Algorithmic Acceleration of B/FV-like Somewhat Homomorphic Encryption for Compute-Enabled RAM

Jonathan Takeshita, Dayane Reis, Ting Gong, Michael Niemier, X. Sharon Hu, and Taeho Jung

University of Notre Dame, Notre Dame IN 46556, USA
{jtakeshi,dreis,tgong,mniemier,shu,tjung}@nd.edu

Abstract. Somewhat Homomorphic Encryption (SHE) allows arbitrary computation with finite multiplicative depths to be performed on encrypted data, but its overhead is high due to memory transfer incurred by large ciphertexts. Recent research has recognized the shortcomings of general-purpose computing for high-performance SHE, and has begun to pioneer the use of hardware-based SHE acceleration with hardware including FPGAs, GPUs, and Compute-Enabled RAM (CE-RAM). CE-RAM is well-suited for SHE, as it is not limited by the separation between memory and processing that bottlenecks other hardware. Further, CE-RAM does not move data between different processing elements. Recent research has shown the high effectiveness of CE-RAM for SHE as compared to highly-optimized CPU and FPGA implementations. However, algorithmic optimization for the implementation on CE-RAM is underexplored. In this work, we examine the effect of existing algorithmic optimizations upon a CE-RAM implementation of the B/FV scheme [18], and further introduce novel optimization techniques for the Full RNS Variant of B/FV [6]. Our experiments show speedups of up to 784x for homomorphic multiplication, 143x for decryption, and 330x for encryption against a CPU implementation. We also compare our approach to similar work in CE-RAM, FPGA, and GPU acceleration, and note general improvement over existing work. In particular, for homomorphic multiplication we see speedups of 506.5x against CE-RAM [33], 66.85x against FPGA [35], and 30.8x against GPU [3] as compared to existing work in hardware acceleration of B/FV.

Keywords: Somewhat Homomorphic Encryption · B/FV Scheme · Full-RNS Variant · Compute-Enabled RAM

1 Introduction

Fully homomorphic encryption (FHE), first presented by Gentry [19], allows arbitrary number of additions and multiplications on ciphertexts. It has wide applications in settings where data privacy is paramount, but the *bootstrapping* procedure required to allow an arbitrary number of operations on ciphertexts is highly complex and computationally intensive. Therefore, most active research

including this work focuses on improving the efficiency of the underlying Somewhat Homomorphic Encryption (SHE) scheme [10, 35, 33, 6, 34], in which a ciphertext can be operated on arbitrarily up to a certain multiplicative depth. Even without the bootstrapping procedure, SHE schemes lead to high overhead in performing homomorphic operations with ciphertexts due to the large size of the ciphertexts.

Recent research recognizes limitations of traditional general-purpose computing in high-performance realizations of SHE [35, 34]. This is due to highly intensive computation involving large amounts of data transfer. Research efforts have thus turned towards special hardware paradigms for accelerating SHE, including Application-Specific Integrated Circuits (ASICs) [28, 40], Field-Programmable Gate Arrays (FPGAs) [29, 34, 35, 23], and GPUs [3]. Each of these paradigms has their own strengths and limitations, such as data transfer, memory availability, word size, and cost. While using FPGAs to accelerate SHE has been highly successful, FPGAs are ultimately limited by data transfer (both within the FPGA, and to/from other parts of the system), limits on the amount of on-chip memory, and limited word size. In much previous research, data transfer has been of great significance, with much effort in design and implementation devoted to mitigating the latency of data transfer. The root cause of this need is the inherent separation and bottlenecks between processing and memory present in traditional computing architectures as well as in specialized solutions such as ASICs, FPGAs, and GPUs. Computing paradigms such as Near-Memory Processing (NMP) and Compute-Enabled RAM (CE-RAM) seek to overcome this by reducing or removing the separation of processing and memory. This approach ameliorates the latency induced by data transfer. In addition, CE-RAM also enables a high degree of parallelism by allowing simultaneous operations on data within the same bank of CE-RAM without significant extra overhead. Existing work has shown the efficacy and potential of applying CE-RAM to modern ring-based SHE schemes [33]. Even without the common algorithmic optimizations such as the Number-Theoretic Transform (NTT) and Residue Number System (RNS), utilizing CE-RAM showed speedup as compared to both a CPU environment and related existing work which incorporated the algorithmic optimizations. This showed the viability of using CE-RAM as a hardware accelerator for SHE, and both actual and potential speedups over existing accelerators.

However, algorithmic optimization in the CE-RAM environment is severely underexplored. The efficacy of NTT and RNS has not been explored in CE-RAM, and the algorithmic study of the impact of CE-RAM's support of arbitrary word sizes is unprecedented. In this paper, we present the algorithmic optimization of the SHE implementations with CE-RAM, by applying NTT, RNS, and our novel optimization. We study the Full RNS Variant of Bajard et al. [6] that optimizes the B/FV homomorphic encryption scheme [18], implementing essential SHE operations. There exists another RNS variant of the B/FV scheme formulated by Halevi et al., based on floating-point operations [20], but we choose to study the integer-only variant of Bajard et al. because there is no computational disadvantage to one full-RNS variant over the other [7] and the integer-only vari-

ant of Bajard et al. is easier to implement in CE-RAM, as it does not require floating-point computations.

In this work, we realize the RNS and NTT optimizations for the B/FV scheme, along with new optimizations in the CE-RAM implementation. We performed extensive experiments with comparisons to a CPU implementation and a previous CE-RAM implementation. This is the first such work implementing these advanced algorithmic optimizations working within CE-RAM and its limitations. While a myriad of other work applying these optimizations exists [35, 34, 10], our work differs in that CE-RAM is only friendly to a small class of parameters. The results indicate speedups of up to 784.2x for homomorphic multiplication, compared against a state-of-the-art software library run on a server-grade computer. To the best of our knowledge, this is the best speedup in the literature. Our optimized CE-RAM implementation outperforms other hardware accelerators of B/FV with speedups in homomorphic multiplication of up to 506.5x against existing a CE-RAM implementation, 66.85x against an FPGA implementation, and 30.8x against a GPU implementation.

Summary of contributions: (1) For the first time, we apply RNS and NTT to the polynomial operations accelerated by CE-RAM and show that such algorithmic optimizations improve homomorphic multiplication by up to 506.5x as compared to an existing CE-RAM implementation. (2) We propose novel algorithmic optimization with new RNS base choices friendly to CE-RAM that allow quicker calculation of modular reduction and some modular multiplications in CE-RAM. This yields additional speedups of up to 1.49x for homomorphic multiplication and 2.89x for encryption/decryption. (3) We present how to continue to support the full-RNS designs and NTT under the new RNS base, overcoming the challenges presented by the limitations of current CE-RAM capabilities.

2 Preliminaries

2.1 Notations

For $x \in \mathbb{R}$, $\lfloor x \rceil, \lfloor x \rfloor$ indicate rounding to the nearest integer and rounding down respectively. Also, we use plain lowercase letters (e.g., x, y) to denote scalar values and bold lowercase letters (e.g., $\mathbf{x}, \mathbf{y}$) to denote polynomials. The division-with-rounding (DWR) operation is applied to polynomials as $DWR^c_{a,b}(\mathbf{x}) = (\lfloor \lfloor \frac{a}{b} \cdot x_i \rceil \rceil_c)$. We use R to denote a quotient ring of polynomials in the form $\mathbb{Z}[x]/\Phi_M(x)$, where $\Phi(x)$ is the M^{th} cyclotomic polynomial with M being a power of 2, i.e., its degree N which is $M/2$. R_t is the ring $\mathbb{Z}_t[x]/\Phi(x)$, with coefficients in the set $\mathbb{Z}_t = [\frac{-t}{2}, \frac{t}{2}) \cap \mathbb{Z}$. $|x|_t$ is the ordinary modular reduction defined as $|x|_t = x - \lfloor \frac{x}{t} \rfloor t$, and $\lceil x \rfloor_t$ is the centered modular reduction defined as $\lceil x \rfloor_t = x - \lfloor \frac{x}{t} \rceil t$. For $\mathbf{x} \in R$, we use $[\mathbf{x}]_t$ to denote an element in R_t which is obtained by applying the centered modular reduction to individual coefficients componentwise. Temporarily, a division may applied to $\mathbf{x} \in R$ which is performed to every coefficient componentwise. Such a polynomial in $\mathbb{Q}[x]/\Phi(x)$ is immediately mapped to R_t by applying the rounding function to every coefficient.

2.2 The Original B/FV Scheme [18]

Suppose we have $t > 1$ (plaintext modulus) and $q > t$ (ciphertext modulus). The B/FV scheme operates on plaintexts in R_t and ciphertexts in R_q^2. The secret key $\mathbf{s}$ is a randomly chosen element of R with coefficients from a distribution bounded in magnitude by 1 (denoted as 1-bounded distribution). The public key is $(\mathbf{p_0}, \mathbf{p_1}) = ([\mathbf{as} + \mathbf{e}]_q, \mathbf{a})$, where $\mathbf{a}$ is chosen uniformly at random from R_q and $\mathbf{e}$ is also chosen from a 1-bounded distribution. Denote $\Delta = \lfloor \frac{q}{t} \rfloor$.

Encryption: To encrypt $\mathbf{m} \in R_t$, first randomly sample $\mathbf{e_1}, \mathbf{e_2}$ from a 1-bounded distribution and $\mathbf{u}$ uniformly from R_q. Then compute $(\mathbf{c_0}, \mathbf{c_1}) = ([\Delta\mathbf{m} + \mathbf{p_0u} + \mathbf{e_1}]_q, [\mathbf{p_1u} + \mathbf{e_2}]_q)$.

Homomorphic Addition/Subtraction: Given ciphertexts $(\mathbf{c_0}, \mathbf{c_1})$, $(\mathbf{c'_0}, \mathbf{c'_1})$, homomorphic addition is calculated by computing $(\mathbf{c_0}^+, \mathbf{c_1}^+) = ([\mathbf{c_0} + \mathbf{c'_0}]_q, [\mathbf{c_1} + \mathbf{c'_1}]_q)$. Subtraction proceeds similarly.

Decryption: To decrypt $(\mathbf{c_0}, \mathbf{c_1})$ under a key $\mathbf{s}$, compute $\mathbf{m} = [\lfloor \frac{t}{q} [\mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s}]_q \rceil]_t$.

Homomorphic Multiplication: Given ciphertexts $\mathbf{c} = (\mathbf{c_0}, \mathbf{c_1})$, $\mathbf{c'} = (\mathbf{c'_0}, \mathbf{c'_1})$, homomorphic multiplication is computed by first finding $\mathbf{d_0} = [\lfloor \frac{t \cdot \mathbf{c_0} \cdot \mathbf{c'_0}}{q} \rceil]_q$, $\mathbf{d_1} = [\lfloor \frac{t \cdot \mathbf{c_0} \cdot \mathbf{c'_1} + \mathbf{c_1} \cdot \mathbf{c'_0}}{q} \rceil]_q$, $\mathbf{d_2} = [\lfloor \frac{t \cdot \mathbf{c_1} \cdot \mathbf{c'_1}}{q} \rceil]_q$. Then using precomputed relinearization keys $\mathbf{rlk_0}, \mathbf{rlk_1}$ (tuples of T pairs of elements of R_q), write $\mathbf{d_2}$ in base T with $\mathbf{d_2}^{(i)}$ as its i^{th} digit, and return $(\mathbf{d'_0} = [\mathbf{d_0} + \sum_i \mathbf{rlk}[i][0]\mathbf{d_2}^{(i)}]_q, \mathbf{d'_1} = [\mathbf{d_1} + \sum_i \mathbf{rlk}[i][1]\mathbf{d_2}^{(i)}]_q)$

2.3 The Full-RNS Variant of the B/FV Scheme

RNS: In the B/FV scheme, operations take place on polynomials with large coefficients in $\mathbb{Z}_q$. By the Chinese Remainder Theorem (CRT), if q is a product of k pairwise coprime numbers q_i, $\mathbb{Z}_q$ is isomorphic to the product of the rings $\{\mathbb{Z}_{q_i}\}_i$. A number in $\mathbb{Z}_q$ can then be represented by a k-tuple of numbers in $\{\mathbb{Z}_{q_i}\}_i$, which is denoted as *RNS form*. The i^{th} component of an integer x's RNS form is $|x|_{q_i}$ (or $[x]_{q_i}$ in a centered representation). Because of the isomorphism addition and multiplication on numbers in $\mathbb{Z}_q$ can be performed by performing addition/multiplication componentwise on the RNS form. If each modulus q_i is small enough to fit into a computer word (e.g., 64 bits), then individual operations in an the RNS form become single-precision operations. Also, operations on numbers in RNS form can be parallelized, as each component is independent.

Full-RNS Variant: A Full-RNS variant of an SHE scheme refers to a variant where all operations are performed on the RNS form extended to the polynomial rings throughout the entire homomorphic evaluation. Namely, parallel polynomial operations in smaller subrings $\{R_{q_i}\}_i$ are performed instead of an operation in the large ring R_q without reconstructing the polynomials in R_q. This leads to enhanced performance [6, 20]. Unlike algorithms only involving addition, subtraction, and multiplication (e.g., homomorphic addition), decryption and homomorphic multiplication require operations not easily performed in RNS form (e.g. division and rounding). The full-RNS variant thus focuses on these algorithms. Let $Q = \{q_0, q_1, \cdots, q_k\}$ and $B = \{b_0, b_1, \cdots, b_\ell\}$ be sets of numbers

Algorithm 1 Full RNS B/FV Decryption

procedure $DEC_{RNS}((\mathbf{c_0}, \mathbf{c_1}), \mathbf{s})$
  $\mathbf{s}^{(t)} \leftarrow \lfloor - FastBConv(\lfloor \gamma t \mathbf{x} \rceil_q, Q, \{t\}) \cdot \lfloor q^{-1} \rceil_t \rceil_t$,
  $\mathbf{s}^{(\gamma)} \leftarrow \lfloor - FastBConv(\lfloor \gamma t \mathbf{x} \rceil_q, Q, \{\gamma\}) \cdot \lfloor q^{-1} \rceil_\gamma \rceil_\gamma$
  return $[(\mathbf{s}^{(t)} - [\mathbf{s}^{(\gamma)}]_\gamma) \cdot \lfloor \gamma^{-1} \rceil_t]_t$.
end procedure

Algorithm 2 Full RNS B/FV Homomorphic Multiplication

procedure $MULT_{RNS}(\mathbf{c} = (\mathbf{c_0}, \mathbf{c_1}), \mathbf{c}' = (\mathbf{c'_0}, \mathbf{c'_1}))$
  Use $FastBConv$ to convert $\mathbf{c}, \mathbf{c}'$ from Q to $B_{sk} \cup \{\tilde{m}\}$. (We now have $\mathbf{c}, \mathbf{c}'$ in $Q \cup B_{sk} \cup \{\tilde{m}\}$.)
  Reduce extra multiples of q in the $B_{sk} \cup \tilde{m}$-representation using Small Montgomery Reduction [8]. (The intermediate results are now in $Q \cup B_{sk}$.)
  Compute the polynomial products $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}}) = (\mathbf{c_0} \cdot \mathbf{c'_0}, \mathbf{c_0} \cdot \mathbf{c'_1} + \mathbf{c_1} \cdot \mathbf{c'_0}, \mathbf{c_1} \cdot \mathbf{c'_1})$ and scale by t (in $Q \cup B_{sk}$).
  Do a fast floor (using $FastBConv$) from $Q \cup B_{sk}$ to B_{sk}. (This is an approximation of the DWR operation.)
  Perform a Shenoy and Kumaresan-like reduction from B_{sk} to Q [36].
  return $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}})$ (in base Q)
end procedure

each relatively coprime with every other element of $Q \cup B$, called *RNS bases*. Let q be the product of the elements of Q. *Fast Base Conversion*, or $FastBConv$, is defined by

$$FastBConv(x, Q, B) = \Big\{ \sum_{i=1}^{k} \lfloor x_i \cdot \frac{q_i}{q} \rceil_{q_i} \cdot \frac{q}{q_i} \bmod b_i \Big| b_i \in B \Big\} \tag{1}$$

The $FastBConv$ quickly and approximately converts a number x in RNS form with base Q to one with base B, and it is applied coefficientwise to polynomials. This is used for the operations other than additions and multiplications. It is faster than full CRT reconstruction mainly because the intermediate modular reduction by Q is skipped. As a consequence, the result in base B will be off by some multiple of q, which can be corrected with various efficient methods.

Decryption: The full-RNS variant's decryption is described in Algorithm 1, using a number γ coprime to t (though this is not strictly necessary).

Homomorphic Multiplication: The full-RNS variant's homomorphic multiplication (sans relinearization) is described in Algorithm 2, which gives us $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}})$ in base Q. In this algorithm, we temporarily extend ciphertexts using $FastBConv$ from base Q to $B_{sk} = B \cup \{m_{sk}\}$, to hold the result of a polynomial tensor where coefficients may be as large as nq^2. The next step is relinearization that reduces this ciphertext back to 2 elements. Suppose we are given $(\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}}, \tilde{\mathbf{c_2}})$ as from Algorithm 2. Let $rlk_{RNS}[0], rlk_{RNS}[1]$ be the relinearization keys (precomputed tuples of polynomials in R_q). Define $D_{RNS}(\tilde{\mathbf{c_2}}) = (\lfloor \tilde{\mathbf{c_2}} \frac{q_1}{q} \rceil_{q_1}, \lfloor \tilde{\mathbf{c_2}} \frac{q_2}{q} \rceil_{q_2}, \cdots, \lfloor \tilde{\mathbf{c_2}} \frac{q_k}{q} \rceil_{q_k})$. We then perform relinearization by computing: $([\tilde{\mathbf{c_0}} + \langle D_{RNS}(\tilde{\mathbf{c_2}}), rlk_{RNS}[0] \rangle]_q, [\tilde{\mathbf{c_1}} + \langle D_{RNS}(\tilde{\mathbf{c_2}}), rlk_{RNS}[1] \rangle]_q)$.

This full-RNS variant of Bajard et al. [6] shows a large practical speedup in runtime (up to 20 times faster for decryption, and 4 times faster for multiplication), due in part to faster individual operations on RNS components.

2.4 NTT

The convolution of a polynomial multiplication is a $\mathcal{O}(N^2)$ algorithm. Applying the NTT (Appendix C) to polynomials results in transformed polynomials, where coefficientwise multiplication in the new domain corresponds to polynomial multiplication in the original domain. Thus polynomial multiplication can be performed in $\mathcal{O}(N \cdot log(N))$ by transforming polynomials, performing coefficientwise multiplication, and applying the inverse NTT is commonly used to expedite polynomial multiplications [10, 35, 29].

2.5 Compute-Enabled RAM

Compute Enabled RAM (CE-RAM) refers to SRAM storage units with integrated processing elements. The most notable features of CE-RAM are the lack of separation between memory and processing and the massive potential for parallel processing that this allows. In this work, we consider CE-RAM as a SRAM bank at the level of L3 cache. CE-RAM can then be used as coprocessing hardware, performing intensive computations in the CPU's stead, and transferring its data to/from DRAM through the standard memory hierarchy. In the previous work of Reis et al. [33], CE-RAM (referred to as CiM in their paper, but we choose the term CE-RAM to avoid confusion with the analog In-memory Computing) has been utilized as a coprocessing unit, and used to accelerate homomorphic operations of the B/FV scheme. This approach, without algorithmic optimizations of RNS and NTT, is reported to have a significant speedup. Against a CPU implementation with RNS and NTT, a speedup of up to 5.8x for homomorphic multiplication is reported. This work was a first pioneering effort in applying CE-RAM to SHE, and faced several limitations, including a limited choice of ciphertext modulus (only powers of two) and a lack of algorithmic optimizations. Unfortunately, due to the lack of algorithmic optimization, the speedup drops to 1.5x when $N = 2^{14}$ and the ciphertext modulus size $|q|$ is 438 bits [33], which is the largest parameter set among the existing work that we compare against.

3 Novel Optimizations Using Special Moduli

In this section, we discuss choices of RNS bases that are well suited for implementing the Full RNS Variant of the B/FV scheme in CE-RAM [18, 6].

3.1 NTT Implementation with CE-RAM

In this work, the NTT algorithm is mapped into a sequence of micro-operations (micro-ops) that can be executed with the CE-RAM hardware described in [33]. The NTT algorithm, shown in Algorithm 7 in Appendix C, takes two inputs: **(i)** the polynomial coefficients in bit-reverse order, and **(ii)** the constant Y', which relates to pre-computed twiddle factors of NTT. The polynomial coefficients are

initially stored in CE-RAM arrays in a natural manner, therefore the first step for CE-RAM execution of NTT is to implement the bit-reverse (or permutation) step. A sequence of N **MOVE** micro-ops (enabled by the CE-RAM's in-place move buffers, and coordinated by CE-RAM's controller) is used to permute the positions of the coefficients stored in memory. Once the coefficients are in bit-reverse order, execution of the procedure $NTT_p(a)$ can be initiated.

The NTT procedure consists of 3 nested loops. The outermost loop is executed $log(N)$ times with CE-RAM, and it corresponds to NTT stages (i.e., stage $1, 2, ..., log(N)$). Each NTT stage has a number of groups in it, whose execution is controlled by the second (butterfly) loop in Algorithm 7. Inside the butterfly loop, pre-computed twiddle factors are stored in the CE-RAM arrays with parallel **WRITE** micro-ops. Finally, the NTT core consists of modular multiplication between coefficients and the twiddle constants, followed by a subtraction and an addition. In CE-RAM, operations in the NTT core map to **MODMULT**, **SUB**, and **ADD** micro-ops. N micro-ops of each type should be performed (one for each coefficient of the polynomial). Note that CE-RAM can merge the "Butterfly" and "NTT Core" loops, and parallelize the execution of the N micro-ops of each category with the N sets of customized peripherals in a CE-RAM bank.

3.2 Choosing Special Moduli for Optimization

Note that the precise form of the ciphertext modulus q (e.g., whether q is prime, a product of coprime numbers, or a prime power) does not impact the security of the scheme because the RLWE problem is difficult for arbitrary q [18, 6, 24]. Even before the advent of ring-based SHE, extensive research has been done in the domain of hardware-based RNS systems in searching for moduli that allow for efficient modular reduction and computations [39]. In this work, we present novel optimizations by leveraging existing studies on special moduli, which are specialized for the word sizes of CE-RAM that can be larger than common fixed-size word sizes (e.g., 64 bits). We first find moduli that allow algorithmic optimizations that are especially useful for CE-RAM. In CE-RAM, multiplication is not a basic operation; it is constructed through shifting and addition [33]. We thus consider optimizations that obviate multiplications and replace them with simpler operations. Our strategy is to 1) choose moduli that allow efficient modular reductions, which can accelerate modular polynomial arithmetic as well as the NTT, and 2) choose moduli that allow additional SHE-specific algorithmic optimizations. W investigate two moduli sets for these goals.

Balanced 3-moduli Set: The first set of moduli we consider is the well-studied set $S_1 = \{q_0, q_1, q_2\} = \{2^g - 1, 2^g, 2^g + 1\}$ [39]. This set has the advantage of being *balanced*, i.e., all elements are equal-sized, and admits easy modular reductions (shown in Section 3.3), which can be simplified further within the specific application of SHE. The main disadvantage is that the set is limited to 3 elements, limiting its range and scalability. The set is also limited to representing numbers of approximately $3g$ bits, however with CE-RAM we can choose g to be larger than 64 bits, making this less of a concern. This is mitigated in the CE-RAM because there is no limit in the word size. Further, these moduli produce

Algorithm 3 Fast Modular Reduction - Mersenne (optimized)

1: **procedure** $FMR_{-1}(x, 2^g - 1)$ ▷ Returns $x \mod 2^g - 1$ for $x \in [0, (2^g - 1)^2)$
2: $y \leftarrow x >> g$
3: $x \leftarrow x\&(2^g - 1)$
4: $x \leftarrow x + y$
5: **if** $x \geq 2^g - 1$ **then**
6: $x \leftarrow x - (2^g - 1)$
7: **end if**
8: **return** x
9: **end procedure**

Algorithm 4 Fast Modular Reduction - Fermat-like (optimized)

1: **procedure** $FMR_{+1}(x, 2^g + 1)$ ▷ Returns $x \mod 2^g + 1$ for $x \in [0, (2^g + 1)^2)$
2: $y \leftarrow x >> g$
3: $x \leftarrow x\&(2^g - 1)$
4: **if** $x \geq y$ **then**
5: $x \leftarrow x - y$
6: **else**
7: $x \leftarrow ((2^g + 1) - y) + x$
8: **end if**
9: **return** x
10: **end procedure**

forms of $|\frac{q_i}{q}|_{q_i}$ that admit extremely efficient computation of the $x \cdot |\frac{q_i}{q}|_{q_i}$, allowing optimization of $FastBConv$ as shown in Section 3.4.

Mersenne/Fermat-like Coprimes: Sets of the form $S_2 = \{2^m - 1, 2^m + 1, 2^{2m} + 1, \cdots 2^{2^{f-1}m} + 1, 2^{2^f m} + 1\}$ form a set of coprime moduli [31]. This set of moduli has certain advantages: first, its inverses $|\frac{q_i}{q}|_{q_i}$ are powers of two, allowing the multiplications in $FastBConv$ to become bitshifts. Second, all moduli in the set are one separated from a power of two, allowing efficient modular reduction by Algorithms 3 and 4. These combined show that modular multiplication by $|\frac{q_i}{q}|_{q_i}$ in $FastBConv$ can be done efficiently with circular bitshifting (Ch. 9, Thm. 9.2.12 [15]). We include relevant lemmas and proofs in the appendix. However, this set is unbalanced, which leads to relative performance disadvantages as shown in the experiments.

3.3 Optimizing Modular Reduction

The moduli in both S_1 and S_2 are powers of two, or unit distance from powers of two. Modular reduction with these moduli can be performed efficiently. In the case of q_1 of S_1, the modulus is a power of two, and reduction is a simple mask. In the other cases, we can extend well-known formulae for efficient modular reduction [15]. In the context of Full-RNS variants of SHE, we perform modular arithmetic by reducing after each operation, and arguments to be reduced are less than the square of the modulus. This obviates the need for iteration or multiplication in modular reduction, so that only simple bit operations and linear-complexity arithmetic are required. This leads us to the optimized Algorithms 3 and 4, which can be used for the moduli in S_1 and S_2. These are applied in all operations in RNS base Q (including NTT).

Algorithm 5 Fast Modular Shifting - Mersenne

procedure $FMS_{-1}(x, y, 2^g - 1)$ ▷ Returns $x \cdot 2^y \mod 2^g - 1$
  $hi \leftarrow x >> (g - y)$
  $lo \leftarrow x\&((1 << y) - 1)$
  if $lo \geq hi$ **then**
    return $lo - hi$
  else
    return $((1 << g) - 1) - hi + lo$
  end if
end procedure

Algorithm 6 Fast Modular Shifting - Fermat-like

procedure $FMS_{+1}(x, y, 2^g + 1)$ ▷ Returns $x \cdot 2^y \mod 2^g + 1$
  $hi \leftarrow x >> (g - y)$
  $lo \leftarrow x\&((1 << y) - 1)$
  return $lo|hi$
end procedure

3.4 Optimizing *FastBConv*

In the frequently used operation $FastBConv$, terms of the form $|x \cdot \frac{q_i}{q}|_{q_i}$ are computed. When using the moduli of S_1, computing this can be done extremely efficiently, with only simple bit-wise operations as shown in Theorems 1 and 2 and Lemma 2 (proofs of nontrivial lemmas and theorems are in the appendix.).

Lemma 1. *For the moduli of* S_1, $|\frac{q_0}{q}|_{q_0} = 2^{g-1}$, $|\frac{q_1}{q}|_{q_1} = 2^g - 1$, *and* $|\frac{q_2}{q}|_{q_2} = 2^{g-1} + 1$.

Theorem 1. *For the moduli of* S_1 *with* $x \in [0, q_0)$, $|x \cdot \frac{q_0}{q}|_{q_0}$ *is* $\frac{x}{2}$ *when* x *is even and* $2^g - 1 + \frac{x-1}{2}$ *when* x *is odd.*

$|x \cdot \frac{q_0}{q}|_{q_0}$ can be computed efficiently as `(x>>1) + ((x&1) << (g-1))`.

Lemma 2. *For the moduli of* S_1 *with* $x \in [0, q_1)$, $|x \cdot \frac{q_1}{q}|_{q_1} \equiv -x \mod 2^g$.

$|x \cdot \frac{q_1}{q}|_{q_1}$ can be computed efficiently as `((1 << g) - x) & ((1 << g) - 1)`.

Theorem 2. *For the moduli of* S_1 *with* $x \in [0, q_2)$, $|x \cdot \frac{q_2}{q}|_{q_2}$ *is* $\frac{x}{2}$ *when* x *is even and* $2^{g-1} + \frac{x+1}{2}$ *when* x *is odd.*

$|x \cdot \frac{q_2}{q}|_{q_2}$ can be computed efficiently as `((x+1)>>1) + ((x&1) << (g-1))`.

For the moduli of S_2, the terms $|\frac{q_i}{q}|_{q_i}$ are powers of two, turning the multiplication into shifting, and the modular reductions can again use Algorithms 3 and 4. Alternately, and more efficiently, we can utilize circular bitshifting to perform multiplication and reduction in a single operation, as shown in Algorithms 5 and 6. These optimizations are most useful in systems such as CE-RAM where it is most advantageous to replace multiplications with simple bitwise operations.

3.5 Extended Base

The optimizations presented in Sections 3.3 and 3.4 are only applicable to operations within base Q. Using more moduli, as when operating in the extended

base B (as described in Section 2.3), presents a challenge - coprime moduli close to a power of two offset by at most one (allowing efficient modular reduction) are already exhausted. We thus choose prime moduli $b_i = 2^g + c$ with $|c| < g$, where $2^g + 1$ is the largest modulus in whichever of S_1 or S_2 being used for Q. This is done so that the largest element of Q and the elements of B are of the same size (number of bits), so that the complexity of multiplication does not increase beyond what is already determined by Q. We further require $b_i \equiv 1 \mod 2N$, to allow for efficient computation of the parameters needed for the NTT (as discussed in Section 3.6). No special optimizations such as in Sections 3.3 and 3.4 are available, so Barrett reduction [9] is utilized for modular reduction for components outside of base Q. Barrett reduction is commonly utilized in other similar research [34, 10]. The moduli $m_{sk}, \tilde{m}$ can be chosen as described in [6].

3.6 Finding NTT Parameters

We first discuss applying the NTT for moduli of the form $2^g+1, 2^g-1$. NTT with moduli of $2^g + 1$ are known in the literature [1] and are called Fermat Number Transforms (FNT). The twiddle factors will be chosen as 2 or the quadratic residue of 2, in which case they will be the primitive $2g^{th}$ (resp. $4g^{th}$) root of unity. Then when $M \mid 2g$ (resp. $4g$), $2^{2g/M}$ is the M^{th} root of unity. Similarly, the NTT for $2^g - 1$ (called Mersenne Number Transform MNT) is described in [32], with twiddle factors being 2. In the scenario of RLWE-based homomorphic encryption, $M = 2N$. For moduli of the form 2^g, we are considering a finite field $GF(2)$. NTT twiddle factors over finite fields are largely known. Since $\varphi(2^g) = 2^{g-1}$, we find the primitive root of unity and raise it to $2^{g-1}/M$ to find the M^{th} root of unity.

As noted in Section 3.5, we can choose moduli in the extended base B to be NTT-friendly; with $b_i \equiv 1 \mod 2N$ a $2N^{th}$ primitive root of unity is easily found, which allows efficient computation of NTT parameters (namely the twiddle factors Y in Algorithm 7) [34, 26]. The moduli $\tilde{m}$ and m_sk can be chosen similarly.

4 Experimental Evaluation

Our optimizations do not change the overall asymptotic complexity of the RNS variant, for the same reason that the variant does not improve the asymptotic complexity [6]. However, they greatly improve the practical efficiency of the CE-RAM implementations, which is shown with our extensive experimental evaluation. We implement primitive operations (modular polynomial arithmetic, NTT, coefficientwise multiplication, and our optimized procedures) in CE-RAM and use those measurements to derive the runtime of relevant homomorphic operations: homomorphic multiplication, addition, and subtraction, along with encryption and decryption. These are compared to a software implementation of B/FV, as well as different research works in hardware acceleration of SHE. Relinearization is considered to be a part of multiplication (i.e., each relinearization

Table 1. Parameter Cases

Case	Moduli	Using Our Novel Optimization?
Case A	$\{2^{146}-1, 2^{146}, 2^{146}+1\}$	No
Case B	$\{2^{146}-1, 2^{146}, 2^{146}+1\}$	Yes
Case C	$\{2^{220}+1, 2^{110}+1, 2^{110}-1\}$	Yes

is run on a ciphertext with two components, and each multiplication has relinearization as a subroutine). As homomorphic multiplication is the most intensive operation, it is the operation we refer to most when evaluating our work and comparing it to other research.

4.1 CE-RAM Environment and Parameters

We chose a polynomial modulus degree of $N = 2^{14}$ and ciphertext modulus size $|q|$ of at least 438. This choice of parameters provides 128 bits of security [5], matching or exceeding the security afforded by parameters chosen by Roy et al [35], Reis et al. [33], Al Badawi et al. [3], and HEAX [34]. These parameters also yield a scheme with multiplicative depth of at least 10. We choose the sets $S_{balanced} = \{2^{146}-1, 2^{146}, 2^{146}+1\}$ and $S_{unbalanced} = \{2^{220}+1, 2^{110}+1, 2^{110}-1\}$ (i.e., Mersenne/Fermat-like coprimes), and use these sets as the modulus set q. Additional coprime moduli (e.g. the elements of the extended base B) are chosen to be numbers slightly larger than the largest modulus in a set, as discussed in Section 3.5. We consider cases A and B to be using $S_{balanced}$ and case C to use $S_{unbalanced}$, with cases B and C additionally utilizing our novel optimizations presented in Section 3 (Table 1).

CE-RAM is evaluated following the same methodology reported by Reis et al. [33], i.e., with circuit-level simulations of a CE-RAM bank based on multiple 1KB CE Static Random-Access Memory Cells (SRAM) arrays. The circuit simulation tools employed are Cadence Encounter (for the CE-RAM controller circuits) [17], and Synopsys HSPICE (for the RAM array and compute-enabled peripherals) [37]. Circuits are based on the 14nm BSIM-CMG FinFET model [16]. Each CE-RAM array has 8 rows and 1024 columns, consisting of SRAM cells and customized memory peripherals. In our evaluation, we employ a 16 MB CE-RAM, which is built of 16,384 arrays of 8×1024 size (a tiled architecture). CE-RAM performs its operations at the bitline level, which means that the coefficients of 2 polynomials need to be column-aligned for computing in-memory polynomial operations. This condition is ensured by an appropriate mapping of the polynomials to the CE-RAM arrays, i.e., polynomials are always entirely mapped to the same row index across the 16,384 arrays. Through this mapping, two polynomial primitives are performed simultaneously (at the same clock cycle). Because this mapping holds up to 512 bits for coefficients in each row, we can use the spare space to hold coefficients modulo m_{sk} and $\tilde{m}$, avoiding a need for an extra row during calculations with operands in the base $Q \cup B_{sk} \cup \tilde{m}$.

Using circuit simulations, we measure the time for executing each CE-RAM instruction. We then proceed with the mapping of each homomorphic operation into a sequence of CE-RAM instructions. Based on the time measured for each

Table 2. Speedup of Full-RNS CE-RAM (wo/ NTT) vs. Existing CE-RAM (wo/ RNS,NTT) [33]

Case	Hom. Mult.
Case A	1.28×
Case B	**1.90x**
Case C	1.12×

instruction and the mapping of the homomorphic operation to the instructions, we compute the overall time of homomorphic operations in CE-RAM.

We also consider the impact of data transfer in reporting our results, and when comparing to other work, as appropriate. The DDR4 specification gives a peak data transfer rate of 25,600 MB/s, which gives $70\mu s$ seconds per ciphertext transfer from DRAM to CE-RAM (referred to hereafter as the *specified data transfer rate*) In practice, the overhead from data transfer can be significantly less, due to pipelining and interleaving of execution and data transfer. Based on experiments run in our CPU environments, we estimate $21.2\mu s$ seconds per ciphertext (referred to hereafter as the *actual data transfer rate*) transfer from DRAM to CE-RAM. This figure was derived by observing the time difference between operating upon ciphertexts that were/were not resident in cache memory.

4.2 Comparison to CE-RAM Implementation of B/FV

We define the speedup of case X against case Y as $\frac{\text{Run time in case } Y}{\text{Run time in case } X}$.

Full-RNS Variant: To examine the speedup from applying the full-RNS design, we compare our current work with that of Reis et al. [33], as that work used CE-RAM (referred to as CiM) for B/FV but implemented the textbook scheme without the use of any RNS variant. The results for homomorphic multiplication are shown in Table 2. The speedup was found using the actual data transfer time, though the speedup was nearly identical to two decimal places when the specified data transfer time was used. We note that in [33], the ciphertext modulus was chosen to be a power of two, enabling extremely efficient modular reduction with a simple bitmask. Considering this, we conclude that applying RNS alone does not bring much improvement, due to the efficient CE-RAM parameter choices of previous work and the variable word size of CE-RAM.

NTT: To examine the speedup we get from applying NTT to CE-RAM, we evaluated variants of encryption, decryption and homomorphic multiplication with and without NTT-based polynomial multiplication. These results are shown in Table 3. From this, we see that applying NTT brings a speedup of two orders of magnitude for homomorphic multiplication. Of the optimizations we bring to CE-RAM, NTT is the most effective; this is expected, as the runtime of decryption and homomorphic multiplication in the RNS variant is dominated by polynomial multiplication, which NTT optimizes.

Novel Optimizations: To find the efficacy of our novel optimizations, we examine the speedup for Cases B and C against Case A (Table 4). We note a modest speedup for homomorphic multiplication with Case B, and a larger speedup for other operations. Data transfer dominates the latency in homomorphic ad-

Table 3. Speedup of CE-RAM with NTT v.s. CE-RAM without NTT (w/ Full-RNS in both cases)

Case	Operation	Hom. Mult.	Decrypt	Encrypt
Case A	Speedup wo/ transfer	396.90x	236.37x	1525.53x
	Imp. w/ actual transfer	292.56x	127.91x	554.11x
	Imp. w/ spec. transfer	182.22x	82.69x	224.86x
Case B	Speedup w/o transfer	412.59x	315.47x	1085.19x
	Imp. w/ actual transfer	265.90x	73.75x	162.63x
	Imp. w/ spec. transfer	146.28x	38.89x	55.60x
Case C	Speedup wo/ transfer	449.68x	338.39x	1100.51x
	Imp. w/ actual transfer	332.32x	109.45x	249.14x
	Imp. w/ spec. transfer	207.53x	60.77x	90.09x

Table 4. Speedup of CE-RAM with Novel Optimizations v.s. CE-RAM without Novel Optimizations (not including NTT)

Case	Hom. Mult.	Decrypt	Encrypt	Hom. Add.	Hom. Sub.
Case B, no data trans.	1.49x	2.91x	4.56x	37.08x	35.15x
Case B, actual data trans.	1.49x	2.89x	4.55x	1.01x	1.01
Case B, spec. data trans.	1.49x	2.86x	4.52x	1.00x	1.00
Case C, no data trans.	0.87x	1.72x	2.71x	33.94x	32.32x
Case C, actual data trans.	0.87x	1.72x	2.70x	1.01x	1.01x
Case C, spec. data trans.	0.87x	1.71x	2.69x	1.00x	1.00x

* Overhead of Hom. Mult. dominates the overhead of Hom. Add./Sub. by two orders of magnitude.

dition/subtraction, so the speedup is only notable without data transfer. We conclude the Case B with optimized balanced set is the most effective.

4.3 Comparison to CPU Implementation of B/FV

The software library used for our CPU experiments is the highly optimized Microsoft SEAL homomorphic encryption library [10]. SEAL applies the full-RNS variant and NTT to B/FV. Our tests on CPU are written in C++.

For rigorous testing against a wide range of ordinary CPU environments, we evaluate our CE-RAM approach against three different computers (Table 5). We chose these computers to represent a server used in cloud computing (CPU-Server, used for comparison with other research), a research workstation (CPU-Workstation), and an ordinary user's computer (CPU-Laptop). Our CPU experiments use the standard parameters of $N = 2^{14}$, $|q| = 438$, but with moduli limited to computer-supported word size (64 bits). Table 6 shows the speedup that CE-RAM enjoys compared to our CPU environments, using our actually measured data transfer time. Most notably, for homomorphic multiplication, we see speedups of two orders of magnitude. Table 7 shows the number of operations per second that can be computed with CE-RAM using different estimates for data transfer. These are more useful for comparison with related research.

In all three cases we tested, a significant speedup over the CPU environment is seen. For homomorphic multiplication, we see a speedup of two orders of magnitude (up to **784x** faster against CPU-Server). The use of algorithmic optimizations (Case B) shows additional speedup as compared to Case A (no additional optimizations). This is much more pronounced in encryption and decryption, though homomorphic multiplication/addition/subtraction still see some speedup from these optimizations. Encryption and decryption show a more

Table 5. CPU Environment Specifications

Data	CPU-Server	CPU-Workstation	CPU-Laptop
CPU	AMD EPYC 7451, 2.3 GHz	Intel i7, 3.4 GHz	Intel i5, 2.30GHz
Level 1 Cache Size	64KB(i)/32KB(d)	32KB(i)/32 KB(d)	32KB(i)/32 KB(d)
Level 2 Cache Size	512KB	256KB	256KB
Level 3 Cache Size	8MB	8MB	3MB
Memory Size	128GB	8GB	8GB
OS	Red Hat	Red Hat	Ubuntu 18.04 LTS
Sole user?	No	Yes	Yes

Table 6. Speedup of CE-RAM vs. CPU cases (actual data trans.)

Environment	Case	Hom. Mult.	Hom. Add	Hom. Sub.	Decrypt	Encrypt
CPU-Server	Case *A*	579.9x	5.1x	5.2x	75.8x	225.0x
	Case *B*	**784.2x**	5.1x	5.3x	143.9x	330.3x
	Case *C*	575.6x	5.1x	5.3x	122.1x	290.2x
CPU-Laptop	Case *A*	577.5x	5.7x	5.9xx	78.1x	216.3x
	Case *B*	**781.0x**	5.8x	5.9x	148.3x	317.5x
	Case *C*	573.2x	5.8x	5.9x	125.9x	279.0x
CPU-Workstation	Case *A*	406.8x	3.6x	3.8x	52.3x	151.5x
	Case *B*	**550.1x**	3.6x	3.9x	99.2x	222.4x
	Case *C*	403.7x	3.6x	3.9x	84.2x	195.4x

modest speedup, though they still improve upon the CPU environments by two orders of magnitude.

The unbalanced moduli set (Case C) does bring some speedups over a balanced set without optimizations (Case A) - for every operation besides homomorphic multiplication, computation time with the unbalanced set improved upon the balanced set without optimizations. However, for homomorphic multiplication the unbalanced set was slower than the balanced set, regardless of whether optimizations were enabled. This is likely due to multiplication's complexity, which increases quadratically with the number of bits in the operands, so that the runtime of parallel operations is now dominated by the size of the largest RNS component. We conclude that despite its interesting mathematical properties, the unbalanced set is less useful for implementing RNS variants in CE-RAM.

4.4 Considering Throughput with Projection

To consider the effects of more parallelism and interleaving and examine possible throughput, assuming two 16MB banks of CE-RAM allows maximum utilization of data transfer between DRAM and SRAM without failing to write back results as soon as they are available (note that the actual data transfer time is about one-eighth the time to compute a B/FV homomorphic multiplication in Case *B*). Now considering total end-to-end throughput, this approximately doubles the homomorphic multiplications per second CE-RAM can achieve to 11222 in Case B with actual data transfer. While further increasing the number of SRAM banks could further improve throughput, this would require increasing the total SRAM size beyond 32MB, as well as more sophisticated logic for handling the pipelining. One can also implement the architecture of CE-RAM at the main memory (i.e., DRAM).

4.5 Comparison to Other Hardware Accelerators of B/FV

CE-RAM: Beyond simply improving on a CPU environment, our work also shows speedups over existing work applying CE-RAM to the B/FV scheme [33],

Table 7. Operations per second with CE-RAM

Data Transfer	Case	Hom. Mult.	Hom. Add	Hom. Sub.	Decrypt	Encrypt
Actual	Case A	4150	15568	15567	8558	12734
	Case B	**5611**	15740	15740	16248	18694
	Case C	4118	15739	15739	13789	16426
Specified	Case A	2579	4727	4740	4658	5671
	Case B	**3078**	4756	4756	6275	6609
	Case C	2567	4756	4756	5871	6301
Computation Only	Case A	5635	1383126	1381215	13421	27636
	Case B	**8719**	51282051	48543689	52074	89685
	Case C	5577	46948357	44642857	33138	53946

Table 8. Speedup of Hom. Mult., Our vs. Existing [33] CE-RAM

Case	spec. data trans.	actual data trans.
Case A	232.8x	374.6x
Case B	**277.8x**	**506.5x**
Case C	231.7x	371.8x

as shown in Table 8. With Case B, we see a speedup of up to 506.5x for homomorphic multiplication as compared to the existing CE-RAM implementation. As noted in Section 4.2, we only analyze the homomorphic multiplication because it dominates additive operations.

FPGA: The work of Roy et al. [35] constructs an FPGA coprocessor for the B/FV scheme. The closest set of parameters used by Roy et al. was $N = 2^{13}$, $|q| = 360$. We compare our CE-RAM runtime for Case B against Roy et al.'s estimated runtime for homomorphic multiplication in Table 9, and speedups of an order of magnitude (66.85x using the actual data transfer rate) are observed. We note that these are even more impressive when noting that our evaluation uses larger parameters, providing a higher security level.

GPU: Al Badawi et al.'s GPU acceleration of B/FV [2] demonstrates the efficacy of applying the parallelism of GPUs to RNS variants of B/FV. Of the parameters they use, the closest ones to ours are $N = 2^{13}$, $|q| = 360$. All three parameter/optimization cases we tested showed a speedup against the GPU-based acceleration (Table 10), even though our work used a larger and more demanding set of parameters, with a higher level of security. For homomorphic multiplication, we see speedups of up to 30.8x with Case B.

5 Related Work

Other works in hardware acceleration of homomorphic encryption consider their hardware as part of a cloud computing system, where clients can outsource encrypted computations to a server that will utilize the hardware to more efficiently carry out the client's desired calculations [34, 35]. Consequently, these works mainly consider homomorphic computations, leaving encryption and decryption to the client. Our work considers more generally the impact that CE-RAM can have on the entire scheme, following the more versatile scenario of [3].

Other hardware solutions for efficient SHE have two types of data transfer that leads to latency. Data must be moved from the ordinary memory hierarchy (i.e. from DRAM) to the special hardware, and once resident in the accelerator it must be moved between storage and processing elements. In contrast, CE-RAM

Table 9. Speedup of Hom. Mult. against Roy et al.[35]

Actual data trans.	Spec. data trans.	Computation only
66.85x	36.6x	103.48x

Table 10. Speedup of CE-RAM against B/FV-GPU [2]

Case	Hom. Mult.	Hom. Add	Decrypt	Encrypt
Case A	19.9x	4.1x	2.0x	29.9x
Case B	**30.8x**	153.8x	7.9x	96.9x
Case C	19.7x	140.8x	5.0x	58.3x

can be directly integrated into a computer's ordinary memory hierarchy as a L3 cache, allowing data to be moved with the ordinary mechanisms for data transfer between the CPU and main memory. Once the data has been moved into CE-RAM, it is instantly available for computation without the need to move data to processing elements. CE-RAM also has the advantage of user-chosen word size. Other implementations (e.g., FPGAs, GPUs, ASICs, and CPUs) have word sizes that are either set or not easily configurable. In contrast, CE-RAM can easily choose word size as convenient for the application.

CE-RAM: Our work is the closest to the line of research seeded by Reis et al. [33], which was the first work pioneering the use of CE-RAM for the acceleration of SHE. We improve upon this work by implementing RNS and NTT optimizations, and choosing RNS moduli systems that allow novel optimizations especially friendly to CE-RAM systems. These optimizations give us an order of magnitude of speedup. We also consider actual data transfer times found in our evaluation, and present all essential functions of the B/FV scheme instead of homomorphic operations only. We further consider a higher level of parallelization (beyond simply within polynomials and RNS representations) by expanding our system to use a 16MB bank, which allows two simultaneous operations.

FPGA: The next most similar work to ours is that of Roy et al. [35], which uses an FPGA as a coprocessor for the more intensive portions of the B/FV encryption scheme. Their work involved careful pipelining and utilized RNS and NTT to accelerate polynomial arithmetic. They report a speedup of 13x as compared to a CPU implementation using an Intel i5 at 1.8GHz. Existing work in using CE-RAM has already been shown to improve upon the work of Roy et al., even without RNS or NTT optimizations [33].

HEAX [34] is another FPGA-based coprocessor architecture designed to accelerate the CKKS scheme [11], and it is the current state-of-the-art hardware accelerator. CKKS is similar to but substantially different from B/FV, as the ciphertext moduli in CKKS change with homomorphic multiplications. Despite this difference, we compare our optimized CE-RAM against HEAX since it is the state-of-the-art hardware accelerator with greatest speedups. HEAX reports speedups of two orders of magnitude, as compared to a CPU implementation using Microsoft SEAL. The scenario HEAX considers is multiple CPU-side processes interleaving data transfer to/from the external FPGA, while we consider only a single process using the standard memory hierarchy to transfer data to/from CE-RAM. (This is partially due to the still nascent software tooling available for CE-RAM). One of the main difficulties the HEAX architecture

faces is the limited on-chip memory available to the FPGA, forcing the FPGA to rely on off-chip memory. This induces extra cost and latency, though intelligent parallelism mitigates much of this. HEAX reports speedups of up to two orders of magnitude against a CPU implementation running on an Intel Xeon at 1.8GHz. The parameter set we consider is the same as the most intensive set of HEAX. When compared to a CPU implementation, HEAX achieved a speedup of 174.4x for homomorphic multiplication, which is on the same order of magnitude as our speedups of 748.2x against CPU (with actual data transfer time; our speedup is 430.1x with the specified transfer time). We can conclude that our work brings a similar or slightly better speedup to HEAX, even without any pipelining of data transfer implemented in CE-RAM. Assuming the concurrent operation of two 16MB CE-RAM banks as discussed in Section 4.4, for homomorphic multiplication we may observe a 6.38x speedup using this CE-RAM system as compared to HEAX in theory. For a further comparison to HEAX, we estimated the runtime of CKKS multiplication implemented in CE-RAM. We note that our system design and parameter choices are not optimized for CKKS due to the aforementioned difference. Further, we do not consider the pipelining and interleaving of execution utilized by HEAX, and only consider the end-to-end runtime of a single operation with data transfer to and from CE-RAM. With this consideration, we see a speedup of CE-RAM's CKKS homomorphic mult. of 6.82x (17841 ops/s) without data transfer, 3.19x (8363 ops/s) with the actual data transfer rate, and 1.44x (3755 ops/s) with the specified data transfer rate. Similarly, assuming two 16MB CE-RAM banks (Section 4.4), these speedups may be doubled in theory.

FPGAs have also been utilized as general-purpose cryptographic coprocessors for polynomial ring operations, and as coprocessors specifically designed for acceleration of the YASHE scheme (closely related to B/FV, but proven insecure [4]) [30]. There also exist FPGA-based accelerators of NTRU-based SHE schemes by applying RNS and NTT for polynomial operations. Ozturk et al. construct a FPGA-based accelerator for polynomial arithmetic intended for use in accelerating the LTV scheme [29, 27]. Against a CPU implementation, they report speedups of 102x for multiplication and 195x for relinearization. This is in the same order of magnitude with our speedups of 550x as reported in Table 6. Besides being used for scheme-specific operations, FPGAs have also been used to accelerate polynomial and integer arithmetic [12, 22] in ways that are generally useful for SHE, and for other applications. Another work [23] using FPGAs with CKKS attempts to decrease the amount of storage needed for NTT operations by computing twiddle factors on-the-fly, saving over 99.8% of the memory that would normally be used to store these. The lessened need for data transfer from this innovation resulted in a 118x speedup against a (less optimized) CPU implementation, and 28x against a similar work using FPGAs. One FPGA-based accelerator for homomorphic multiplication in the LTV scheme [14, 27] reported an speedup of 4.42x against a CPU implementation, with the polynomial degree 2^{14} (but only 80 bits of security) and a full API and Linux driver.

GPU: Al Badawi et al. implemented the Full RNS Variant of B/FV on a GPU using CUDA. Much of the innovation in their work comes from choosing the memory layout of polynomial representations and using parallelism to mitigate data transfer. The GPU implementation reports speedups of up to 13.18x for homomorphic multiplication as compared to Microsoft SEAL run on an Intel Xeon operating at 2.4GHz. Like our work, all essential SHE operations of the B/FV scheme can be run on the accelerator, making this work useful for more general use cases besides cloud-outsourced homomorphic encryption. In contrast, both HEAX and Roy et al. only consider a coprocessor architecture, where the most intensive operations are performed by the FPGA.

ASIC: There has also been some work in using ASICs for SHE [40], but ASICs are not easily reconfigurable, and are thus not as widely used in research despite their great efficiency.

CPU: There exist several high-performance software implementations of homomorphic encryption schemes for use on ordinary CPUs. These libraries include HELib implementing BGV [21], Microsoft SEAL [10] implementing B/FV and CKKS, FV-NFLib and PALISADE implementing the B/FV [25, 13] (with PALISADE implementing the floating-point RNS variant of B/FV [20]). While software implementations of homomorphic encryption are becoming mature, hardware acceleration generally outperforms general-purpose computing for homomorphic encryption. GPUs, FPGAs, and CPUs are further limited by hard bounds on machine word size, e.g. 64 bits for modern processors, 27 bits for HEAX (two words are used to store numbers, giving 54 bits), 30 bits for Roy et al. CE-RAM is not limited by a strict upper bound on word size, though it still faces the universal issue of larger numbers requiring asymptotically more computation.

RNS: Special RNS systems are well-studied in electrical engineering for their applications to parallel processing [39]. Many of these systems are not practical for use in CE-RAM due to their complexity, being originally conceived of in the context of FPGAs or other specialized hardware. However, we are able to observe and use some ideas from these works. In particular, we note that moduli near a power of two are amenable to the CE-RAM environment.

6 Conclusion

In this paper, we applied algorithmic optimizations to further accelerate the B/FV SHE scheme in CE-RAM. The optimizations we applied include the NTT, the full RNS variant, and new optimizations for modular reduction and $FastBConv$. We evaluate the effects of applying these optimizations, and compare the benefits of our work with a CPU-based software implementation, as well as with other relevant research efforts. For homomorphic multiplication of B/FV, our approach achieves speedups of up to 784.2x against a CPU server, 506.5x against previous work in CE-RAM [33], 66.85x against an FPGA acceleration [35], and 30.8x against a GPU implementation [3]. Compared to both GPUs and FPGAs, CE-RAM systems are currently at a nascent state with much

room of improvement. Nevertheless, CE-RAM SHE realizations show a greater benefit as compared to GPUs and FPGAs.

References

1. R. Agarwal and C. Burrus. Fast convolution using Fermat number transforms with applications to digital filtering. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 22(2):87–97, 1974.
2. A. Al Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung. High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda. *IACR CHES*, pages 70–95, 2018.
3. A. Q. 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 TETC*, 2019.
4. M. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched ntru assumptions. In *Annual International Cryptology Conference*, pages 153–178. Springer, 2016.
5. M. Albrecht et al. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.
6. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In *SAC*, pages 423–442. Springer, 2016.
7. J.-C. Bajard, J. Eynard, P. Martins, L. Sousa, and V. Zucca. Note on the noise growth of the rns variants of the bfv scheme.
8. J.-C. Bajard, J. Eynard, and N. Merkiche. Montgomery reduction within the context of residue number system arithmetic. *Journal of Cryptographic Engineering*, 8(3):189–200, 2018.
9. P. Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 311–323. Springer, 1986.
10. H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine. Simple encrypted arithmetic library v2. 3.0. *Microsoft*, 2017.
11. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ICTACIS*, pages 409–437. Springer, 2017.
12. A. Cilardo and D. Argenziano. Securing the cloud with reconfigurable computing: An fpga accelerator for homomorphic encryption. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1622–1627, 2016.
13. D. Cousins, K. Rohloff, Y. Polyakov, and G. J. Ryan. The PALISADE lattice cryptography library. `https://palisade-crypto.org/`, 2015–2020.
14. D. B. Cousins, K. Rohloff, and D. Sumorok. Designing an fpga-accelerated homomorphic encryption co-processor. *IEEE ToETiC*, 5(2):193–206, 2017.
15. R. Crandall and C. B. Pomerance. *Prime numbers: a computational perspective*, volume 182. Springer Science & Business Media, 2006.
16. J. P. Duarte et al. Bsim-cmg: Standard finfet compact model for advanced circuit design. In *ESSCIRC*, pages 196–201, Sep. 2015.
17. C. D. Environment. Cadence design systems. *Inc., www.cadence.com*, 2005.
18. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
19. C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.
20. S. Halevi, Y. Polyakov, and V. Shoup. An improved rns variant of the bfv homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*, pages 83–105. Springer, 2019.

21. S. Halevi and V. Shoup. Bootstrapping for helib. In *EUROCRYPT*, pages 641–670. Springer, 2015.
22. C. Jayet-Griffon, M. . Cornelie, P. Maistri, P. Elbaz-Vincent, and R. Leveugle. Polynomial multipliers for fully homomorphic encryption on fpga. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6, 2015.
23. S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar. Hardware architecture of a number theoretic transform for a bootstrappable rns-based homomorphic encryption scheme. In *IEEE FCCM*, pages 56–64. IEEE, 2020.
24. A. Langlois and D. Stehlé. Hardness of decision (r) lwe for any modulus. Technical report, Citeseer, 2012.
25. T. Lepoint. Fv-nfllib: Library implementing the fan-vercauteren homomorphic encryption scheme.
26. P. Longa and M. Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.
27. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *ACM STOC*, pages 1219–1234, 2012.
28. T. Oder, T. Güneysu, F. Valencia, A. Khalid, M. O'Neill, and F. Regazzoni. Lattice-based cryptography: From reconfigurable hardware to asic. In *ISIC*, pages 1–4, 2016.
29. E. Öztürk, Y. Doröz, B. Sunar, and E. Savas. Accelerating somewhat homomorphic evaluation using fpgas. *IACR Cryptol. ePrint Arch.*, 2015:294, 2015.
30. T. Pöppelmann, M. Naehrig, A. Putnam, and A. Macias. Accelerating homomorphic evaluation on reconfigurable hardware. In *IACR CHES*, pages 143–163. Springer, 2015.
31. F. Pourbigharaz and H. M. Yassine. Intermediate signed-digit stage to perform residue to binary transformations based on crt. In *IEEE ISCAS*, volume 2, pages 353–356 vol.2, 1994.
32. C. M. Rader. Discrete convolutions via Mersenne transforms. *IEEE Transactions on Computers*, C-21:1269–1273, 1972.
33. D. Reis, J. Takeshita, T. Jung, M. Niemier, and X. S. Hu. Computing-in-memory for performance and energy efficient homomorphic encryption. *arXiv preprint arXiv:2005.03002*, 2020.
34. M. S. Riazi, K. Laine, B. Pelton, and W. Dai. Heax: An architecture for computing on encrypted data. In *ACM ASPLOS '20*, pages 1295–1309, 2020.
35. S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *2019 IEEE International Symposium on HPCA*, pages 387–398. IEEE, 2019.
36. A. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in rns. *IEEE Trans. on Computers*, 38(2):292–297, 1989.
37. Synopsys Inc. HSPICE. *Version O-2018.09-1*, 2018.
38. J. Takeshita, M. Schoenbauer, R. Karl, and T. Jung. Enabling faster operations for deeper circuits in full rns variants of fv-like somewhat homomorphic encryption.
39. W. Wang, M. Swamy, M. O. Ahmad, and Y. Wang. A study of the residue-to-binary converters for the three-moduli sets. *IEEE ToCS I: Fundamental Theory and Applications*, 50(2):235–243, 2003.
40. E. Öztürk, Y. Doröz, E. Savaş, and B. Sunar. A custom accelerator for homomorphic encryption applications. *IEEE Trans. on Computers*, 66(1):3–16, 2017.

A Proofs for Novel Optimizations

In this section, we present proofs of correctness for our novel RNS optimizations.

A.1 Proof of Theorem 1

Proof. If x is even, then $x = 2y$, and $x \cdot 2^{g-1} = (2y)2^{g-1} = 2^g y \equiv y \mod 2^g - 1$. If x is odd, then $x = 2y + 1$, and $x2^{g-1} = (2y + 1)2^{g-1} = y2^g + 2^{g-1} \equiv 2^{g-1} + y \mod (2^g - 1)$. □

A.2 Proof of Lemma 2

Proof. $x(2^g - 1) = x \cdot 2^g - x \equiv -x \mod 2^g$. □

A.3 Proof of Theorem 2

Proof. If x is even, then $x = 2y$, and $x(2^{g-1} + 1) = (2y)(2^{g-1} + 1) = 2^g y + 2y \equiv -y + 2y \equiv y \mod 2^g + 1$. If x is odd, then $x = 2y + 1$, and $x(2^{g-1} + 1) = (2y + 1)(2^{g-1} + 1) = y2^g + 2y + 2^{g-1} + 1 \equiv -y + 2y + 2^{g-1} + 1 \equiv y + 2^{g-1} + 1 \mod 2^g - 1$. □

B Proofs for Fermat-like Coprimes

Let the terms q_i be elements of $S_2 = \{2^m - 1, 2^m + 1, 2^{2m} + 1, 2^{4m} + 1, \cdots 2^{2^{f-1}m} + 1, 2^{2^f m} + 1\}$, as in Section 3.2. Then the following results hold [31]:

Lemma 3. *For $q_i \in S_2$ with $q_0 = 2^m - 1$, $|\frac{q_0}{q}|_{q_0} = 2^{m-(k-1)}$.*

Proof. Note that $\frac{q}{q_0} = (2^m + 1)(2^{2m})(2^{4m}) \cdots (2^{2^f m})$. Because 2^m is equal to 1 modulo q_0, each of the $f + 1$ terms in this product is equal to two. Thus $|\frac{q}{q_0}|_{q_0} = |2^{f+1}|_{q_0}$. The inverse of this is $|\frac{q_0}{q}|_{q_0} = 2^{m-(f+1)} = 2^{m-(k-1)}$. □

Lemma 4. *For $q_i = 2^{2^i m} + 1$, $i \in [1, k]$, $|\frac{q_i}{q}|_{q_i}$ is $2^{2^{i-1}m-(f-i+2)}$.*

Proof. Note that $\frac{q}{q_i} = (2^{2^i m} - 1)(2^{2^{i+1}m} + 1)(2^{2^{i+2}m} + 1) \cdots (2^{2^f m} + 1)$. We see that $|(2^{2^i m} - 1)|_{q_i} = |(-1) - 1|_{q_i} = |-2|_{q_i}$. For the remaining terms $2^{2^j m} + 1$ (for $j \in [i+1, f]$), we have $|2^{2^j m} + 1|_{q_i} = |(2^{2^{i-1}m})^{2^{j-(i-1)}} + 1|_{q_i} = |(-1)^{2^{j-(i-2)}} + 1|_{q_i}$. Because $j > i$, this is equal to $|1+1|_{q_i} = 2$. Combining these, we see that $|\frac{q}{q_i}|_{q_i} = |(-2)2^{f-(i-1)}|_{q_i} = |-2^{f-i+2}|_{q_i}$. Then the inverse term $|\frac{q_i}{q}|_{q_i}$ is $2^{2^{i-1}m-(f-i+2)}$. □

In both of these terms, the exponent of two should always be positive; if this is not the case then too many moduli have been chosen for too small a dynamic range [38].

C NTT Algorithm

Algorithm 7 gives the algorithm of the Number-Theoretic Transform.

Algorithm 7 Number-Theoretic Transform (NTT) (From [35]).

Input: $a \in \mathbb{Z}_p^n (p \equiv 1 \bmod 2n)$, $Y \in \mathbb{Z}_p^n$ (CE-RAM stores powers of ψ in bit-reverse order, and $Y' = \lfloor Y \cdot 2^w / p \rceil$).
Output: $\tilde{a} \leftarrow NTT_p(a)$ in bit-reverse order.

1: **procedure** $NTT_p(a)$
2: **for** $(m = 1; m < n; m = 2m)$ **do**
3: **for** $(i = 0; i < m; i++)$ **do** ▷ Butterfly Loop
4: **for** $(j = \frac{i \cdot n}{m}; j < \frac{(2i+1)n}{2m}; j++)$ **do** ▷ NTT Core
5: $v \leftarrow MultRed(a_{j+\frac{n}{m}}, y_{m+i}, p)$
6: $a_{j+\frac{n}{m}} \leftarrow a_j - v \ (mod\ p)$
7: $a_j \leftarrow a_j + v \ (mod\ p)$
8: **end for**
9: **end for**
10: **end for**
11: $\tilde{a} \leftarrow a$
12: **end procedure**