and individual compilations among accelerators. With this Dialect as a boundary, a lowering pass below the Polynomial Dialect is all that is needed to support the new accelerator to this compiler.

Conversion from CKKS Dialect to Polynomial Dialect: In this conversion, as a common process among accelerators, lowering from CKKS Dialect to Polynomial Dialect is performed. The Polynomial Dialect defines operations such as add, mult, ntt, intt, modup, moddown, and the CKKS operations can be represented by dozens of Polynomial operations.

Conversion from Polynomial Dialect to GPU executable: In this conversion, Polynomial Dialect operations are transformed into accelerator executables as a specialized process for each accelerator. In this work, we support NVIDIA GPUs and implement them in CUDA. Specialized optimizations for each accelerator can be applied in this conversion. For example, accessing global memory (DRAM) in a CUDA application is costly, requiring hundreds of cycles. 32 threads execute the same instruction in CUDA, and when these 32 threads access consecutive addresses, accesses to DRAM are coalesced into a single access. To achieve such accesses to consecutive addresses, we selected the number of threads at kernel startup and extracted parallelism.

4 PRELIMINARY EVALUATION

Setup. In this experiment, we evaluate and analyze the performance of the GPU implementation generated from this compiler. This evaluation runs on Intel(R) Xeon(R) w9-3495X CPU @ 1.90GHz with 502GiB RAM and NVIDIA GeForce RTX 4090 with 24GB RAM. For benchmarking, we perform addition and multiplication with two different parameters.

Performance analysis. We compare the performance with a state-of-the-art GPU implementation of 100x[6] and a CPU implementation of Microsoft SEAL[12]. Table1 shows the result of executing addition and multiplication. The GPU implementation of the compiler is about 4 – 14 times faster than the CPU implementation of SEAL. On the other hand, the performance is only 26 – 35% better than that of the existing GPU library, 100x. To analyze the detailed performance, we conduct a profiling using NVIDIA Nsight Compute[1]. Figure2 shows an analysis of CUDA kernel stalls. The "selected" is the percentage of runs without stalls, and the others are the percentage of stalls due to some reason. Add and HadamardProduct kernels are memory-intensive operations, and the majority of stalls are due to memory accesses. ModUp, ModDown, and (i)NTT kernels have a large percentage of stalls due to "wait." In this state, Warp was stalled and is waiting on a fixed latency execution dependency. Since this percentage should be small in optimized kernels, our implementation requires tuning.

5 CONCLUSION

This paper proposes an MLIR-based HE compiler for GPUs using MLIR. This compiler enables users to perform GPU acceleration without knowledge of CUDA and related tools. In the future, we pursue better performance by parallelizing polynomial operations and overlapping data transfer and computation. In addition, we aim to support other accelerators and enable generating codes for multiple accelerators with the same input.

ACKNOWLEDGMENTS

This work was supported by JST, PRESTO Grant Number JPMJPR22P5, Japan.

REFERENCES

[1] 2023. NVIDIA Nsight Compute. https://developer.nvidia.com/nsight-compute.
[2] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 409–437.
[3] Wei Dai and Berk Sunar. 2016. cuHE: A homomorphic encryption accelerator library. In *Cryptography and Information Security in the Balkans: Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers 2*. Springer, 169–186.
[4] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 2020. EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN conference on programming language design and implementation*. 546–561.
[5] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. Tensorfhe: Achieving practical computation on encrypted data using gpgpu. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 922–934.
[6] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 114–148.
[7] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1237–1254.
[8] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 711–725.
[9] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2–14.
[10] Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. 2022. HECATE: Performance-aware scale optimization for homomorphic encryption compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 193–204.
[11] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 238–252.
[12] SEAL 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..
[13] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. 2023. {HECO}: Fully Homomorphic Encryption Compiler. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4715–4732.