Accelerating Lattice based Proxy Re-Encryption schemes on GPUs

Gyana Sahu[0000−0002−4423−3098] and Kurt Rohloff[0000−0003−0389−5092]

New Jersey Institute of Technology, Newark NJ 07102, USA
{grs22,rohloff}@njit.edu

Abstract. Proxy Re-Encryption (PRE) is an indispensable tool in many public-key cryptographic schemes that enables users to delegate decryption rights to other users via a proxy. In this work, we present a high performance implementation of PRE schemes on NVIDIA GPUs. We target two lattice based PRE schemes, BV-PRE and Ring-GSW PRE defined over polynomial rings. We design a parallel Number Theoretic Transform (NTT) procedure capable of working on arbitrary precision moduli (in CRT form) and demonstrate several low level and GPU optimizations techniques to accelerate the PRE schemes.
For the same or higher security settings our results show 39x to 228x factors of improvement in performance with a peak throughput of 6.3 Mbps when compared to the CPU implementation of the BV-PRE scheme in the PALISADE lattice crypto software library. Similarly, for the Ring-GSW PRE scheme we achieve a peak throughput of 49 Mbps and up to 11x improvement in performance.

Keywords: Homomorphic encryption · Ring-LWE · GPU acceleration · CUDA.

1 Introduction

First introduced in the work of Blaze, Bleumer and Strauss [3], Proxy Re-Encryption (PRE) is a powerful cryptographic primitive that allows a subscriber (Bob) to exchange and interpret encrypted data received from a publisher (Alice) without ever exchanging any secret key. To interpret the messages, Alice creates and gives to a Proxy (Polly) a re-encryption key which then allows Polly to transform messages encrypted with Alice's public key into an encrypted message that can be decrypted by Bob's secret key. Furthermore, semantic security of proxy re-encryption guarantees that the proxy Polly, does not learn anything about Alice's secret key or messages.

Proxy re-encryption can be a useful tool in brokering information exchanges in untrusted environments such as cloud computing platforms. Users can choose to store contents in encrypted form on the cloud and then register re-encryption keys of other users they want to interact with. Now, the cloud acting as a proxy can re-encrypt messages on the fly and deliver encrypted messages that can be read by the desired users. Even if the cloud is corrupted by a malicious adversary and all the data stored on the cloud is compromised, the adversary cannot retrieve meaningful information out of it. Further, an adversary in possession of re-encryption keys cannot deduce the secret keys of either the producer or consumer. Additionally, if the PRE scheme in deployment is *key private* secure then the adversary cannot even trace the identities of Alice and Bob. It can be easily seen that a PRE scheme is not just restricted to cloud computing environments but the same idea can be extended to other similar privacy concerning applications. PRE schemes have been proposed for use in digital rights management (DRM) systems [24], secure file storage systems [2], email list services [17], and many other applications.

In the literature of cryptology many proxy re-encryption schemes have been proposed starting with the pioneering work of the BBS [3] PRE scheme. Other PRE schemes have been known to be constructed on bilinear pairings and the decisional Diffie-Hellman (DBDH) assumption. Many of these schemes are computationally intensive and inefficient to implement in real world applications. Another problem associated with some of these PRE schemes is their *bi-directional* nature which allows them to perform re-encryption from publisher encrypted data to consumer encryption and also in reverse using the same re-encryption key. This is considered to be an undesirable property because of unwarranted rights acquired by the proxy. A *uni-directional* PRE scheme is more practical as it possesses translation capability in one direction only. Another important property that defines the flexibility of a PRE scheme is the number of hops for which it can be re-encrypted. A *multi-hop* PRE scheme was first presented by Canneti and Hohenberger [7]. Among the many open problems cited in their work was the construction of a PRE scheme which is simultaneously *multi-hop* and *uni-directional*.

The emergence of lattice based cryptography has paved the road for many new constructions with some of them being quite efficient and secure against quantum computers. Among the most notable developments is the breakthrough work of Gentry [14] on fully homomorphic encryption. A subsequent line of work by Brakerski and Vaikuntanathan [4–6] presented FHE schemes based on standard lattice assumptions which can be reduced to worst case hardness of approximating lattice problems. These FHE schemes are endowed with much better noise controlling mechanisms which allow them to evaluate circuits of greater depth. However, to keep the noise growth to a minimum, multiplication is often performed using a binary multiplication tree. Gentry *et al.* [13] introduced a FHE scheme roughly similar to Regev's [23] encryption scheme which further improved upon this noise growth during circuit evaluation by restricting it to a quasi additive nature. Such asymmetric noise growth can be used to support sequential multiplication of ciphertexts.

Building a PRE primitive with these FHE schemes presents a simple yet powerful approach for extending them for information exchanges in an untrusted environment. Further, these PRE schemes mitigate some of the above mentioned problems associated with traditional PRE schemes as they are inherently endowed with a *uni-directional* and *multi-hop* nature. A construction of a PRE scheme based on the BV [5] FHE scheme was presented in [22] where the authors exploited the key-switching procedure for achieving the re-encryption functionality. Along the same lines, we extended the PRE scheme to the Ring-GSW [16] FHE scheme. Evaluation of both BV-PRE and Ring-GSW-PRE schemes (with $r = 1$) with standard parameter set and security factor (100-bits) shows that the re-encryption procedure can be completed in the order of $10 - 100$ milliseconds.

Nowadays, owing to the ever increasing computational power and network efficiency most of the applications hosted on the cloud are expected to work in real time. An application involving a PRE primitive is no different. Being a low level primitive, the PRE scheme in deployment is expected to deliver the least possible latency. The above mentioned PRE schemes share the similarity of using the algebraic structure of ideal lattices as polynomial rings (RLWE). As a result, many of the computational bottlenecks in the implementation of these PRE schemes arise due to large ring dimensions, arbitrary precision multiplications, polynomial tensors, number theoretic transforms, matrix multiplication, etc. Over the years many of these problems have been remedied by switching over to better algorithms or optimizations. However, to provide additional speedups better hardware architectures have to be considered. In a heterogeneous computing model, these hardware accelerators acting as co-processors can be orchestrated by CPUs to achieve the desired levels of throughput. Among the common hardware accelerators such as FPGAs, ASICs and GPUs the most common and readily available solution is provided by GPUs. Modern GPUs consist of streaming multi-core processors which can be utilized to accelerate parts of computations that can be processed in parallel.

Lattice based cryptography and more specifically RLWE based FHE schemes being amenable to such parallelism have shown significant folds of performance improvement when implemented on GPUs. For example, in [8] the authors presented an implementation of the NTRU FHE scheme on GPUs and evaluated the AES and Prince block ciphers resulting in 2.5-7.6x factors of speedup over CPUs. Similarly, in [9] the authors present a homomorphic encryption accelerator library, cuHE targeting LTV [20], BGV [4] and DHS [11] FHE schemes. Speedups of 12-41x were reported for homomorphic sorting of ciphertexts. In another work, [16] leveraged the power of GPUs towards constructing a homomorphic Bayesian spam filter, secure multiple keyword search and evaluation of binary decision trees based on the Ring-GSW FHE scheme. For the same security settings [16] reported a factor of 10x improvement in performance when compared with the IBM HeLib [15] software library. Continuing in this line of work, we present an implementation of PRE schemes based on BV and Ring-GSW FHE schemes.

Our Contributions: We enumerate our main contributions and scope of the paper as follows.

– We present a GPU implementation of number theoretic transforms (NTT) based on finite field arithmetic where butterfly operations are performed in parallel. In order to extend the finite field over large numbers we keep integers in CRT representation. Our NTT implementation targets both small ($n \leq 1024$) and large ($n > 1024$) polynomial dimensions.

– Armed with a parallel implementation of NTT operations, next we target the parallelization of bit decomposition procedure. Relinearization along with bit/digit decomposition is considered to be the most critical procedure in many homomorphic encryption schemes and accelerating this operation imparts overall efficiency to the FHE scheme.
– Finally, utilizing the above implementations we demonstrate the acceleration of BV-PRE and Ring-GSW PRE schemes. In our implementation, we have reduced the number of memory transfers between host and device to a minimum by storing most of the dynamic elements on GPU memory. Another key feature of our implementation is the use of CUDA streams which allow concurrent execution of kernels thereby minimizing latency.

Paper Organization: In Section 2, we first provide the basic syntax of a public-key encryption scheme augmented with proxy re-encryption procedures. Section 3 introduces the mathematical notations and preliminaries we have used throughout the paper. Section 4 discusses the implementation of the underlying arithmetic layer, number theoretic transforms and bit decomposition procedures on the GPU along with other optimizations. The next two sections, 5 and 6 describe the lattice-based PRE schemes. In Section 7, we provide the parameters selected for implementation. Section 8 discusses the evaluation of PRE-schemes and their overall speedups.

2 Design

2.1 Syntax of unidirectional PRE Scheme

We recall that a non-interactive PRE scheme is an ensemble of PPT algorithms Π = (*ParamsGen, KeyGen, ReKeyGen, Encrypt, ReEncrypt, Decrypt*), which can be defined as per the following syntax:

– **ParamsGen**(1^λ): It takes the security parameter λ and returns the corresponding public parameters pp.
– **KeyGen**$(pp, 1^\lambda)$: It takes the public parameters pp and returns the key pair (pk, sk).
– **ReKeyGen**(pp, sk_i, pk_j): It takes the public parameters, secret key of publisher i, public key of subscriber j and returns the re-encryption key $rk_{i\to j}$.
– **Encrypt**(pp, pk, m): Given public key and public parameters, it encrypts the message m and returns a ciphertext c.
– **ReEncrypt**$(pp, rk_{i\to j}, c_i)$: It transforms a ciphertext c_i of the party i into a ciphertext c_j that can be decrypted by party j.
– **Decrypt**(pp, sk, c): It recovers the message m from ciphertext c.

3 Preliminaries and Mathematical Notations

In this work, we represent scalars in plain, vectors by lower-case bold letters (e.g., $\mathbf{a}$) and matrices by upper-case bold letters (e.g., $\mathbf{A}$). Ring elements, for example

$x \in R_q$, are represented by lower case letters in plain. We define $[k] = \{1, \cdots, k\}$ for any non-negative integer k. The i-th norm of a vector $\mathbf{v}$ is denoted by $\|\mathbf{v}\|_i$. We denote the tensor (Kronecker) product of two matrices $\mathbf{A}$ and $\mathbf{B}$ as $\mathbf{A} \otimes \mathbf{B}$. We extend this notation to a vector $\mathbf{v}$ where the Kronecker product is represented as $\mathbf{v} \otimes \mathbf{A}$. Unless explicitly mentioned logarithms are to be understood with base 2. We denote the horizontal and vertical concatenation of matrices by operators $(\cdot||)$ and $(\cdot||^{\intercal})$ respectively. For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}^{n \times n}$ $[\mathbf{A} \mid\mid \mathbf{B}]$ produces a matrix $\mathbf{C} \in \mathbb{Z}^{n \times 2n}$. Similarly, $[\mathbf{A} \mid\mid^{\intercal} \mathbf{B}]$ produces a matrix $\mathbf{C} \in \mathbb{Z}^{2n \times n}$.

3.1 Gadget Matrix and Relinearization functions:

For LWE dimension n and modulus q, we use the following "gadget" vector [21]:

$$\mathbf{g} = \left(1, 2, 4, \cdots, 2^{\ell-1}\right) \in \mathbb{Z}_q^{\ell}, \text{ where } \ell = \lceil \log q \rceil.$$

The *gadget matrix* $\mathbf{G}$ is then defined as the diagonal concatenation of the $\mathbf{g}$ vector n times. Formally, it is written as $\mathbf{G} = \mathbf{g} \otimes I_n \in \mathbb{Z}_q^{\ell n \times n}$. To perform relinearization we define the following operations [5,13] with respect to an element a which can be either a vector or matrix or polynomial ring.

- **BitDecomp** (a): For a vector $\mathbf{a} \in \mathbb{Z}_q^n$ this operation produces a bit decomposed and expanded vector $\mathbf{a}' \in \mathbb{Z}_q^{n\ell}$ where $a_i = \sum_{j=0}^{\ell} 2^j a'_{i\ell+j}$. In case of a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ this operation produces a matrix that is expanded along the column resulting in $\mathbf{A}' \in \mathbb{Z}_q^{m \times n\ell}$. Finally, in case of a polynomial ring $a \in R_q$ this operation produces $\mathbf{a}' \in R_q^{\ell}$.
- **PowerOf2** (a): Given a vector $\mathbf{a} \in \mathbb{Z}_q^n$ this operation produces an expanded vector $\mathbf{a}' \in \mathbb{Z}_q^{n\ell}$ where $\mathbf{a}' = \left(a_0, 2a_0, \cdots, 2^{\ell-1}a_0, \cdots, 2^{\ell-1}a_{n-1}\right)$. Similarly, for a polynomial ring $a \in R_q$ we get $\mathbf{a}' \in R_q^{\ell}$ where $a'_i = 2^i a$.

Using these operations, we can produce a product of $\mathbf{a}$ and $\mathbf{b}$ as follows:

$$\langle \mathbf{BitDecomp}\left(\mathbf{a}\right), \mathbf{PowersOf2}\left(\mathbf{b}\right) \rangle = \langle \mathbf{a}, \mathbf{b} \rangle$$

4 Number Theoretic Transform and Bit-Decomposition

4.1 Number Theoretic Transform

Cryptosystems based on the RLWE security assumption are defined over a polynomial ring $R = \mathbb{Z}\left[X\right]/\Phi_m\left(X\right)$ where $\Phi_m\left(X\right)$ is an irreducible monic cyclotomic polynomial of order m. This notation is extended to a polynomial R_q modulo an integer q where the coefficients of the polynomial are in the interval $(-q/2, q/2]$. Alternatively, an element $a \in R_q$ is simply considered to be a coefficient vector $\mathbf{a} \in \mathbb{Z}_q^{\varphi(m)}$. While addition of these polynomials is quite efficient, multiplication leads to quadratic time complexity. To circumvent this inefficiency, we represent polynomial rings in the so called "Evaluation" representation. For a polynomial $a \in R_q$ the coefficients can be converted to the evaluation domain $\bar{a}$ by evaluating $a(X)$ at each of the m-th primitive roots of unity modulo q. The coefficients of $\bar{a}$ are related to polynomial a through the relation $\bar{a}_i = a\left(\omega^i\right) \mod q$ where $(i, m) = 1$ and ω is a m-th primitive root of unity modulo q.

This back and forth conversion of a polynomial can be achieved efficiently by using number theoretic transforms (NTT) which is roughly similar to the classical n-dimensional fast Fourier transform where a finite field is used instead of complex numbers. Concretely, in our implementation, we use the power of two cyclotomics ($m = 2^k$) where $\Phi_m(X)$ is maximally sparse and the ring dimension $n = \varphi(m) = m/2$ is also a power of two. The power of two cyclotomics along with NTT has become so pervasive in lattice based cryptography that the overall efficiency of the cryptosystem depends upon the latency of NTT procedure. For this reason, we chose to implement NTT as an iterative Cooley-Tukey algorithm. More specifically, we implemented the NTT routine with Fermat theoretic transform (FTT) optimization which eliminates interleaved zero paddings when using the conventional NTT procedure.

4.2 Parallel NTT

Exploiting the NVIDIA GPU architecture, we reduce the NTT latency further by mapping the butterfly computations of each of the $\log n$ stages to an independently processing thread of a thread block. In the NVIDIA CUDA architecture, each kernel or device function can be potentially divided into a 3-dimensional array of blocks where a block further consists of several threads. Because of hardware restrictions, a maximum of 1024 threads can be assigned to a block. Further, the threads within a block have the capability to share data and more importantly synchronize with each other. In our implementation, for small polynomials ($n \leq 1024$) we map the coefficients entirely to a thread block and synchronize the thread block after completion of a stage as shown in Figure 1. We use shared memory for storing the intermittent results as latency associated with global memory is higher than that of shared memory which resides on the chip. For larger polynomial rings ($n > 1024$), we use a combination of block level synchronization and stream level synchronization to avoid data race conditions. As stream level synchronization avoids data race conditions via global memory synchronization we pay the penalty of using slower memory but only for a fraction of the NTT procedure calls.

The evaluation of the proposed NTT procedure on GPU platforms and CPU platforms is shown in Figure 2. From the figure, we can see that the CPU platform running on a single thread achieves slightly better performance for smaller ring dimensions. As the ring dimensions grow higher we can see that the GPU platform starts showing significant improvement in performance.

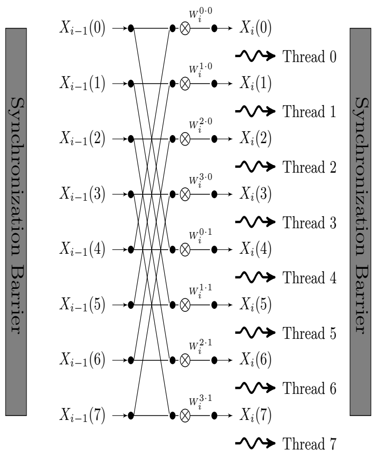

Fig. 1: Parallel implementation of the i-th stage of NTT on a GPU, $N = 8$.

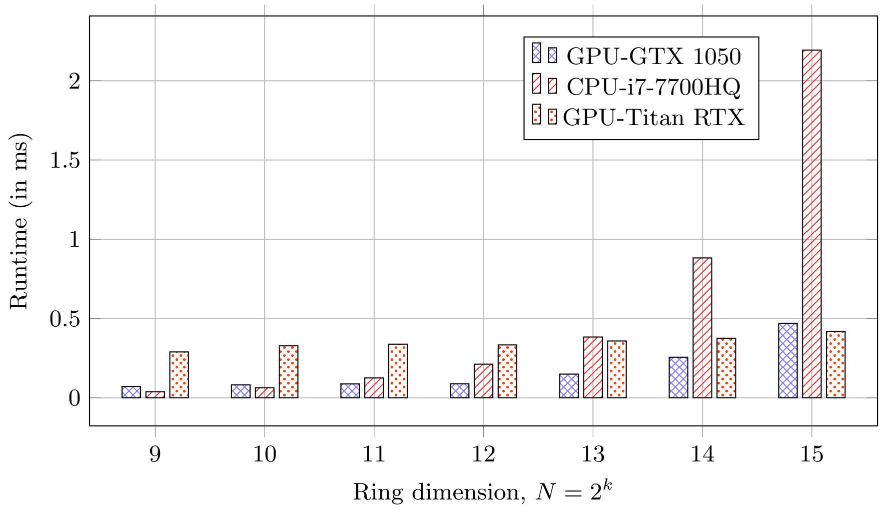

Fig. 2: Comparison of CPU and GPU runtimes of the NTT algorithm.

4.3 Barrett Modulo Reduction and Arbitrary Precision Support

For modulo reduction, we used a variation of the generalized Barrett modulo reduction [10] as outlined in Algorithm 1. Barrett modulo reduction requires a pre-computation term $\mu = \lfloor 2^{2b}/q \rfloor$ for a particular modulus q and its bit width, $b = \lceil \log_2 q \rceil$. We pre-compute these terms and transfer them to GPU global memory for read only access to any kernel. NVIDIA GPUs are restricted to a 32-bit architecture and 64-bit arithmetic are supported only through assembly code emulation. For this reason, we prefer moduli with bit width closer to 32-bits so that the precomputation term μ fits into the word size. Concretely, we allow up to $29 - 30$ bit width moduli in our implementation.

Lattice based cryptosystems employ the addition of low norm noise terms to base their security on Ring-LWE and LWE assumptions. For preserving the correctness constraints so that ciphertexts are decrypted correctly, the modulus q should be chosen large enough such that the final accumulated error terms do not "wrap around" modulo q. To extend support for larger moduli we store a set of increasing prime moduli q_i by an application of the Chinese Remainder Theorem (CRT). We only reconstruct the coefficients into larger terms for the purpose of decryption or bit-decomposition where the polynomial needs to be represented in terms of the larger modulus, $q = \prod_{i=0}^{t-1} q_i$.

Evaluation of the NTT procedure on a GPU with varying number of moduli, t and ring dimension N can be seen in Figure 3. For most of the ring dimensions, the runtimes vary a little. This is due to the fact that GPUs have the capability to improve throughput by hiding latency with the concurrent execution of the NTT procedure on different polynomials. On a CPU platform the runtime is estimated to scale linearly with the number of moduli assuming a single thread execution environment.

Algorithm 1: Mod Barrett Reduction

```
Input  : $x \in [0, (q-1)^2]$, modulus $q$, bit-width $b = \lceil \log q \rceil$, $\bar{q} = 2 \cdot q$ and
         $\mu = \lfloor 2^{2b}/q \rfloor$.
Output: $z \leftarrow x\%\ q$
1  $z \leftarrow x \gg b$ ;
2  $z \leftarrow z \cdot \mu$ ;
3  $z \leftarrow z \gg b$ ;
4  $z \leftarrow z \cdot q$ ;
5  $x \leftarrow x - z$ ;
6  if $x >= \bar{q}$ then
7      $z \leftarrow x - \bar{q}$;
8  end if
9  if $z >= q$ then
10     $z \leftarrow z - q$;
11 end if
12 return $z$
```

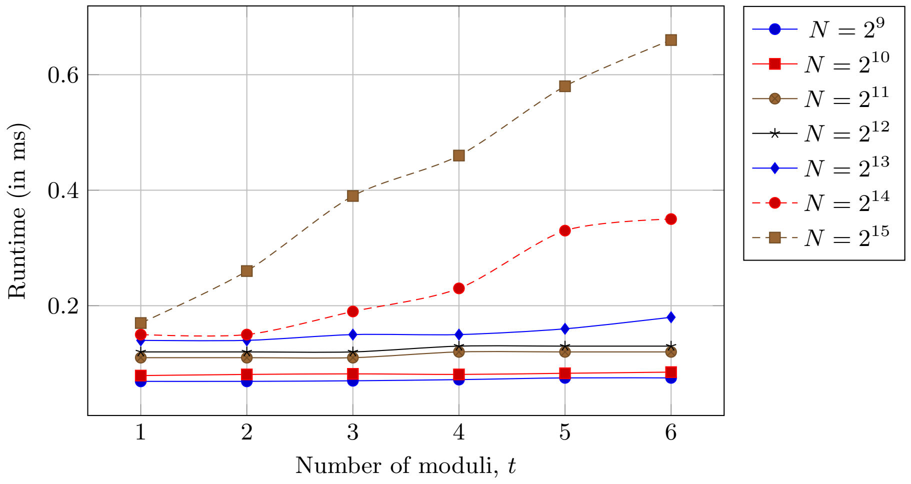

Fig. 3: GPU runtimes of the NTT algorithm with varying dimension N and moduli t.

4.4 Bit Decomposition

Bit decomposition along with the relinearization procedure forms the backbone of lattice based cryptography. While bit decomposing integers is simple in finite field arithmetic, it is accompanied with additional overheads in Ring-LWE based cryptosystems. In Ring-LWE cryptosystems, ciphertexts and other key elements are mostly present in evaluation representation. To bit decompose, polynomial rings need to be switched back to coefficient representation. At this stage, bit decomposition of the polynomial results in a vector of b polynomials, where b is the bit length of the modulus q. To perform further computations, these polynomials need to be converted back to evaluation representation by a series of NTT calls. Since the bit decomposed polynomials are independent of each other, we apply NTT procedures on them in parallel using a three dimensional CUDA grid mapping. To avoid race conditions in the NTT procedure, we provide the kernel with appropriate synchronization. From Figure 4, we can observe that our GPU implementation of bit decomposition outperforms runtimes of the CPU platform for all ring dimensions and further the speedups are more pronounced in case of higher ring dimensions.

5 PRE Cryptosystem with BV FHE scheme

The BV-PRE [22] scheme described here is based on the BV FHE [5] scheme introduced by Brakerski and Vaikuntanathan. The message space for the scheme is restricted to $\mathcal{M} \in R_p$ where p is the plaintext modulus.

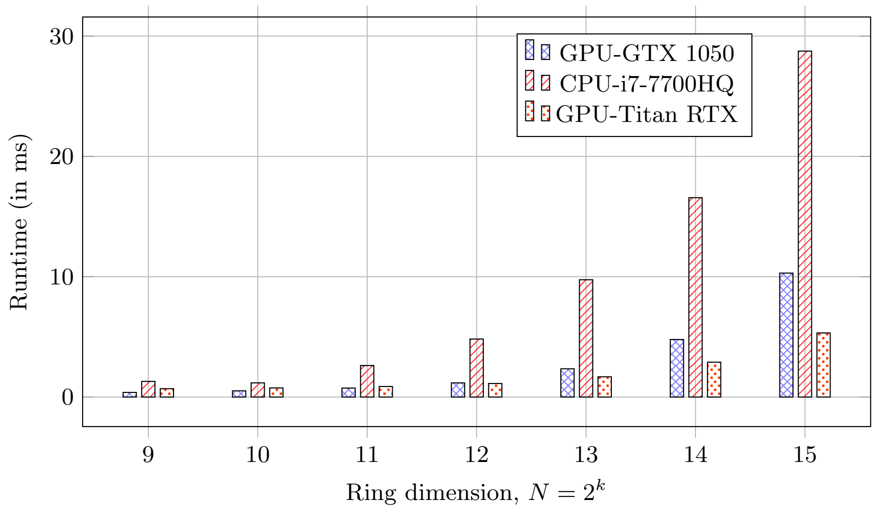

Fig. 4: GPU vs CPU runtimes of bit decomposing a polynomial ring with varying ring dimension N.

5.1 BV Encryption Scheme

The scheme is parameterized using the following quantities:

– Security parameter (λ),
– Ciphertext modulus q and plaintext modulus $2 \leq p \ll q$,
– Ring dimension n,
– D-bounded discrete Gaussian error distribution χ_e with distribution parameter σ_e,
– Ternary uniform distribution $\mathcal{T}$ which samples from $\{-1, 0, 1\}$,
– Discrete uniform distribution $\mathcal{U}_q$,

The scheme consists of the following operations:

• **ParamsGen**(1^λ): Choose positive integers $q = q(\lambda)$ and $n = n(\lambda)$. Return $pp = (p, q, n)$.

• **KeyGen**(pp, λ): Sample polynomials $a \leftarrow_{\$} \mathcal{U}_q$, $s \leftarrow_{\$} \mathcal{T}$ and $e \leftarrow_{\$} \chi_e$. Compute $b := a \cdot s + pe \in R_q$. Set the public key pk and private key sk as follows:

$$sk := (1, s) \in R_q^2,\ pk := (a, b) \in R_q^2$$

• **Encrypt**$(pp, pk = (a, b), m \in \mathcal{M})$: Sample polynomials $v \leftarrow_{\$} \mathcal{T}, e_0, e_1 \leftarrow_{\$} \chi_e$. Compute the ciphertext $c = (c_0, c_1) \in R_q^2$ as follows:

$$c_0 = b \cdot v + pe_0 + m \in R_q,\ c_1 = a \cdot v + pe_1 \in R_q$$

• **Decrypt**$(pp, sk = s, c = (c_0, c_1))$: Compute the ciphertext error $t = c_0 - s \cdot c_1 \in R_q$. Output $m' = t \bmod p$. For correct decryption, the coefficients of the noise polynomial, t should not wrap around modulo q.

5.2 Proxy Re-Encryption Scheme

We refer to the publisher of information as party A and the subscriber of information via the proxy as party B. Additional operations pertaining to PRE computation are as follows:

• **Preprocess**$(pp, \lambda, sk_B = (1, s_B))$ Sample uniformly distributed random polynomials $\alpha_i \leftarrow_\$ \mathcal{U}_q$ and error polynomials $e_i \leftarrow_\$ \chi_e$ for $i \in [0,\ \lceil \log_2(q)/r \rceil)$. Here r is the relinearization window. Compute the following elements:

$$\gamma_i = \alpha_i \cdot s_B + pe_i \in R_q;\ pk_B = (\alpha_i, \gamma_i)_{i \in \{0,1,\cdots,\lfloor \log_2(q)/r \rfloor\}}$$

• **ReKeyGen**$(pp, sk_A = (1, s_A), pk_B)$: Compute β_i and set the re-encryption key as follows:

$$\beta_i = \gamma_i - s_A \cdot (2^r)^i \in R_q,\ rk_{A \to B} = (\alpha_i, \beta_i)_{i \in \{0,1,\cdots,\lfloor \log_2(q)/r \rfloor\}}$$

• **ReEncrypt**$(pp, rk_{A \to B} = (\alpha_i, \beta_i), c = (c_0, c_1))$: To get the re-encrypted ciphertext $c' = (c'_0, c'_1)$ first apply 2^r base decompositions and proceed as follows:

$$c'_0 = c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \beta_i\right),\ c'_1 = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \alpha_i\right)$$

where $c_1^{(i)}$ represents the i-th digit of the base-2^r decomposition of c_1.

Under the new secret key $sk = (1, s_B)$ decryption of the ciphertext $c' = (c'_0, c'_1)$ can be shown as

$$\begin{aligned} c'_0 - s_B \cdot c'_1 &= c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \beta_i\right) - s_B \cdot \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \alpha_i\right) \\ &= c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \left\{\alpha_i \cdot s_B + pe_i - s_A \cdot (2^r)^i\right\}\right) \\ &\qquad\qquad - s_B \cdot \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \alpha_i\right) \\ &= c_0 - s_A \cdot c_1 + pE_i;\ E_i = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot e_i\right) \end{aligned}$$

To preserve the correctness of the re-encrypted ciphertext $c' = (c'_0, c'_1)$, we derive the following constraint:

$$\begin{aligned} \|t\|_\infty &\leq 3\sqrt{n}pD + \sqrt{n}pD \cdot D_r \lceil \log_2(q)/r \rceil \\ &\approx \sqrt{n}pDD_r \left(\lceil \log_2(q)/r \rceil + 1\right) \ni r > 1 \\ \Rightarrow q &\geq 2\sqrt{n}pDD_r \left(\lceil \log_2(q)/r \rceil + 1\right) \end{aligned}$$

where, D_r is the bound of the bit decomposed polynomial in base 2^r representation.

5.3 Security

The security of PRE schemes is generally covered under indistinguishability against chosen-plaintext attacks (IND-CPA). For brevity we omit the formal definition of the IND-CPA security notion and capture the security with the following theorem.

Theorem 1. *[22] Under the* $\mathbf{RLWE}_{\Phi,q,\chi_e}$ *assumption, the BV-PRE scheme is* IND-CPA *secure. Specifically, for a poly-time adversary* $\mathcal{A}$*, there exists a poly-time distinguisher* $\mathcal{D}$ *such that*

$$Adv_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot Adv_{\mathcal{D}}^{\mathbf{RLWE}_{\Phi,q,\chi}}$$

where Q_{rk} *and* Q_{re} *are the numbers of re-encryption key queries and re-encryption queries, respectively;* N *is the number of honest entities;* λ *is the security parameter;* Φ *is the cyclotomic polynomial defining the ring* $R_q = \mathbb{Z}_q[x]/\langle\Phi\rangle$ *and* $\rho = \lceil \log_2(q)/r \rceil$.

6 PRE Cryptosystem with Ring-GSW FHE scheme

The Ring-GSW PRE scheme described here is based on the Ring-GSW FHE scheme, a RLWE variant of the GSW [13] FHE scheme. The message space for the scheme is restricted to $\mathcal{M} \in R_p$ similar to the BV-FHE scheme.

6.1 Ring-GSW Encryption Scheme

The scheme is parameterized similar to the BV-FHE scheme. It consists of the following operations:

• **ParamsGen**(1^λ): Choose positive integers $q = q(\lambda)$ and $n = n(\lambda)$. Return $pp = (\ell, N, p, q, h, n)$ where $\ell = \lceil \log_2(q)/h \rceil$ and $N = 2\ell$. Here h is the relinearization factor.

• **KeyGen**(pp, λ): Sample polynomials $a \leftarrow_{\$} \mathcal{U}_q$, $s \leftarrow_{\$} \mathcal{T}$ and $e \leftarrow_{\$} \chi_e$. Compute $b := a \cdot s + pe \in R_q$. Set the public key pk and private key sk as follows:

$$sk := (1; -s) \in R_q^{2\times 1},\ pk := \mathbf{A} = [a\ b] \in R_q^{1\times 2}$$

• **Encrypt**$(pp, pk = \mathbf{A}, m)$: Sample random matrix $\mathbf{R}$ from ternary distribution and an error matrix $\mathbf{E} \in R_q^{N\times 2}$ from discrete Gaussian distribution. Compute the ciphertext $\mathbf{C} \in R_q^{N\times 2}$ as follows:

$$\mathbf{R} = \{r_0, \cdots, r_{N-1}\} \leftarrow_{\$} \mathcal{T}_{R_q}^N,\ \mathbf{E} \leftarrow_{\$} \chi_{e,R_q}^{N\times 2}$$
$$\mathbf{C} = m \cdot \mathbf{G} + \mathbf{R} \otimes \mathbf{A} + p\mathbf{E}$$

• **Decrypt**$(pp, sk = (1; -s), \mathbf{C})$: Message m' is recovered by multiplying the first row of the ciphertext $\mathbf{C}$ with the secret key sk. This is shown as:

$$m' = (\mathbf{C}_0 \times sk \bmod q) \bmod p$$

Decryption works correctly as the first row of the ciphertext is in the form of a BV scheme ciphertext.

6.2 Proxy Re-Encryption Scheme

We describe the operations pertaining to PRE computation for a publisher A and a subscriber B as follows:
• **ReKeyGen**(pp, sk_A, pk_B): The re-encryption key consists of two polynomial matrices. To generate the matrices, we first sample two ternary uniform matrices $\mathbf{R}_0, \mathbf{R}_1 \in R_q^N$. Next, we sample two error matrices from χ_{e,R_q} and set the evaluation matrices $\mathbf{EK}\,(i)$ as follows:

$$\mathbf{R}_i \leftarrow_{\$} \mathcal{T}_{R_q}^N, \quad \mathbf{E_i} \leftarrow_{\$} \chi_{R_q,B}^{N\times 2}, \quad i \in \{0,1\}$$
$$\mathbf{EK}[i] = \mathbf{R}_i \otimes \mathbf{A}_B + p\mathbf{E}_i + (\text{PowerOf2}\,(sk_A) \gg i)$$
$$rk_{A\to B} = \{\ \mathbf{EK}[0],\ \mathbf{EK}[1]\ \}$$

• **ReEncrypt**$(pp, \mathbf{C}_A, rk_{A\to B})$: We use the top ℓ rows of the ciphertext $\mathbf{C}_A$ to perform re-encryption and denote this as $\mathbf{C}_A^{\text{top}}$. Next, we multiply each of the matrices $\mathbf{EK}[i]$ with $\mathbf{C}_A^{\text{top}}$ and reassemble the results into a Ring-GSW ciphertext $\mathbf{C}_{A\to B}$. This is shown as follows:

$$\mathbf{C}_{A\to B}^i = \text{BitDecomp}\left(\mathbf{C}_A^{\text{top}}\right) \cdot \mathbf{EK}[i] \in R_q^{\ell\times 2}$$
$$\mathbf{C}_{A\to B} = \left[\mathbf{C}_{A\to B}^0 \,||^{\intercal}\, \mathbf{C}_{A\to B}^1\right]$$

To formulate the correctness constraint we have to ensure that there is no wrap around mod-q in the noise term t while decrypting the re-encrypted ciphertext. The noise term t is given by:

$$t = \mathbf{C}_{A\to B,0} \times sk_B = \mathbf{C}_{A\to B,0,0} - s_B\mathbf{C}_{A\to B,0,1}$$

Let $\alpha_i = \text{BitDecomp}\left(\mathbf{C}_{A,j,0}^{\text{top}}\right)$ and $\beta_i = \text{BitDecomp}\left(\mathbf{C}_{A,j,1}^{\text{top}}\right)$ for $(i,j) \in [0,\ell)$. Then, $\mathbf{C}_{A\to B,0}$ can be shown as:

$$\mathbf{C}_{A\to B,0} = [\alpha_i\ \beta_i]_{i=0}^{\ell-1} \cdot [r_j\mathbf{A}_B + p\mathbf{E}_j + \text{PowerOf2}\,(sk_A)]$$
$$\text{where } i \in [0,\ell) \text{ and } j \in [0, 2\ell)\,.$$

$$\begin{aligned}\mathbf{C}_{A\to B,0,0} &= b_B \sum_{i=0}^{\ell-1} (\alpha_i r_i + \beta_i r_{\ell+i}) + p\sum_{i=0}^{\ell-1}(\alpha_i\mathbf{E}_{i,0} + \beta_i\mathbf{E}_{\ell+i,0}) \\ &+ \sum_{i=0}^{\ell-1}\alpha_i\text{PowerOf2}\,(1) + \sum_{i=0}^{\ell-1}\beta_i\text{PowerOf2}\,(-s_A) \\ &= b_B r_0' + p\mathbf{E}_{0,0}' + \alpha - s_A\beta\end{aligned}$$

Similarly,

$$\begin{aligned}\mathbf{C}_{A\to B,0,1} &= a_B \sum_{i=0}^{\ell-1} (\alpha_i r_i + \beta_i r_{\ell+i}) + p\sum_{i=0}^{\ell-1}(\alpha_i\mathbf{E}_{i,1} + \beta_i\mathbf{E}_{\ell+i,1}) \\ &= a_B r_0' + p\mathbf{E}_{0,1}'\end{aligned}$$

Therefore,

$$t = r_0' \left(b_B - a_B s_B\right) + p\left(\mathbf{E}_{0,0}' - s_B \mathbf{E}_{0,1}'\right) + \alpha - s_A\beta \ \cong m \bmod p$$

For correct decryption $\|t\|_\infty \leq q/2$. By using the central limit theorem we arrive at the final correctness constraint:

$$\|t\|_\infty \leq 2p\sqrt{n}D_h D\lceil \log_2(q)/h\rceil \cdot \left(2\sqrt{n}+1\right) + 3p\sqrt{n}D \approx 6pnD_h D\lceil \log_2(q)/h\rceil$$
$$\Rightarrow q \geq 12pnD_h D\lceil \log_2(q)/h\rceil$$

6.3 Security

IND-CPA security of the Ring-GSW-PRE scheme is defined in a similar manner as in the BV-PRE scheme and only differs in the parameter of ρ which describes the number of RLWE samples in the re-encryption key.

Theorem 2. *Under the $\boldsymbol{RLWE}_{\Phi,q,\chi_e}$ assumption, the Ring-GSW PRE scheme is* IND-CPA *secure. Specifically, for a poly-time adversary $\mathcal{A}$, there exists a poly-time distinguisher $\mathcal{D}$ such that*

$$Adv_{\mathcal{A}}^{cpa}(\lambda) \leq \left(\rho \cdot (Q_{rk} + Q_{re}) + N + 1\right) \cdot Adv_{\mathcal{D}}^{\mathbf{RLWE}_{\Phi,q,\chi_e}}$$

where Q_{rk} and Q_{re} are the numbers of re-encryption key queries and re-encryption queries, respectively; N is the number of honest entities; λ is the security parameter; Φ is the cyclotomic polynomial defining the ring $R_q = \mathbb{Z}_q[x]/\langle\Phi\rangle$ and $\rho := 4\lceil\log_2(q)\rceil$.

7 Parameter Selection

Estimating parameters for LWE or RLWE based encryption schemes is a significantly challenging task. On one hand we have to ensure that the chosen parameters generate an underlying RLWE instance that is hard to solve as per known attacks while on the other hand we have to also meet the correctness constraint. The correctness constraint can be trivially achieved by choosing an arbitrarily large modulus q. However, such a strategy is not suitable for efficient implementation and further leads to insecure LWE instances. It was shown in [18] when the modulus is exponential in the LWE dimension, $q \geq 2^{\mathcal{O}(n)}$ and error distribution is narrow enough, the secret key can be recovered in polynomial time using standard lattice basis reduction algorithms. In order to layout concrete parameters, Lindner and Peikert [19] gave a heuristic relation that computes the runtime of BKZ lattice reduction algorithm for a particular root Hermite factor, δ. This is shown as:

$$\log_2(t_{BKZ}) \geq \frac{1.8}{\log_2(\delta)} - 110$$

Further, Gentry *et al.* [12] gave a relation which computes minimum LWE dimension secure for a particular modulus q, standard deviation of error distribution σ_e and root Hermite factor, δ. Combining the two we can express the minimum LWE dimension to support κ-bits of security as follows:

$$n \geq \frac{\log_2(q/\sigma_e)(\kappa + 110)}{7.2}$$

In particular, we used the runtime of BKZ2.0 [1] (considered to be a improved version of BKZ algorithm) given by the relation

$$\log_2(t_{\mathrm{BKZ2.0}}) \geq \frac{0.009}{\log_2^2 \delta} - 27$$

Again, combining this with the minimum LWE dimension relation from [12] we arrive at our final security constraint as follows:

$$n \geq \frac{\log_2(q/\sigma_e)\sqrt{\kappa + 27}}{0.379}$$

In our implementation we targeted for $\kappa = 128$-bits of security for both BV-PRE and Ring-GSW-PRE scheme. Using the correctness constraint from the respective schemes we generated the working modulus for various ring dimensions as shown in Table 1.

n	b	
	BV-PRE scheme	Ring-GSW-PRE scheme
512	18	-
1024	18	25
2048	19	26
4096	19	27
8192	20	28
16384	20	29

Table 1: Minimum modulus bits, b that satisfies 128-bits of security for the BV-PRE and Ring-GSW PRE scheme according to the respective correctness constraints and varying dimension, n. $D = 15$ and $\sigma = 4$.

8 GPU Implementation and Results

8.1 Software Implementation

We evaluated both the BV-PRE and the Ring-GSW-PRE scheme on two NVIDIA GPU devices, namely, GeForce GTX-1050 and Titan-RTX as shown in Table 2.

While GPU1 is a commodity grade notebook GPU, GPU2 is a much more powerful GPU targeted towards compute intensive applications. Our software implementation follows the modular structure of the PALISADE homomorphic encryption library separating crypto implementations from lower level math layers. Our implementation is compiled with the CUDA 10.0 NVCC compiler along with C++14 support. Our single threaded execution environment consists of 64-bit x86 architecture with operating system for GPU1 and GPU2 as Ubuntu 18.04 and Scientific Linux 6.10 (available on university HPC) respectively.

Feature	CPU1	CPU2	GPU1	GPU2
Model	Intel i7-7700HQ	Intel Silver 4114	NVIDIA GeForce GTX 1050	NVIDIA Titan RTX
Cores	4	10	640	4608
Clock Rate	2.8 GHz	2.2 GHz	1.49 GHz	1.77 GHz
Multiprocessors	-	-	5	72
RAM Memory	16 GBytes	181 GBytes	4042 MBytes	24190 MBytes
CUDA Capability	-	-	6.1	7.5

Table 2: Configuration and features of GPU with their corresponding CPU used for the evaluation of the PRE schemes.

In our implementation we primarily aimed to improve the runtimes of operations pertaining to the encryption scheme and the PRE scheme by switching the CPU routines with CUDA kernel calls. We outline the main aspects of our GPU optimizations as follows:

• **Optimized pre-computation phase**: Our implementation consists of a pre-computation phase wherein we compute most of the cryptosystem parameters and NTT related parameters on the CPU and transfer them to GPU global memory. For faster memory transfers we use coalesced memory spaces which require fewer memory transfer calls. On the CPU side we mostly use memory allocation using pinned host memory. Memory transfer using pinned memory is faster than pageable host memory because the GPU can directly access such memory spaces. Lastly, we remark that we do not make use of constant or texture memory as it reported little or no improvement in performance. Moreover, such memory spaces are available in very limited number and not scalable for higher ring dimensions.

• **Memory related optimizations**: Being a heterogeneous platform, one should expect a significant amount of data transfers between CPU and GPU memory. However, it is known that such data transfers are generally slower (because of lower bandwidth PCIe) and can sometimes degrade the overall performance of an application. To get a more realistic performance estimate of the various operations, we eliminated most of the data transfers by allocating memory for most of the cryptosystem elements on GPU memory directly.

• **Fast Random Generators**: Our implementation relies on the CUDA random number generation library cuRAND for generating random polynomials.

The distributions targeted in our application are uniform random distribution $\mathcal{U}_q$, discrete Gaussian distribution χ_e with standard deviation σ_e, binary and ternary uniform distributions. Except for uniform distribution, all other distributions were generated using continuous Gaussian distribution on pre-allocated memory and then launching appropriate kernels to reduce them in a particular range. More specifically, we used the CURAND RNG PSEUDO MTGP32 set of generators which is 5x faster than other random number generators of the same family and atleast 10x faster than CPU random number generators. We remark that we have not evaluated the cryptographic security of this random number generator and assume that it generates long enough internal states with desirable statistical properties which prevents any adversarial attack.
• **Relinearization**: As described in Section 4.4 we benefit significantly from mapping the bit-decomposed polynomials to a three dimensional CUDA kernel wherein the NTT procedure can be invoked in parallel. Once the polynomials are evaluated they need to be reduced back into a single polynomial. To do this efficiently, we launch a parallel reduction kernel with grid dimension roughly equal to the number of bit-decomposed polynomials. After each kernel call, we obtain the partial results in half the polynomials we start with. Repeating this process until we reach a single reduced polynomial makes the relinearization process very efficient with only $\mathcal{O}(\log(\log(q)))$ reduction kernel calls.
• **Streams**: On GPUs we can increase the throughput of kernels by launching them simultaneously on independent streams. For example, NVIDIA K20 has the ability to support upto 32 concurrent kernels launched on a separate stream. In our implementation, we use streams mainly for the parallel generation of noise, synchronizing NTTs, asynchronous memory transfers and kernel parallelization. For taking advantage of stream APIs, we always keep the ciphertext components independent; for example in the Ring-GSW PRE scheme we keep the ciphertext as a vector of column polynomials in row major order.

8.2 Experimental Results

For experimental analysis of our PRE schemes, we use latency and throughput as primary yardsticks. A PRE scheme can be divided into a static and dynamic phase. The static phase takes into account all sorts of key generation, pre-computations and parameter setup. The performance of any real time system that uses a PRE scheme is largely determined by the dynamic phase where computations are performed on the fly. Therefore, we only report the runtimes and throughput for dynamic phase operations by varying the ring dimension, n and modulus bit length, b determined as per 128-bit security setting. For recording throughput, we consider messages as binary string with length equal to the ring dimension.

From Tables 3 and 4 we can observe that encryption and decryption runtimes for the BV-PRE scheme vary in a very small amount with increasing ring dimensions. We remark that encryption runtimes for smaller ring dimensions are still slower with break-even occurring for $n = 2048$ and hence, encryption for smaller ring dimensions are more suitable for CPU platforms. Re-encryption runtime for

PRE Scheme	Parameters		Runtime			Throughput		
	n	b	**Enc (ms)**	**Dec (ms)**	**ReEnc (ms)**	**Enc (Kbps)**	**Dec (Kbps)**	**ReEnc (Kbps)**
BV-PRE	512	18	2.14	0.46	0.29	239.25	1113.04	1765.51
	1024	18	2.18	0.52	0.45	469.72	1969.23	2275.55
	2048	19	2.21	0.55	0.85	926.69	3723.63	2409.41
	4096	19	2.28	0.66	1.26	1796.49	6206.06	3250.79
	8192	20	2.76	1.18	2.76	2968.11	6942.37	2968.11
Ring-GSW-PRE	1024	25	3.92	0.6	39.75	261.22	1706.66	25.76
	2048	26	5.34	0.55	61.18	383.52	3723.63	33.47
	4096	27	9.85	0.63	118.86	415.83	6501.58	34.46
	8192	28	19.89	1.08	259.02	411.86	7585.18	31.62

Table 3: Experimental runtime performance of encryption, decryption, and re-encryption operation for ring dimension n at $r = 1$, $p = 5$, and 128-bits of security. Evaluation data reported for GPU1.

PRE Scheme	Parameters		Runtime			Throughput		
	n	b	**Enc (ms)**	**Dec (ms)**	**ReEnc (ms)**	**Enc (Kbps)**	**Dec (Kbps)**	**ReEnc (Kbps)**
BV-PRE	512	18	4.16	0.47	0.31	123.07	1089.36	1651.61
	1024	18	4.04	0.57	0.4	253.46	1796.49	2560
	2048	19	4.24	0.61	0.69	483.02	3357.37	2968.11
	4096	19	4.3	0.63	0.72	952.55	6501.58	5688.88
	8192	20	4.99	0.68	1.3	1641.68	12047.05	6301.53
Ring-GSW-PRE	1024	25	4.75	0.62	35.46	215.57	1651.61	28.87
	2048	26	5.22	0.67	49.5	392.33	3056.71	41.38
	4096	27	7.98	0.71	87.24	513.28	5769.01	46.95
	8192	28	12.2	0.71	166.67	671.47	11538.03	49.15

Table 4: Experimental runtime performance of encryption, decryption, and re-encryption operation for ring dimension n at $r = 1$, $p = 5$, and 128-bits of security. Evaluation data reported for GPU2.

BV-PRE scheme increases linearly with ring dimensions but still does not grow more than twice as observed on CPU implementations. Comparing our results with Table 6 [22] for re-encryption runtimes, we get a performance improvement by a factor of 39x to 228x. Similarly, we get a peak throughput of 6.3 Mbps for the BV-PRE re-encryption procedure from GPU2.

For the Ring-GSW PRE scheme, decryption runtimes are slightly higher than that of the BV PRE scheme because of relatively large modulus bit lengths. Further, the decryption runtimes vary very little and can be treated as constant for all practical purposes. Re-encryption runtimes are drastically higher and consequently, throughput is reduced when compared to BV PRE re-encryption runtimes and throughput. This is due to the fact that the relinearization procedure is performed over multiple rows of the ciphertext matrix. However, when compared to CPU implementations we still get performance improvements of 3.5x to 11x. Runtimes of both BV PRE and Ring-GSW PRE schemes can be further brought down by considering a larger relinearization window but are accompanied by larger noise growth and error bounds.

9 Conclusion and Future Work

In this work, we explored GPU acceleration of BV PRE and Ring-GSW PRE schemes and showed that GPUs are indeed capable of improving performance by more than an order of magnitude. Moreover, from our experiments, we found that GPUs are more effective in working with larger ring dimensions. We presented several lower level optimizations and parallel NTT implementations tailored specifically for GPU platforms which can be further extended to accelerate other FHE schemes and applications.

In this direction, we would like to support the acceleration of other compute intensive tasks such as bootstrapping LWE FHE schemes, machine learning on encrypted data and other similar privacy concerning applications.

References

1. Albrecht, M.R., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the Complexity of the BKW algorithm on LWE. Designs, Codes and Cryptography **74**(2), 325–354 (2015)
2. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security (TISSEC) **9**(1), 1–30 (2006)
3. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 127–144. Springer (1998)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**, 13 (2014)
5. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Annual cryptology conference. pp. 505–524. Springer (2011)

6. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing **43**(2), 831–871 (2014)
7. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 185–194. ACM (2007)
8. Dai, W., Doröz, Y., Sunar, B.: Accelerating NTRU based homomorphic encryption using GPUs. In: 2014 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2014)
9. Dai, W., Sunar, B.: cuhe: A homomorphic encryption accelerator library. In: International Conference on Cryptography and Information Security in the Balkans. pp. 169–186. Springer (2015)
10. Dhem, J.F., Quisquater, J.J.: Recent results on modular multiplications for smart cards. In: International Conference on Smart Card Research and Advanced Applications. pp. 336–352. Springer (1998)
11. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES Evaluation using NTRU. IACR Cryptology ePrint Archive **2014**, 39 (2014)
12. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Advances in Cryptology–CRYPTO 2013, pp. 75–92. Springer (2013)
14. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. In: Stoc. vol. 9, pp. 169–178 (2009)
15. Halevi, S., Shoup, V.: Algorithms in Helib. In: Annual Cryptology Conference. pp. 554–571. Springer (2014)
16. Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: Scalable homomorphic implementation of encrypted data-classifiers. IEEE Transactions on Computers **65**(9), 2848–2858 (2016)
17. Khurana, H., Heo, J., Pant, M.: From proxy encryption primitives to a deployable secure-mailing-list solution. In: International Conference on Information and Communications Security. pp. 260–281. Springer (2006)
18. Laine, K., Lauter, K.: Key recovery for LWE in polynomial time (2015)
19. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Cryptographers Track at the RSA Conference. pp. 319–339. Springer (2011)
20. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. pp. 1219–1234 (2012)
21. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 700–718. Springer (2012)
22. Polyakov, Y., Rohloff, K., Sahu, G., Vaikuntanathan, V.: Fast Proxy Re-Encryption for Publish/Subscribe Systems. ACM Transactions on Privacy and Security (TOPS) **20**(4), 14 (2017)
23. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 34 (2009)
24. Taban, G., Cárdenas, A.A., Gligor, V.D.: Towards a secure and interoperable DRM architecture. In: Proceedings of the ACM workshop on Digital rights management. pp. 69–78. ACM (2006)