2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) | 978-1-7281-8643-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISPASS51385.2021.00045

Accelerating Fully Homomorphic Encryption Through Microarchitecture-Aware Analysis and Optimization

Wonkyung Jung*, Eojin Lee†, Sangpyo Kim*, Namhoon Kim*, Keewoo Lee*, Chohong Min‡,
Jung Hee Cheon*, and Jung Ho Ahn*
*Seoul National University, † Samsung Electronics, ‡ Ewha Woman's University
gajh@snu.ac.kr

I. Motivation

Homomorphic Encryption (HE) [11] draws significant attention as a privacy-preserving way for cloud computing because it allows computation on encrypted messages called ciphertexts. Among numerous FHE schemes [2]–[4], [8], [9], HE for Arithmetic of Approximate Numbers (HEAAN [3]), which is also known as CKKS (Cheon-Kim-Kim-Song), is rapidly gaining popularity [10] as it supports computation on real numbers. A critical shortcoming of HE is the high computational complexity of ciphertext arithmetic, especially, HE multiplication (HE Mul). For example, the execution time for computation on encrypted data (ciphertext) increases from 100s to 10,000s of times compared to that on native, unencrypted messages. However, a large body of HE acceleration studies, including ones exploiting GPUs and FPGAs, lack a rigorous analysis of computational complexity and data access patterns of HE Mul with large parameter sets on CPUs, the most popular computing platform.

II. A Brief Introduction to HEAAN

We introduce the pertinent details of HEAAN [1], [3]. Let L be the level (the number of HE Mul applicable to the ciphertext until losing data), p be rescaling factor, which is an integer, $Q = p^L$ be the maximum ciphertext modulus, and N be a power-of-two integer. A ciphertext **c** with a ciphertext modulus $q \leq Q$ is represented by two polynomials **c.ax**, **c.bx**, which are elements of a polynomial ring $\in \mathbb{Z}_q[X]/(X^N+1)$. A HE Mul of two such ciphertexts, followed by a *rescaling*, produces an output ciphertext whose ciphertext modulus is $q' = q/p$; because a message is a vector of complex numbers scaled-up by p, rescaling prevents the explosion of the message by dividing each coefficient of the output ciphertext by p.

For given input ciphertexts **c1** and **c2** having the same ciphertext modulus, HE Add is a relatively simple coefficient-wise addition between the polynomials of both ciphertexts; HE Mul, being more complicated, outputs **c3** by computing a tensor product first:

$$\textbf{c3.ax} = \textbf{c1.ax} \cdot \textbf{c2.ax},\ \textbf{c3.cx} = \textbf{c1.bx} \cdot \textbf{c2.bx},$$
$$\textbf{c3.bx} = \textbf{c1.ax} \cdot \textbf{c2.bx} + \textbf{c1.bx} \cdot \textbf{c2.ax},$$

Then the second step, called *relinearization*, updates the output ciphertext with an evaluation key (**evk**), which is an encryption of a square of secret key **sk** $\in \mathbb{Z}_Q[X]/(X^N+1)$ multiplied by Q, in order to enable the output ciphertext to be still representable with a polynomial pair:

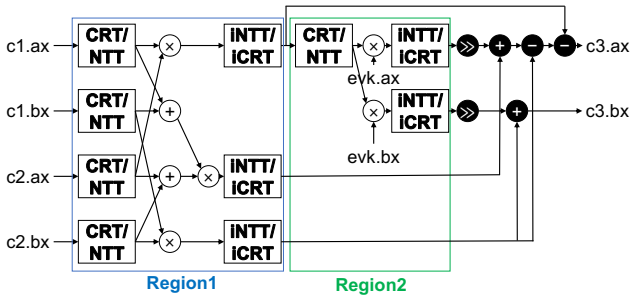


Fig. 1: The overall flow of HE Mul in HEAAN. A white (black) filled symbol represents an operation conducted in an RNS (BigInt) domain. Region 1 and region 2 use different moduli because the former multiplies two log$q$-bit numbers whereas the latter multiplies a log$q$-bit number with an evaluation key polynomial composed of logQ^2-bit numbers.

$$\textbf{c3.ax} = \textbf{c3.ax} + 1/Q \cdot \textbf{c3.cx} \cdot \textbf{evk.ax}$$
$$\textbf{c3.bx} = \textbf{c3.bx} + 1/Q \cdot \textbf{c3.cx} \cdot \textbf{evk.bx}$$

Because multiplication between polynomials, whose coefficients are big integers (BigInt), is costly, HE schemes use *Chinese Remainder Theorem* (CRT [7]) and *Number Theoretic Transform* (NTT [5]). CRT represents each BigInt coefficient $B < q$ in the residue number system (RNS) by the set of remainders $\{b_1, ..., b_{np}\}$, where $b_i = B \mod p_i$ for given np prime moduli $\{p_i\}_{1 \leq i \leq np}$, each of which is coprime to N and satisfies $p_i < \beta$ for a word size $\beta = 2^{64}$. Then, multiplication of two coefficients B_0,$B_1 < q$ is turned into a pointwise multiplication of two residue sets, each representing either B_0 or B_1 and having np residues such that $P = \prod_{i=1}^{np} p_i \geq q^2$. The output of the multiplication is obtained through applying *iCRT*, the inverse transform of CRT; for a given residue set $\{b'_i\}_{1 \leq i \leq np}$, we get the corresponding BigInt representation as follows: $\{\sum_{i=1}^{np} {}^{P}/_{p_i} \cdot ({}^{b'_i \cdot inv({}^{P}/_{p_i})} \mod p_i)\} \mod P = \{\sum_{i=1}^{np} \sum_{j=0}^{k} (({}^{P}/_{p_i})/\beta^j \mod \beta) \cdot \beta^j \cdot (b'_i \cdot inv({}^{P}/_{p_i}) \mod p_i)\} \mod P$, where $k = \log P / \log \beta$ and $inv(\cdot)$ means modular multiplicative inverse.

NTT, a variant of discrete Fourier transform over a finite field of integers, translates polynomial multiplication with $\mathcal{O}(N^2)$ complexity into element-wise multiplication exploiting fast NTT, a variant of FFT limited to integer values, reducing the complexity to $\mathcal{O}(N \log N)$. The overall flow of HE Mul in HEAAN is shown in Figure 1.

978-1-7281-8643-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPASS51385.2021.00045

III. Microarchitecture-aware Optimizations to Maximize HE Mul performance

We focus on the four functions, CRT, iCRT, NTT, and inverse NTT (iNTT), which account for 17.1%, 48.9%, 14.6%, and 14.8% of a HE Mul time, respectively, with the reference parameter set $(q, p, L) = (2^{1200}, 2^{30}, 40)$. They have massive parallelism: the outputs of CRT, which are $N \times np$ residues, can be computed in parallel, whereas NTT and iNTT perform $N/2 \times np$ independent butterfly operations at a time. We identify key challenges and solutions in accelerating HE Mul by focusing on the four major functions.

Loop reordering in iCRT: We exchange the two loops of iCRT to maximize data reuse. By reordering the two loops, a temporal BigInt value that accumulates the partial sum of the innermost loop becomes a double-word value. Since k temporal values can be parallelly computed, the loop reordering expands the N-degree parallelism of iCRT into $N \cdot k$-degree, providing abundant parallelization opportunities to contemporary hardware platforms.

Exploiting SIMD and multiple cores: We populate multiple cores and use AVX-512 SIMD instructions to exploit the massive parallelism of the four major functions of HE Mul, and distribute operations to multiple threads and AVX-512 lanes by carefully considering data access patterns. In CRT, a CPU thread takes the responsibility of a portion of the N coefficients, whereas each AVX-512 lane computes on different residues. In NTT/iNTT, a thread computes on a portion of the prime numbers, whereas each AVX-512 lane computes different coefficients. In iCRT, each thread computes on a part of the N coefficients, and an AVX-512 lane multiplies the i-th residues with $inv(P/p_i)$ first. Then, a lane computes on different j, the positional index on the limbs of P/p_i.

Emulating arithmetic operations: Because AVX-512 does not support parallel 64b mul and 64b ADC (addition with carry) yet, we emulate the instructions with shift, add, and mul. Also, we modify Shoup's modular multiplication (ModMul) [12], which is used in CRT and (i)NTT, by removing one 32b mul required for carry computation for emulating 64b mulhi. It changes the range of output before a correction step in Shoup's ModMul from $[0, 3p_j]$ to $[0, 4p_j]$, requiring one extra conditional subtraction, but saving one 32b mul, two 64b add, and one 64b shift.

Transposition in `iCRT`: iCRT experiences a poor cache utilization when we place the data of a polynomial in N-major order. We implicitly transpose the input of iCRT using the *scatter* instructions in AVX-512 to address this issue.

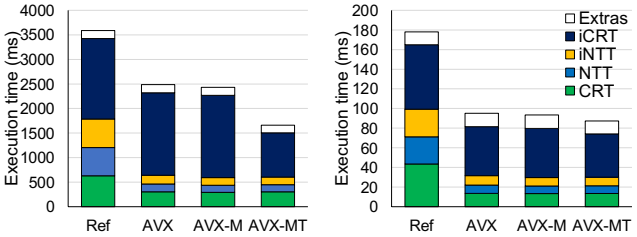
IV. Evaluation

We measured the performance of HE Mul in the reference HEAAN [1] and our AVX-512 implementation with an Intel CPU (Xeon Platinum 8160 operating at 2.1 GHz), using one socket with six memory channels, each with DDR4-2666 DRAM modules.

With the reference parameters (q, p, L)=$(2^{1200}, 2^{30}, 40)$, we evaluated the proposed optimizations in accelerating HEAAN mul by comparing against the performance of the reference HEAAN (**Ref**). We compared the basic implementation utilizing AVX-512 (**AVX**), the one with the modified Shoup's ModMul on top of **AVX** (**AVX-M**), and the one with implicit transposition in iCRT on top of **AVX-M** (**AVX-MT**).

TABLE I: Comparing the execution time of HE Mul among a single- and 24-thread reference HEAAN (**Ref**-1 and **Ref**-24), and a 24-thread optimized AVX-512 implementation (**AVX-MT**-24).

	Execution time (ms) and [Relative speedup]		
Func	**Ref**-1	**Ref**-24	**AVX-MT**-24
`CRT`	628.6	43.5 [14.4×]	13.5 [46.6×]
`NTT`	576.5	27.6 [20.9×]	7.8 [73.6×]
`iNTT`	582.7	28.2 [20.7×]	8.7 [66.6×]
`iCRT`	1636.6	65.7 [24.9×]	44.1 [37.1×]
Extra	162.3	13.1 [12.4×]	13.1 [12.3×]
Total	3586.6	178.0 [20.1×]	87.3 [**41.1**×]



(a) Execution time on single-thread (b) Execution time on 24-threads

Fig. 2: Comparing HE Mul execution time among **Ref**, AVX-512 implementation (**AVX**), and optimized **AVX** (**AVX-M** and **AVX-MT**) when (a) one and (b) 24 threads are utilized.

First, by applying a series of microarchitecture-aware optimizations, **AVX-MT** achieves 41.1× speedup, compared to the single-thread **Ref** (see Table I). Second, **AVX** provides 1.4× and 1.9× speedups over **Ref** when a single and 24 threads are utilized, respectively (see Figure 2). Also, it experiences 26.1× speedup by increasing the number of threads from 1 to 24, exhibiting better scalability than **Ref** (20.1×); **Ref** is more susceptible to a load imbalance across threads, and hardware prefetching hits more frequently in **AVX** for its higher spatial locality. In **AVX-M**, both NTT and iNTT are 10% faster than **AVX**, and iCRT experiences a 17% speedup in **AVX-MT** compared to **AVX-M** by alleviating the memory-bound issue; they are less effective in the multi-threaded case. Finally, we analyzed the performance limiting factors of a single HE Mul in **AVX-MT** by measuring the number of operations per second (OPS) in terms of AVX instructions (see Table II). We obtained 63.9% and 53.2% of the machine peak OPS (764.7GOPS) in CRT and iCRT, respectively.

From the Top-down microarchitecture analysis [13] on CRT and iCRT using the Intel VTune profiler [6], we observed that the portion of core bound (due to a sub-optimal use of the available execution units in the CPU [13]) on CRT and iCRT is 36.2% and 35.5%, respectively, which is dominant besides the

TABLE II: The number of executed AVX-512 operations per second (OPS) of each function per HE Mul. DRAM bandwidth utilization is measured in region 2 of Figure 1.

	CRT	NTT	iNTT	iCRT
Number of operations	6.58G	2.10G	2.44G	17.9G
OPS	**489G**	269G	279G	**407G**
DRAM bandwidth (GB/s)	13.0	**54.4**	**53.8**	46.5

retiring category (where the CPU pipeline is busy with useful operations), whose portion is 56.7% and 45.0%. In contrast, NTT and iNTT achieve lower OPS as they are more often bound to the limited throughput of the memory hierarchy.

REFERENCES

[1] "HEAAN with Faster Multiplication," https://github.com/snucrypto/HEAAN/releases/tag/2.1, Sep. 2018.

[2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, 2014.

[3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017.

[4] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption Over the Torus," *Journal of Cryptology*, 2018.

[5] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of computation*, vol. 19, no. 90, 1965.

[6] I. Corporation, "Intel VTune Profiler User Guide," https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top.html, September 2020.

[7] P. Dingyi, S. Arto, and D. Cunsheng, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific, 1996.

[8] 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.

[9] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *The International Association for Cryptologic Research Cryptology ePrint Archive*, vol. 2012, 2012.

[10] iDASH Privacy Protection Challenge 2018, "Secure Genome Analysis Competition," http://www.humangenomeprivacy.org/2018/, 2018.

[11] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On Data Banks and Privacy Homomorphisms," *Foundations of Secure Computation*, vol. 4, no. 11, 1978.

[12] 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 *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[13] A. Yasin, "A Top-down Method for Performance Analysis and Counters Architecture," in *ISPASS*, 2014.