Accelerating Garbled Circuits by Hardware-Software Co-Design

Full version available in HAAC [13]

Jianqiao Mo, Brandon Reagen
New York University
{jm8782, bjr5}@nyu.edu

ABSTRACT

Privacy-preserving computation (PPC) has gained traction to address rising security and privacy concerns. With PPC, functions are computed directly on encrypted data to secure it during computation. Garbled circuits (GCs) are a PPC technology that enable both confidential computing and control over how data is used. The challenge is that they incur significant performance overheads compared to plaintext. Recently, we proposed HAAC [13], a novel hardware-software co-designed GCs accelerator and compiler, to mitigate performance overheads and make PPC more practical. The key idea of our approach is to express arbitrary GCs programs as streams, which simplifies hardware and enables complete memory-compute decoupling. GCs program has a key feature that it is data oblivious, thus we developed a scratchpad that captures data reuse by tracking program execution, eliminating the need for costly hardware managed caches and tagging logic. We evaluate HAAC with VIP-Bench and achieve an average speedup of 589× with DDR4 (2,627× with HBM2) in 4.3mm^2 of area.

1. INTRODUCTION

The growing importance of security and privacy underscores the need for new solutions that offer strong data protection. Privacy-preserving computation (PPC) stands out by providing users with two key advantages: confidentiality and control. Confidential computing enables computation on encrypted data, ensuring that service providers cannot view users' sensitive data while still providing them high-quality services. In addition, certain techniques (namely secure multiparty computation) allow users to control how their data is used, dictating which functions their data is computed with. While promising, the wide adoption of all cryptographically strong PPC techniques is hindered by high computational overheads, which are too high for most applications today. To overcome these hurdles and usher in a new era of private computing, we require innovative hardware solutions.

Numerous PPC techniques are available; we concentrating on garbled circuits (GCs). Each has its own strengths and weaknesses. The foreseeable future likely contains a blend of techniques, as their strengths can be harnessed in combination to address respective limitations [2, 5, 9]. The intent of this paper is not to debate whether GCs or, for instance, Homomorphic Encryption (HE) is superior. Instead, our focus is on illustrating how hardware acceleration can effectively mitigate the performance overheads associated with GCs, making their advantages accessible and practical.

A significant advantage of GCs is the ability to support the implementation of arbitrary functions, such as conditional code (e.g., ReLU) and complex scientific (e.g., exponential). This capability allows GCs to be a crucial component in the execution of private neural inference (PI) [8, 14, 17], particularly for securely evaluating non-linear activation functions. In contrast, constructing non-linear with alternative techniques (e.g., arithmetic HE) often leads to a complex approximation, necessitating model retraining and resulting in accuracy drop. Meanwhile, prior research has highlighted that GCs introduce significant overhead, especially in the hybrid protocols which combine multiple PPCs to uphold high accuracy PI [6, 10, 11]. Recent studies have shown the critical role of accelerating GCs to achieve faster PI, but the associated overheads are substantial [4, 12]. Our experiment shows that, across eight VIP-Bench benchmarks [1], GCs are on average 198,000× slower than plaintext.

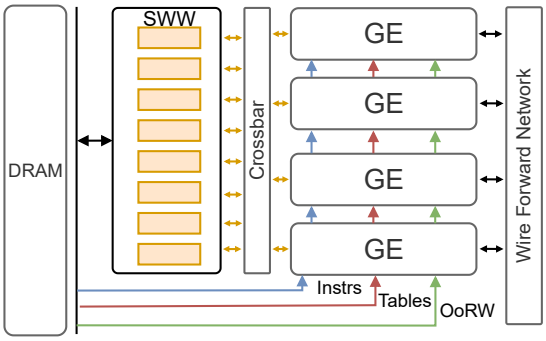

Figure 1: HAAC architecture assuming four GEs and eight SWW banks.

Multiple factors contribute to GC's high overhead. First, each GC gate involves a significant amount of computation, as they are cryptographic functions, distinct from plaintext gates. Single GCs AND gate can involve four AES calls, two key expansions (similar to AES), and a variety of 128-bit logic operations. Second, processing a function requires executing a large number of gates, as it must be expressed as Boolean logic. For example, executing a private Bubble Sort from VIP-Bench requires processing over 12 million gates. Third, GCs are data intensive. Each plaintext gate's inputs and output are represented as a 128-bit ciphertext, and each (AND) gate involves a unique, 32 Byte, cryptographic constant for processing. Therefore, addressing these challenges requires the design of a dedicated hardware accelerator to enhance the efficiency of GCs, as proposed in HAAC (full version in [13]), a hardware-software co-designed solution, making GCs computation more practical.

2. MOTIVATION / KEY INSIGHTS

There are two key properties that enable GCs amenable to effective hardware acceleration. First, the core computations, though complex, are highly suitable for hardware implementation. Leveraging custom-logic hardware and parallelism within gates, HAAC demonstrates significant performance improvement. Second, GCs are exemplars for hardware-software co-design as programs are completely data oblivious [1], i.e., all dependence, memory accesses, and control

flow are determined at compile time. We developed programmable hardware (i.e., ISA support) for executing any GCs program with high performance and efficiency by relying on the compiler to organize *all* data movement and instruction scheduling. As a result, gate computations can be parallelized, while data is seamlessly moving on- and off-chip to mask latency. Besides using on-chip scratchpad (SWW); the irregular data is allowed to stream. Thus, it enables decoupling the execution and data access, while still captures most data reuse. Reminiscent of VLIW, this eliminates costly hardware to extract performance at run-time, allowing more area to be dedicated to actual computation. Alternative approaches are possible but tend to be overly restrictive or unnecessary. Fixed-logic ASICs limit the arbitrary functional support GCs provide; Systolic arrays and vectors constrain how data is laid out and computation ordered, which can restrict hardware's ability to process all programs well [7]; Dataflow is wasteful, as the compiler can handle scheduling and avoid allocating costly structures.

3. OUR PROPOSAL

Building on the insights mentioned above, we proposed HAAC. This approach includes a compiler, ISA, and hardware accelerator, all working in concert to enhance the performance and efficiency of GCs.

HAAC accelerates individual gate execution through a dedicated datapath of the hardware unit, the gate engine (GE). While GEs have the potential for high-performance computing, the challenge is exploiting gate parallelism and orchestrating data movement (operands, constants, instructions) effectively while keeping hardware efficient. A pivotal insight is that, with hardware support, the compiler can express GCs as multiple streams. First, the compiler can leverage instruction-level parallelism to improve intra/inter-GE parallel processing. Knowing the precise order of events, instructions and constants can be streamed to each GE using queues. On the other hand, managing gate inputs and outputs, which are often referred to as *wires*, are more difficult as they do not follow a linear pattern. Streaming all wires on/off-chip is wasteful as they exhibit reuse. We first proposed the sliding wire window (SWW) and wire renaming. The SWW is a scratchpad memory that stores a contiguous address range of wires, and the range increases as the program executes. Renaming is a complementary compiler pass that sequentializes gate output wire addresses according to program order. Together, the SWW and renaming effectively filter off-chip accesses, as recently generated wires are often soon reused in the circuit. Thus, the SWW provides the performance benefits of a cache and efficiency of a scratchpad. It is a feasible solution, but raises another question – how to capture irregular wire reuse without additional hardware overhead? Most wire accesses are filtered by the SWW, while misses, or Out-of-Range (OoR) accesses, still occur as capacity is limited. OoR accesses cause significant performance degradation in HAAC's deep, in-order pipelines. To address this, our second wire optimization is to stream in OoR wires. The insight is that the compiler knows when and which wires will be OoR and can push them on-chip to an OoR wire queue. An important implication of the optimizations is the enabling of complete decoupling of gate execution and off-chip accesses, allowing for total overlap.

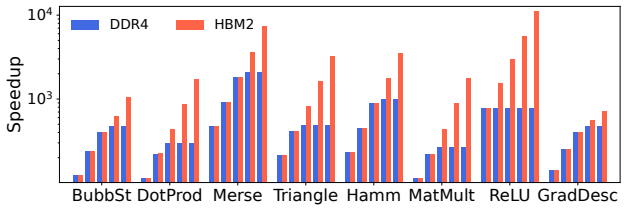

Figure 2: Benchmark performance scaling with respect to GE count, bar clusters show 1, 2, 4, 8, and 16 GEs and speedup is relative to the CPU.

To summarize, HAAC is a novel hardware-software co-design tailored to GCs, including gate engines (GEs) to accelerate gates, queues (instruction, table, and OoR wire), and scratchpad (SWW). The hardware is programmable and supports an ISA, with an optimizing compiler for parallel instruction scheduling (reordering), effective data layout and memory accesses (renaming), and removing unnecessary off-chip communication (eliminating spent wires).

Evaluation set-up and experiment details can be found in the full version [13]. The optimized GE is fully pipelined, capable of computing a GCs gate and accepting a new input every cycle. Figure 2 shows HAAC's speedup relative to running GCs with CPU. We evaluate performance by scaling GEs from 1 to 16 with two types of off-chip bandwidth: DDR4 to fairly compare with our CPU and HBM2 to understand how HAAC benefits from advanced memory technology. In most cases performance scales well but designs can saturate DDR4 bandwidth and become memory bound. With HBM2 the performance continues to scale across the range of GEs considered. Compared to CPU-run GCs, a HAAC accelerator achieves a geomean speedup of 589× with DDR4, and 2,627× with HBM2.

The total area for HAAC, including 16 GEs with a 2 MB SWW and a 64 KB SRAM, is 4.3 mm^2 in 16nm. In HAAC, most of the chip area goes to the GE. Given the small area footprint, we assume HAAC would be used as an IP in a larger SoC, and thus the HBM2 PHY [3, 15, 16] would be shared. The ultimate measure of performance in PPC is how well it performs relative to non-encrypted plaintext. With the same benchmarks, the geomean slowdown compared to plaintext is 76×. The GradDesc benchmark uses true floating point, and performing floating point securely is very expensive relative to plaintext. Considering only integer benchmarks, the geomean slowdown compared to plaintext is only 23×.

4. CONCLUSION

We proposed HAAC, a hardware-software co-design to accelerate GCs. It significantly enhances GCs' performance and efficiency. Our contributions encompass the gate engine (GE), customized memory structures for GCs' data, a high-performance compiler, and the sliding wire window (SWW) for efficient wire reuse.

Overall, HAAC greatly mitigates GCs' performance overhead, making the slowdown much more tolerable. We believe that, potential enhancements, including compiler optimizations, increased parallelism (multiple HAAC cores), and processing-in-memory (PIM), would further bridge this performance gap.

REFERENCES

[1] L. Biernacki, M. Z. Demissie, K. B. Workneh, G. B. Namomsa, P. Gebremedhin, F. A. Andargie, B. Reagen, and T. Austin, "Vip-bench: A benchmark suite for evaluating privacy-enhanced computation frameworks," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, 2021, pp. 139–149.

[2] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," *Cryptology ePrint Archive*, 2014.

[3] A. Feldmann, N. Samardzic, A. Krastev, S. Devadas, R. Dreslinski, K. Eldefrawy, N. Genise, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption (extended version)," 2021.

[4] K. Garimella, Z. Ghodsi, N. K. Jha, S. Garg, and B. Reagen, "Characterizing and optimizing end-to-end systems for private inference," *arXiv preprint arXiv:2207.07177*, 2022.

[5] H. Geng, J. Mo, D. Reis, J. Takeshita, T. Jung, B. Reagen, M. Niemier, and X. S. Hu, "Ppimce: An in-memory computing fabric for privacy preserving computing," *arXiv preprint arXiv:2308.02648*, 2023.

[6] Z. Ghodsi, A. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a relu budget," 2021.

[7] S. U. Hussain, B. D. Rouhani, M. Ghasemzadeh, and F. Koushanfar, "Maxelerator: Fpga accelerator for privacy preserving multiply-accumulate (mac) on cloud servers," *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[8] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1651–1669.

[9] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1575–1590.

[10] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 619–631. [Online]. Available: https://doi.org/10.1145/3133956.3134056

[11] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2505–2522. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[12] J. Mo, K. Garimella, N. Neda, A. Ebel, and B. Reagen, "Towards fast and scalable private inference," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023, pp. 322–328.

[13] J. Mo, J. Gopinath, and B. Reagen, "Haac: A hardware-software co-design to accelerate garbled circuits," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[14] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.

[15] NVIDIA, "Nvidia dgx station a100 system architecture," 2021. [Online]. Available: https://images.nvidia.com/aem-dam/Solutions/Data-Center/nvidia-dgx-station-a100-system-architecture-white-paper.pdf

[16] Rambus, "White paper: Hbm2e and gddr6: Memory solutions for ai," 2020. [Online]. Available: https://go.rambus.com/hbm2e-gddr6-memory-solutions-for-ai

[17] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018, pp. 707–721.