Perspectives of Convolution and Number Theoretic Transform over Prime Power Moduli

Yimeng He[1]

Nanyang Technological University, 50 Nanyang Ave, Singapore
yimeng002@e.ntu.edu.sg

Abstract. Abstract to be done. **(To reference)**

1 Introduction

The convolution problem We are interested in the discrete circular convolution problem where the underlying modulus is a prime power. Specifically, given 2 sequences $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$, $\boldsymbol{b} = (b_0, \ldots, b_{N-1}) \in \mathbb{Z}_{p^m}^N$ for prime power modulus p^m and length N. We aim to efficiently compute their circular convolution product mod p^m:

$$\boldsymbol{c} = \boldsymbol{a} \circledast \boldsymbol{b} \qquad \text{where } c_i = \sum_{i=0}^{N-1} a_i b_{N-i} \,\forall 0 \leq i < N$$

Alternatively, we can recast the problem into one of wrapped polynomial multiplication. Let $f(x) = a_0 + a_1 x + \ldots + a_{N-1} x^{N-1}$, $g(x) = b_0 + b_1 x + \ldots + b_{N-1} x^{N-1} \in \mathbb{Z}_{p^m}[x]$, we want to efficiently calculate the product $h(x) = f(x)g(x) \pmod{p^m, x^N - 1}$.

There is another interpretation of the problem. We define the circulant matrix $\boldsymbol{H} \in \mathbb{Z}_{p^m}^{N \times N}$, $\boldsymbol{H}_{i,j} = a_{j-i \bmod N}$. The goal is to efficiently calculate the matrix vector product $\boldsymbol{c} = \boldsymbol{H} \cdot \boldsymbol{b} \in \mathbb{Z}_{p^m}^N$.

Classical Number Theoretic Transform (NTT) In the special(and important) case where the length $N = 2^n$ and the modulus $p = k2^n + 1$ is a prime, we can find a primitive N^{th} root of unity $\omega \in \mathbb{Z}_p^\times$. The above problem can be handled by the famous radix-2 Cooley-Tuckey algorithm under $O(N \log N)$ time [12].

Let us briefly recall the classical NTT strategy:

1. Find a multiplicative generator $g \in \mathbb{Z}_p^\times$
2. Raise to a power $\omega = g^{\frac{p-1}{N}}$ so that ω is a primitive N^{th} root of unity
3. Use Cooley-Tuckey radix 2 algorithm to compute the fourier transform $\mathsf{FFT}(\boldsymbol{a})_i = \sum a_j \omega^{ij}$, $\mathsf{FFT}(\boldsymbol{b})_i = \sum b_j \omega^{ij}$
4. Perform the elementwise product $\boldsymbol{c}' = \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b})$
5. Finally, use the radix-2 algorithm again to calculate the inverse fourier transform $\boldsymbol{c} = \mathsf{InvFFT}(\boldsymbol{c}')$, $\boldsymbol{c}_i = \frac{1}{N} \sum_{j=0}^{N-1} c'_j \omega^{-ij}$

The more general case Though elegant, classical NTT imposes significant restrictions on the modulus and the convolution length. Since NTT has found important applications in Cryptography, Coding Theory, Communication Theory etc. there is an ongoing research attempting to work around this limitation. The aim is to efficiently compute the circular convolution/NTT under more general modulus and length.

This paper considers this general question and focuses on prime power modulus p^m and arbitrary length. We are mostly interested in a very small prime p: the most interesting case is $p = 2$, since this is the default base in most modern computer architectures.

To the best of our knowledge, one of the most effective, practical and generic solution to the arbirary modulus, arbitrary length problem makes use of multimodular NTT, which is a combination of classical NTT with Residue Number System and Chinese Remainder Theorem. See for example [25, Sect 6]. We refer the reader to the related work section for a brief overview of this strategy.

Our perspectives In the multimodular approach, we must lift both arguments to *integer sequences* and solve the integer circular convolution problem. Apart from bounding the size of the final result, the initial modulus plays almost no role in the multimodular NTT algorithm. This paper explores whether we can use NTT in a modulus-aware fashion. Namely

Is it possible to use NTT over prime power moduli p^m, in such a way that the method is consistent with arithmetic mod p^m*?*

On the positive side, we will show that it is theoretically feasible to adapt NTT to handle prime power moduli and arbitrary (but very restricted) lengths. On the negative side, although our adapted NTT has a theoretical quasi-linear time complexity, a proof-of-concept implementation demonstrates that its concrete efficiency falls behind that of the multimodular NTT method and the efficiency of the state of the art modular polynomial multiplication algorithm.

We conclude that at this stage our idea works in theory, but is not practically efficient. It is an interesting future research direction to explore whether some parts of our strategy might help improve the concrete efficiency of practical solutions to the general NTT problem.

Our method At a high level, our strategy is the same as the classical NTT and its numerous variants. Namely:

1. Find a "suitable" set of roots of unity
2. Pad the arguments to "suitable" lengths
3. Employ "suitable" quasi-linear time algorithms to compute the fourier transform. pointwise multiply the intermediate result and use similar quasi-linear time algorithm to compute the inverse transform

We need a number of less well-known techniques to make the strategy work in our setting. In more detail:

- Section 3 finds and calculates a class of roots of unity compatible with both NTT application and arithmetic of the ring $\mathbb{Z}_{p^m}$. The main idea is to find these roots in the Galois ring extension $\mathsf{GR}(p^m, r)$. For each $r \geq 1$, there is a suitable root of unity of order $p^r - 1$. We will see why such a root is appropriate and provide efficient algorithms to calculate the root.
- In section 4 we briefly recall how to pad arguments so that we can transform a circular convolution problem of length N to one of a more convenient length $N' > N$.
- Efficient convolution and NTT is the main topic of section 5. We suggest 2 algorithms based on factorization characteristics of N'
  - If $N' = q^e$ is a power of a small prime q, we take inspiration from [23] and propose an $O(N \log N)$ recursion algorithm to compute the circular convolution without doing the fourier and inverse fourier transforms.
  - If $N' = N_1 N_2$ where N_1, N_2 are coprime. We employ the Good-Thomas Prime Factorization technique [17,28] and reduce the calculation of length N' fourier transform to 2 consecutive block-wise length N_1 and length N_2 fourier transforms.
  - Finally, we combine the 2 methods above to handle those N' having factorization $N' = q_1 \dots q_{n-1} q_n^e$, where $e \geq 1$ and $q_1, \dots, q_n$ are distinct small primes.
- The last section contains some benchmark results of a proof-of-concept Sagemath/Python implementation of our strategy. We will also discuss bottlenecks and possible improvements of our approach.

Applications NTT has become a cornerstone in modern cryptographic applications, particularly within Post-Quantum Cryptography and Homomorphic Encryption. It plays a vital role in lattice-based cryptography schemes such as Dilithium, Falcon, and Kyber [8,16,7]. In Homomorphic Encryption schemes like BFV, BGV, and CKKS [9,10,11], NTT is also critical for such operations as modulus switching and ciphertext relinearization. However as hinted before, classical NTT imposes severe restrictions on the set of parameters, particularly the existence of suitable N^{th} or $2N^{\text{th}}$ roots of unity (for negacyclic convolution) modulo a prime q. We believe that overcoming the limitation of classical NTT will also open up the range of parameter choices in those schemes and will help us find better and more secure instantiations.

If we focus on the power of 2 modulus, where $p = 2$, our perspective might also have implications in domains such as Coding Theory and Communication. Efficient computation of (linear) circular convolutions is integral to the encoding and decoding procedures of certain codes (see for example [4,14] for cyclic and negacyclic codes). In addition, our result might also testify to the perspective that some operations on finite fields have useful analogues in finite ring settings.

2 Related Works

Classical Fast Fourier Transform (FFT) method was discovered by Gauß, and later rediscovered and popularized by the seminal Cooley-Tukey algorithm [13,12], which uses a divide-and-conquer strategy to reduce the computational complexity from $O(N^2)$ to $O(N \log N)$. Subsequently, the Bluestein FFT algorithm [6] allows efficient Fourier transforms for arbitrary lengths. In a similar vei, the Mixed Radix FFT [2] further supports factorization of input length into arbitrary composite bases.

The analogue of FFT in the realm of modular arithmetic, which is canonically known as Number Theoretic Transform(NTT), operates over fields or rings. Some well known algorithms in this realm are the Nussbaumer transform algorithm [22], Schoenhage-Strassen algorithm [24]. Victor Shoup, for example, introduced the concept of Multimodal-NTT [25] to support efficient modular operations.

The idea of Multimodal-NTT is to combine Chinese remainder theorem with NTT. To compute the circular convolution of 2 integer sequences, suppose the input length N is a power of 2 and all components of the inputs are nonnegative integers no greater than M. Then each component of the result is upper bounded by M^2N. Hence we can find a set of distinct primes $p_1, p_2, \ldots, p_2$ such that

- For each p_i, $\mathbb{Z}_{p_i}$ contains a primitive N^{th} root of unity
- $\prod_{i=1}^{n} p_i \geq M^2N$

We can use the classical NTT algorithm to respecively compute the circular convolution mod p_1, mod $p_2, \ldots,$ mod p_n and obtain the final result mod mod $\prod p_i$ by applying Chinese remainder theorem to each component.

Several works also explored generalizations of NTT to more complex algebraic structures. In particular, Martens and Vanwormhoudt [19] investigated the use of conjugate symmetry in NTTs over regular integer rings. Al Badawi et al. [3] introduced a modified discrete Galois transform tailored for efficient polynomial multiplication in hardware implementations. We hope our work can provide a new incentive for research in this direction.

3 On Roots of Unity

Whether NTT or FFT, all fourier transform related algorithms require certain roots of unity. Suppose we need to compute the fourier transform of N data points, we generally need N distinct roots of unity, but the nature of these roots depends on context.

In scientific computations, data are real/complex numbers in general. The N roots of unity are usually taken to be the evenly-spaced points on the complex unit circle $0 \leq k < N : \zeta^k := \exp\left(\frac{2\pi i k}{N}\right)$.

In discrete computational problems, data are usually finite field/finite domain elements. The often repeated mantra is that the N roots of unity are generated by a *primitive* N^{th} root of unity, an element ζ in the field/domain such that $\zeta^N = 1$, $\forall 0 < k < N : \zeta^k \neq 1$.

In non-integral rings, for example $\mathbb{Z}_{32}, \mathbb{Z}_{24}$, primitivity alone no longer suffices

Example 1. $x^2 - 1 = 0$ has 4 solutions over $\mathbb{Z}_8$: $x = 1, 3, 5, 7 \pmod 8$, among which $3, 5, 7$ are all primitive 2^{nd} roots of unity. Are they suitable for a length-2 NTT? Let us look at the the fourier(Vandermonde) matrix of the root $\zeta = 5$.

$$\boldsymbol{V} = \begin{pmatrix} \zeta^0 & \zeta^0 \\ \zeta^0 & \zeta^1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 5 \end{pmatrix} \qquad \det(\boldsymbol{V}) = 5 - 1 = 4$$

$\boldsymbol{V}$ is not even inverible over $\mathbb{Z}_8$. This is also true for other primitive roots of unity.

On close inspection, the non-invertibility of $\boldsymbol{V}$ is the only obstruction. We conclude that in the context of non-integral rings, the N^{th} root of unity suitable for NTT/FFT application is one for which its corresponding fourier matrix $\boldsymbol{V}$ is invertible.

Definition 1. *(Principal root of unity, adapted from [30]) Let $\mathcal{R}$ be a commutative ring with identity. We call $\zeta \in \mathcal{R}$ a principal N^{th} root of unity if*

1. *N is invertible (equivalently, N is coprime to the ring characterstic $\mathsf{char}(\mathcal{R})$)*
2. *$\zeta^N = 1$*
3. *$\forall 1 \leq k < N : \sum_{i=0}^{N-1} \zeta^{ik} = 0$*

The following proposition is a direct consequence of definition 1.

Proposition 1. *If ζ is a principal N^{th} root of unity, then the fourier matrix $\boldsymbol{V} := \boldsymbol{V}(1, \zeta, \ldots, \zeta^{N-1})$, $\forall 0 \leq i, j < N : \boldsymbol{V}_{i,j} = \zeta^{ij}$ is invertible with inverse $\boldsymbol{V}^{-1} = \frac{1}{N}\boldsymbol{V}^*$, $\boldsymbol{V}^*_{i,j} = \zeta^{-ij}$.*

Let $p \neq 2$ be a prime. A well known result due to Gauß says for any $m \geq 1$, the unit group $\mathbb{Z}_{p^m}^\times \cong \mathcal{C}_{\phi(p^m)} = \mathcal{C}_{p^{m-1}(p-1)}$ is cyclic. In fact, all units in the (unique) order $(p-1)$ subgroup of $\mathbb{Z}_{p^m}^\times$ are principal.

Proposition 2. *Let $\zeta \in \mathbb{Z}_{p^m}^\times$ generates the unique subgroup of order $(p-1)$. Then ζ is a principal $(p-1)^{th}$ root of unity.*

Proof. Let $N = p - 1$. 1 and 2 in definition 1 is obvious. Note that for any $1 \leq k < N$, $(\zeta^k - 1)(1 + \zeta^k + \ldots + \zeta^{k(N-1)}) = \zeta^{Nk} - 1 = 0$. Since $\zeta \bmod p$ is a primitive $(p-1)^{\text{th}}$ root of unity in $\mathbb{Z}_p$, $\zeta^k - 1 \neq 0 \pmod p$. Hence $\zeta^k - 1$ is relatively prime to p^m and is invertible over $\mathbb{Z}_{p^m}$. This proves 3 in definition 1. $\square$

Although $\mathbb{Z}_{p^m}$ contains a principal $(p-1)^{\text{th}}$ root of unity, which is a generator of the unique cyclic subgroup of $\mathbb{Z}_{p^m}^\times$ order $(p-1)$, the problem is that in most cases of interest, the convolution length $N \gg p$ (this holds in particular when $p = 2$). To find large principal roots of unity while still preserving the modular structure forces us to look at extension rings. This is where the Galois Ring comes into our picture.

Galois Rings can be motivated, defined and represented in a number of different ways. We refer the reader to [5,20] for more backgrounds and theories. Here we would like to think of a Galois Ring $\mathsf{GR}(p^m, r)$ as degree r extension of $\mathbb{Z}_{p^m}$ in the same way that the Galois field $\mathbb{F}_{p^r}$ is a degree r extension of $\mathbb{Z}_p$.

Definition 2. *(Galois Ring [29]) The Galois Ring* $\mathsf{GR}(p^m, r)$ *can be represented by a quotient polynomial ring* $\mathbb{Z}[x]/(p^m, f(x))$ *where* $f(x)$ *is a degree* r *monic polynomial which is also irreducible* mod p.

Just like finite fields, all Galois Rings with the same modulus p^m and extension degree r are isomorphic. In some sense $\mathsf{GR}(p^m, r)$ doesn't depend on the particular choice of $f(x)$. However, some $f(x)$ are more convienient from a computational point of view.

In finite field theory, a degree r polynomial $f(x) \in \mathbb{Z}_p[x]$ is called *primitive* if $f(x)$ is monic irreducible and $x \pmod{p, f(x)}$ has order $p^r - 1$ in $\mathbb{F}_{p^r} \cong \mathbb{Z}_p[x]/(f(x))$. In other words, the equivalent class of $[x]$ is a primitive $(p^r - 1)^{\text{th}}$ root of unity.

We would like to find analogues of primitive polynomials over the ring $\mathbb{Z}_{p^m}[x]$. Indeed, using Hensel's lifting technique, we can lift a primitive polynomial $f(x) \in \mathbb{Z}_p[x]$ to $F(x) \in \mathbb{Z}_{p^m}[x]$. We will show that the lifted polynomial has desirable properties.

Theorem 1. *(Hensel Lifting, integral form [32]) Suppose* $f(x) \equiv \alpha_0 g(x)h(x) \pmod{p}$*, where* α_0 *is not divisible by* p *and* $g(x), h(x)$ *are monic polynomials that are coprime* mod p. *Then* $\forall k \geq 1$ *there exist polynomials* $g_k(x), h_k(x) \in \mathbb{Z}[x]$ *unique up to* mod p^k *such that*

1. $g_k(x) \equiv g(x) \pmod{p} \qquad f_k(x) \equiv f(x) \pmod{p}$
2. $f(x) \equiv \alpha_0 g_k(x) f_k(x) \pmod{p^k}$

Proposition 3. *Let* $f(x)$ *be a primitive polynomial* mod p *of degree* r*, there exists a monic polynomial* $g(x)$ *unique up to* mod p *such that*

$$x^{p^r-1} - 1 \equiv f(x)g(x) \pmod{p}$$

Now apply theorem 1 Hensel lifting to the equation above. We can find a unique polynomial $f_m(x) \in \mathbb{Z}_{p^m}[x]$ *such that* $f_m(x) \equiv f(x) \pmod{p}$ *and* $f_m(x) \mid x^{p^r-1} - 1$ *over* $\mathbb{Z}_{p^m}[x]$.

Let the Galois Ring be defined over this polynomial $\mathsf{GR}(p^m, r) \cong \mathbb{Z}[x]/(p^m, f_m(x))$. *We claim that:*

1. *The equivalent class* $x \pmod{p^m, f_m(x)}$ *has order* $p^r - 1$ *over* $\mathsf{GR}(p^m, r)^\times$. *Moreover,*
2. *The equivalent class* $x \pmod{p^m, f_m(x)}$ *is a principal* $(p^r - 1)^{th}$ *root of unity over* $\mathsf{GR}(p^m, r)$. *Hence*
3. *For any* $N \mid p^r - 1$*, the equivalent class* $x^{\frac{p^r-1}{N}} \pmod{p^m, f_m(x)}$ *is a principal* N^{th} *root of unity.*

Proof. 1. Since $f_m(x) \mid x^{p^r-1} - 1$ over $\mathbb{Z}_{p^m}[x]$, $x^{p^r-1} \equiv 1 \pmod{p^m, f_m(x)}$. Suppose there exists a $0 < k < p^r - 1$ such that $x^k \equiv 1 \pmod{p^m, f_m(x)}$. Since $f_m(x) \equiv f(x) \pmod{p}$, we can reduce mod p and obtain $x^k \equiv 1 \pmod{p, f(x)}$, contradiction to the fact that $f(x)$ is a primitive polynomial mod p.

2. The only nontrivial part to verify is condition 3 in definition 1. Let $N = p^r - 1$ and fix $0 < k < N$. It is easy to see that $(x^k - 1)(\sum_{i=0}^{N-1} x^{ik}) = x^{kN} - 1 \equiv 0 \pmod{p^m, f_m(x)}$. We claim that $x^k - 1 \pmod{p^m, f_m(x)}$ has a multiplicative inverse over $\mathbb{Z}_{p^m}[x]$. If the claim is true, we can multiply the inverse and obtain $\sum x^{ik} \equiv 0 \pmod{p^m, f_m(x)}$. The equivalent class $[x]$ is therefore a principal $(p^r - 1)^{\text{th}}$ root of unity.

Claim: $\exists h(x) : (x^k - 1)h(x) \equiv 1 \pmod{p^m, f_m(x)}$

proof of Claim. Since $0 < k < N$, $x^k - 1 \pmod{p, f(x)}$ is nonzero and has a multiplicative inverse $h_1(x)$, i.e., $(x^k - 1)h_1(x) \equiv 1 \pmod{p, f_m(x)}$ (recall that $\mathbb{Z}[x]/(p, f(x)) = \mathbb{Z}[x]/(p, f_m(x))$ is a field). We are going to lift the inverse mod p to one mod p^m. To do so it is most convenient to employ a technique known as Newton-Raphson division [31]

Newton-Rhaphson division: Let $l > 0$. Suppose $\exists h_l(x)$ s.t. $(x^k - 1)h_l(x) \equiv 1 \pmod{p^l, f_m(x)}$. Define

$$h_{l+1}(x) = 2h_l(x) - (x^k - 1)h_l(x)^2$$

Then $(x^k-1)h_{l+1}(x) \equiv 1 \pmod{p^{l+1}, f_m(x)}$

proof of lifting. Write $(x^k-1)h_l(x) = 1 = p^l M_l(x) + f_m(x)N_l(x)$ for some polynomials $M_l(x), N_l(x)$. Then

$$\begin{aligned}
&(x^k-1)^2 h_l(x)^2 = 1 + 2p^l M_l(x) + p^{l+1}(p^{l-1}M_l(x)^2) + f_m(x)\big(f_m(x)N_l(x)^2 \\
&+2N_l(x)(1+p^l M_l(x))\big) \\
&2(x^k-1)h_l(x) = 2 + 2p^l M_l(x) + f_m(x)N_l(x) \\
\Longrightarrow\, &(x^k-1)h_{l+1}(x) = 1 + p^{l+1}M_{l+1}(x) + f_m(x)N_{l+1}(x)
\end{aligned}$$

for some polynomials $M_{l+1}(x), N_{l+1}(x)$. Therefore we can use Newton-Rhaphson division to lift the inverse up to mod p^m. This shows that x^k-1 and the claim is proven.

3. is a straightforward consequence of 2. □

3.1 On Efficient Lifting

We need o address an uncomfortable algorithmic issue before we proceed to the next section. Traditionally, Hensel Lifting is usually accomplished by starting with a coprime factoization $f(x) = g(x)h(x) \pmod{p}$ and use inexpensive polynomial GCD operations to lift to the factorization $f(x) = f_m(x)g_m(x) \pmod{p^m}$. Our trouble is that both $x^{p^r-1}-1$ and $x^{p^r-1}-1/f(x)$ are too big even for moderately large r (We tried to key in $x^{2^{32}-1}-1$ into the Sagemath/Python but the system complains and refuses to digest it) .

Fortunately, there is a way to only lift the primitive polynomial $f(x)$ without knowing or caring about its coprime factor. The method we use comes from [21], but it may already be described and used in other contexts.

Proposition 4. *(Adapted from [21] Theorem 1) Let $k > 0$ and suppose a monic polynomial $f_k(x) \in \mathbb{Z}[x]$ satisfies:*

- $f_k(x) \bmod p^k$ *is irreducible over* $\mathbb{Z}_{p^k}[x]$
- $\exists M$ *coprime to p such that* $f_k(x) \mid x^M - 1$ *over* $\mathbb{Z}_{p^k}[x]$

We let $f_{k+1}(x)$ be a monic polynomial whose roots are the p^{th} power of the roots of $f_k(x)$ over an algebraically closed field. f_{k+1} satisfies the following properties:

1. $f_{k+1}(x) \in \mathbb{Z}[x]$ *is integral*
2. $f_{k+1}(x) \equiv f_k(x) \pmod{p^k}$, *hence $f_{k+1}(x)$ is irreducible over* $\mathbb{Z}_{p^{k+1}}[x]$
3. $f_{k+1}(x) \mid x^M - 1$ *over* $\mathbb{Z}_{p^{k+1}}[x]$

In otherwords, the Hensel Lifting of $f_k(x) \bmod p^k$ is exactly $f_{k+1}(x) \bmod p^{k+1}$

Proof. For the proof we are going to use some p-adic theory. Let $\mathcal{Z}_p$ be the ring of p-adic integers and $\mathcal{Q}_p$ the field of p-adic rationals. Since M is coprime to p the polynomial x^M-1 has M distinct roots of unity over an extension field of $\mathcal{Q}_p$.

By Hensel Lifting Lemma theorem 1, there exists $f_{k+1}(x), g(x) \in \mathbb{Z}[x]$ such that $f_{k+1}(x) \mid x^M - 1$ over $\mathbb{Z}_{p^{k+1}}[x]$ and $f_{k+1}(x) \equiv f_k(x) + p^k g(x) \pmod{p^{k+1}}$. Therefore item 1 and 2 of proposition 4 are immediate, and it remains to show item 3.

Moreover, if α_k is a root of $f_k(x)$ over $\mathbb{Z}_{p^k}$, namely $f_k(\alpha_k) \equiv 0 \pmod{p^k}$. There would exist a root $\alpha_{k+1} = \alpha_k + p^k\delta$, where δ lies in an extension field of $\mathcal{Q}_p$, of $f_{k+1}(x)$ over $\mathbb{Z}_{p^{k+1}}$. In other words, $f_{k+1}(\alpha_{k+1}) \equiv 0 \pmod{p^{k+1}}$.

Since $f_k(x) \mid x^M - 1$ over $\mathbb{Z}_{p^k}[x]$, there exists an ϵ in an extension field of $\mathcal{Q}_p$ such that $\alpha_k^M = 1 + p^k\epsilon$. Because

$$\begin{aligned}
\alpha_{k+1}^p &= (\alpha_k + p^k\delta)^p = \alpha_k^p + O(p^{k+1}) \equiv \alpha_k^p \pmod{p^{k+1}} \\
\alpha_{k+1}^{pM} &= (\alpha_k + p^k\delta)^{pM} = \alpha_k^{pM} + O(p^{k+1}) = (1+p^k\epsilon)^p + O(p^{k+1}) = 1 + O(p^{k+1}) \equiv 1 \pmod{p^{k+1}}
\end{aligned}$$

Hence the p^{th} power of distinct roots α_k of $f_k(x)$ over $\mathbb{Z}_{p^k}$ are all distinct roots of $x^M - 1$ over $\mathbb{Z}_{p^{k+1}}$. In addition, $\alpha_{k+1}^p \equiv \alpha_k^p \equiv \alpha_k \pmod{p}$, and we must therefore have $f_k(\alpha_k^p) \equiv 0 \pmod{p^k}$.

The roots of f_{k+1} over $\mathbb{Z}_{p^{k+1}}$ are, up to mod p^{k+1} the p^{th} power of all the roots of $f_k(x)$ over $\mathbb{Z}_{p^k}$. Therefore item 3 is proven. □

Similar to GCD operations, we can in fact calculate the lifting using operations over the ground ring $\mathbb{Z}$. This can be done with polynomial resultants.

Definition 3. *(Univariate Resultant [33]) Let $f(x) = a_0 + a_1x + \ldots + a_nx^n$, $g(x) = b_0 + b_1x + \ldots + b_mx^m$ where $a_n, b_m \neq 0$. The resultant of f, g is the determinant of the $(m+n) \times (m+n)$ Sylvester matrix:*

$$\mathsf{Res}(f,g) := \det \begin{pmatrix}
a_0 & a_1 & \cdots & a_n & 0 & \cdots & 0 \\
0 & a_0 & a_1 & \cdots & a_n & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & a_0 & a_1 & \cdots & a_n \\
b_0 & b_1 & \cdots & b_m & 0 & \cdots & 0 \\
0 & b_0 & b_1 & \cdots & b_m & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & b_0 & b_1 & \cdots & b_m
\end{pmatrix}$$

In particular, if $(\alpha_i)_{i=1}^n$, $(\beta_j)_{j=1}^m$ are the roots of f, g in some extension field. Then

$$\mathsf{Res}(f,g) = a_0^m \prod_{i=1}^n g(\alpha_i) = (-1)^{mn} b_0^n \prod_{j=1}^m f(\beta_j)$$

Proposition 5. *Let $d > 0$, $f(x) \in \mathbb{Z}[x]$ be a degree n monic polynomial, $g(x) = \mathsf{Res}_y(x - y^d, f(y))$ the resultant of $x - y^d, f(y)$ considered as polynomials in y with coefficients in $\mathbb{Z}[x]$. Then $g(x) \in \mathbb{Z}[x]$ with leading coefficient ± 1. Moreover, the roots of $g(x)$ are exactly the d^{th} power of all the roots of $f(x)$ over an algebraically closed field.*

Proof. Let $\mathcal{K}$ be an algebraically closed field containing $\mathbb{Q}(x)$. Let $\beta_0, \ldots, \beta_{n-1} \in \mathcal{K}$ be the roots of $f(y)$ and $\zeta \in \mathcal{K}$ a primitive d^{th} root of unity. We have

$$x - y^d = \prod_{i=0}^{d-1} (x^{\frac{1}{d}} - \zeta^i y), \qquad f(y) = \prod_{i=0}^{n-1} (y - \beta_i)$$

Therefore the product over root property in definition 3 asserts that

$$\begin{aligned}
&\mathsf{Res}_y(x-y^d, f(y)) = (-1)^n \prod_{i=0}^{d-1} f(\zeta^{d-i}x^{\frac{1}{d}}) = (-1)^n \prod_{i=0}^{d-1}\prod_{j=0}^{n-1}(\zeta^{d-i}x^{\frac{1}{d}} - \beta_j)\\
&= (-1)^n \prod_{j=0}^{n-1}\left(\prod_{i=0}^{d-1}\zeta^{d-i}\right)\prod_{i=0}^{d-1}(x^{\frac{1}{d}} - \zeta^i\beta_j) = (-1)^n\zeta^{\frac{d(d-1)n}{2}}\prod_{j=0}^{n-1}(x-\beta_j^d)\\
&= \pm\prod_{j=0}^{n-1}(x-\beta_j^d)
\end{aligned}$$

Therefore, up to sign, $g(x)$ is the monic polynomial whose roots are exactly the d^{th} power of the roots of $f(x)$. $g(x) \in \mathbb{Z}[x]$ since the coefficients of $x - y^n, f(y)$ all belong to $\mathbb{Z}[x]$ $\square$

Remark 1. The Sagemath library contains a method called adams_operator_on_roots that raises the roots of a polynomial to any degree. We could not identify the origin of this name, but upon inspection on the source code, we saw that the implementation computes the resultant polynomial as in proposition 5.

We can combine propositions 4 and 5 to construct an efficient algorithm to lift primitive polynomials.

Algorithm 1 Hensel Lift Primitive Polynomial

```
Input: Monic degree $r$ primitive polynomial $f(x) \in \mathbb{Z}_p[x]$ and exponent $m$
Output: The Hensel Lifted monic polynomial $f_m(x) \in \mathbb{Z}_{p^m}[x]$
1: Let $f_1(x) = f(x)$ and regard $f_1(x) \in \mathbb{Z}[x]$
2: for $k \leftarrow 2$ to $m$ do
3:     Calculate $g(x) = \mathsf{Res}_y(x - y^p, f_{k-1}(y))$ and normalize $g(x)$ to be monic
4:     Let $f_k(x) = g(x) \bmod p^k$
5: end for
6: Return $f_m(x)$
```

Remark 2. Alternatively, we may simply calculate $g(x) = \mathsf{Res}_y(x - y^{p^m}, f(y))$ and directly obtain $f_m(x) = g(x) \bmod p^m$. The difference is mainly that of efficiency.

In algorithm 1 we calculate $m-1$ determinants whose matrices are of dimension $p + r - 1$ whereas here we calculate 1 determinant whose matrix is of dimension $p^m + r - 1$

Remark 3. When $p = 2$, we may use the following simplified procedure inside the for loop of algorithm 1:

- 3(b). Calculate $G(x) = f_{k-1}(x)f_{k-1}(-x)$. Replace x^2 by x to obtain $g(x) = G(\sqrt{x})$. Normalize $g(x)$ to be monic.
- 4(b). Let $f_k(x) = g(x) \bmod 2^k$

Example 2. Let $p = 2$, $f(x) = x^5 + x^2 + 1$ is a monic primitive polynomial over $\mathbb{Z}_2[x]$. According to algorithm 1 and the previous remark

$$\begin{aligned}
f_1(x)f_1(-x) &= -x^{10} + x^4 + 2x^2 + 1\\
f_2(x) &= x^5 + 3x^2 + 2x + 3\\
f_2(x)f_2(-x) &= -x^{10} - 4x^6 + 9x^4 + 14x^2 + 9\\
f_3(x) &= x^5 + 4x^3 + 7x^2 + 2x + 7\\
f_3(x)f_3(-x) &= -x^{10} - 8x^8 - 20x^6 + 33x^4 + 94x^2 + 49\\
f_4(x) &= x^5 + 8x^4 + 4x^3 + 15x^2 + 2x + 15
\end{aligned}$$

We can verify that the class $[x]$ is a principal $2^5 - 1 = 31^{\text{th}}$ root of unity in the Galois Ring $\mathsf{GR}(2^4, 5) \cong \mathbb{Z}[x]/(16, f_4(x))$

Example 3. Let $p = 3$, $f(x) = x^5 + 2x + 1$ is a monic primitive polynomial over $\mathbb{Z}_3[x]$. According to algorithm 1

$$\begin{aligned}
\mathsf{Res}_y(x - y^3, f(y)) &= -x^5 + 6x^2 - 8x - 1 \\
f_2(x) &= x^5 + 3x^2 + 8x + 1 \\
\mathsf{Res}_y(x - y^3, f_2(y)) &= -x^5 - 9x^4 - 27x^3 - 3x^2 - 440x - 1 \\
f_3(x) &= x^5 + 9x^4 + 3x^2 + 8x + 1 \\
\mathsf{Res}_y(x - y^3, f_2(y)) &= -x^5 - 738x^4 - 1944x^3 - 1650x^2 - 440x - 1 \\
f_4(x) &= x^5 + 9x^4 + 30x^2 + 35x + 1
\end{aligned}$$

We can easily verify that the equivalent class $[x]$ is a principal $3^5 - 1 = 242^{\text{th}}$ root of unity in the Galois Ring $\mathsf{GR}(3^4, 5) \cong \mathbb{Z}[x]/(81, f_4(x))$

4 Padding to Better Lengths

NTT/FFT related algorithms usually work best, or only work, over special lengths. Given arbitrary length argument, we usually need to 0-pad them to longer sequences to apply these algorithms. In this paper we will adopt the following simple padding strategy.

Proposition 6. *Let $\boldsymbol{u} = (u_0, \dots, u_{N-1})$, $\boldsymbol{v} = (v_0, \dots, v_{N-1})$ be 2 sequences of length N. For any $M \geq 2N - 1$, we can reduce the length N circular convolution problem to the length M circular convolution problem using the following padding strategy:*

- *Let $\Delta_1 = M - 2N + 1$ and let $\boldsymbol{u}' = \boldsymbol{u} \parallel \boldsymbol{0}^{\Delta_1} \parallel (u_1, \dots, u_{N-1})$*
- *Let $\Delta_2 = M - N$ and let $\boldsymbol{v}' = \boldsymbol{v} \parallel \boldsymbol{0}^{\Delta_2}$*
- *Calculate the length M circular convolution $\boldsymbol{w}' = \boldsymbol{u}' \circledast \boldsymbol{v}'$ and retain only the first N result $\boldsymbol{w} = ({w'}_0, \dots, {w'}_{N-1})$*

Proof. For any $0 \leq i < N$:

$$\begin{aligned}
w'_i &= \sum_{j=0}^{M-1} u'_j v'_{i-j} = \sum_{j=0}^{M-1} v'_j u'_{i-j} = \sum_{j=0}^{N-1} b_j a'_{i-j} \\
&= \sum_{j=0}^{i} v_j u_{i-j} + \sum_{j=i+1}^{N-1} v_j u_{M-(j-i)} = \sum_{j=0}^{i} v_j u_{i-j} + \sum_{j=i+1}^{N-1} v_j u_{N-(j-i)} \\
&= \sum_{j=0}^{N-1} v_j u_{i-j} = w_i
\end{aligned}$$

□

Example 4. Suppose $N = 3$ and $\boldsymbol{u} = (1, 2, 3)$, $\boldsymbol{v} = (-1, 0, 1)$. To calculate their circular convolution it is convenient to pad the inputs to length $M = 8 > 2N - 1$. According to proposition 6

$$\boldsymbol{u}' = (1, 2, 3, 0, 0, 0, 2, 3), \qquad \boldsymbol{v}' = (-1, 0, 1, 0, 0, 0, 0, 0)$$

We then calculate the circular convolution of $\boldsymbol{u}', \boldsymbol{v}'$, or equivalently calculate

$$\begin{bmatrix} 1 & 3 & 2 & 0 & 0 & 0 & 3 & 2 \\ 2 & 1 & 3 & 2 & 0 & 0 & 0 & 3 \\ 3 & 2 & 1 & 3 & 2 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 3 & 2 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 2 & 1 & 3 & 2 \\ 2 & 0 & 0 & 0 & 3 & 2 & 1 & 3 \\ 3 & 2 & 0 & 0 & 0 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

And only retain the first 3 components

5 Efficient NTT and Convolution

Now that we have a large enough principal root of unity, and a way to pad inputs to arbitrary greater lengths, it seems like we just need to plug in an off-the-shelf NTT/FFT algorithm and finish our job. Unfortunately, the very limited roots of unity in our setting severely limit the scope of applicable algorithms.

Example 5. Let $p = 2$. Proposition 3 guarantees principal roots of unity of order $N \mid 2^r - 1$. But N is then always odd, hence the famous Cooley-Tuckey radix 2 algorithm [12], split radix 4 algorithm [15], as well as Bluestein FFT algorithm [6] cannot be applied here.

Example 6. Let $p = 2$ and $N = 1000$. We could instead use a radix q Cooley-Tuckey algorithm, for instance $q = 3$. The smallest power of $3 \geq 2N - 1$ is $M = 3^7 = 2187$, and the smallest r such that $M \mid 2^r - 1$ is the multiplicative order of $2 \bmod 3^7$ and equals $r = 1458$. The huge polynomial degree clearly renders the computation impractical.

These 2 examples suggest that a pure-radix strategy is not generally applicable or practical, and therefore we need to incorporate mixed radix strategy. Our method is the combination of the following 2 algorithms.

1. The first algorithm handles the case of prime power length. Here we depart from FFT methodology and consider another $O(N \log N)$ algorithm that computes the circular convolution in 1 pass. Our algorithm is a slightly generalized version found in [23] that handles arbitrary radix.
2. The second algorithm is essentially the Good-Thomas Prime Factorization algorithm [17,28]. If the length $N = N_1 N_2$ can be factored into coprime factors, this method reduces the computation of length N fourier transform to computations of block-wise fourier transforms of length N_1 and N_2

In section 5.3 the 2 algorithms above are combined to handle smooth convolution length $N = q_1 q_2 \dots q_{n-1} q_n^e$ where $q_1, \dots, q_n$ are small primes and $e \geq 1$.

5.1 Prime Power Case

First we recall the notion of an f-circulant matrix.

Definition 4. *f-circulant matrix Let $(a_0, \ldots, a_{N-1})$ be a sequence and f a number. The f-circulant matrix associated with $(a_0, \ldots, a_{N-1})$ is an $N \times N$ matrix $\boldsymbol{A}$ satisfying:*

$$\forall 0 \le i, j < N : \ \boldsymbol{A}_{i,j} = \begin{cases} a_{j-i} & \textit{if } i \le j \\ f a_{N+j-i} & \textit{otherwise} \end{cases}$$

The 1-circulant matrix is just a circulant matrix, while a (-1)-circulant matrix is usually called a nega-cyclic circulant matrix.

We first present the simple and elegant method described in [23]

Theorem 2. *[23] Let $\mathcal{R}$ be a commutative ring with identity, N a power of 2 and $f \in \mathcal{R}$. Suppose*

- *$\mathcal{R}$ contains an N^{th} root of unity and an N^{th} root of f*
- *$2, f$ both have multiplicative inverse in $\mathcal{R}$*

Then the multiplication of an f-circulant matrix $\boldsymbol{A} \in \mathcal{R}^{N \times N}$ and a vector $\boldsymbol{b} \in \mathcal{R}^N$ can be done with $O(N \log N)$ ring operations using algorithm 2.

Algorithm 2 CircMatMult: Multiply an f-circulant matrix with a vector

```
Input: $f$-circulant matrix $\boldsymbol{A}$, input vector $\boldsymbol{b}$, $f$ and length $N$
Output: product $\boldsymbol{A} \cdot \boldsymbol{b}$
1: if $N = 2$ then                                                        ▷ Base Case
2:     Return $\boldsymbol{A} \cdot \boldsymbol{b}$
3: end if
4: Decompose $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ f\boldsymbol{A}_2 & \boldsymbol{A}_1 \end{pmatrix}$ where $\boldsymbol{A_1}, \boldsymbol{A_2} \in \mathcal{R}^{N/2 \times N/2}$. Parse $\boldsymbol{b} = (\boldsymbol{b}_1 \parallel \boldsymbol{b}_2)^\top$ where $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathcal{R}^{N/2}$
5: Calculate
       $\boldsymbol{M}_1 = \boldsymbol{A}_1 + \sqrt{f}\boldsymbol{A}_2 \in \mathcal{R}^{N/2 \times N/2}$
       $\boldsymbol{M}_2 = \boldsymbol{A}_1 - \sqrt{f}\boldsymbol{A}_2 \in \mathcal{R}^{N/2 \times N/2}$
       $\boldsymbol{d}_1 = \sqrt{f}\boldsymbol{b}_1 + \boldsymbol{b}_2 \in \mathcal{R}^{N/2}$
       $\boldsymbol{d}_2 = \sqrt{f}\boldsymbol{b}_1 - \boldsymbol{b}_2 \in \mathcal{R}^{N/2}$
6: Recursively calculate
       $\boldsymbol{e}_1 = \text{CircMatMult}(\boldsymbol{M}_1, \boldsymbol{d}_1, \sqrt{f}, \frac{N}{2})$
       $\boldsymbol{e}_2 = \text{CircMatMult}(\boldsymbol{M}_2, \boldsymbol{d}_2, -\sqrt{f}, \frac{N}{2})$
7: Compute $\boldsymbol{c}_1 = \frac{\boldsymbol{e}_1 + \boldsymbol{e}_2}{2\sqrt{f}}$, $\boldsymbol{c}_2 = \frac{\boldsymbol{e}_1 + \boldsymbol{e}_2}{2} - \boldsymbol{e}_2$
8: Return $\boldsymbol{c} = (\boldsymbol{c}_1 \parallel \boldsymbol{c}_2)^\top$
```

Proof. On one hand, $\begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ f\boldsymbol{A}_2 & \boldsymbol{A}_1 \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_1\boldsymbol{b}_1 + \boldsymbol{A}_2\boldsymbol{b}_2 \\ \boldsymbol{A}_1\boldsymbol{b}_2 + f\boldsymbol{A}_2\boldsymbol{b}_1 \end{pmatrix}$

Assume the recursion step of the algorithm gives the correct result. Then

$$\boldsymbol{e}_1 = \boldsymbol{M}_1\boldsymbol{d}_1 = \sqrt{f}(\boldsymbol{A}_1\boldsymbol{b}_1 + \boldsymbol{A}_2\boldsymbol{b}_2) + (\boldsymbol{A}_1\boldsymbol{b}_2 + f\boldsymbol{A}_2\boldsymbol{b}_1)$$
$$\boldsymbol{e}_2 = \boldsymbol{M}_2\boldsymbol{d}_2 = \sqrt{f}(\boldsymbol{A}_1\boldsymbol{b}_1 + \boldsymbol{A}_2\boldsymbol{b}_2) - (\boldsymbol{A}_1\boldsymbol{b}_2 + f\boldsymbol{A}_2\boldsymbol{b}_1)$$

Hence the final result is correct.

Next we claim that $\boldsymbol{M}_1, \boldsymbol{M_2}$ are respectively $\sqrt{f}, -\sqrt{f}$-circulant matrices. This ensures that the recursion step is correct. Below we prove this fact for $\boldsymbol{M}_1$.

Since $\boldsymbol{A}$ is f-circulant, $\exists (a)_{i=0}^{N-1}$ such that $\boldsymbol{A}_{i,j} = \begin{cases} a_{j-i} & \text{if } i \le j \\ f a_{N+j-i} & \text{otherwise} \end{cases}$.

For $0 \le i < \frac{N}{2}$, define $\alpha_i = a_i + \sqrt{f} a_{\frac{N}{2}+i}$.

If $0 \le i \le j < \frac{N}{2}$:

$$\begin{aligned}(\boldsymbol{M}_1)_{i,j} &= (\boldsymbol{A}_1)_{i,j} + \sqrt{f}(\boldsymbol{A}_2)_{i,j} = \boldsymbol{A}_{i,j} + \sqrt{f}\boldsymbol{A}_{i,j+\frac{N}{2}} \\ &= a_{j-i} + \sqrt{f} a_{j+\frac{N}{2}-i} = \alpha_{j-i}\end{aligned}$$

If $0 \le j < i < \frac{N}{2}$:

$$\begin{aligned}(\boldsymbol{M}_1)_{i,j} &= \boldsymbol{A}_{i,j} + \sqrt{f}\boldsymbol{A}_{i,j+\frac{N}{2}} = f a_{N+j-i} + \sqrt{f} a_{j+\frac{N}{2}-i} \\ &= \sqrt{f}(a_{\frac{N}{2}+j-i} + \sqrt{f} a_{\frac{N}{2}+\frac{N}{2}+j-i}) = \sqrt{f}\alpha_{\frac{N}{2}+j-i}\end{aligned}$$

Hence $\boldsymbol{M}_1$ is $\sqrt{f}$ circulant. The case of $\boldsymbol{M}_2$ follows similarly.

Finally, the algorithm follows the classical divide and conquer qpproach and hence has time complexity $O(N \log N)$. □

We now propose a generalized version of the algorithm described in [23]. Ours support any radix B that satisfies the condition in the following theorem.

Theorem 3. *Let $\mathcal{R}$ be a commutative ring with identity, $B > 0$ and N a power of B. Let $f \in \mathcal{R}$. Suppose*

- *$\mathcal{R}$ contains an N^{th} root of unity and an N^{th} root of f*
- *B, f both have multiplicative inverse in $\mathcal{R}$*

Then the multiplication of an f-circulant matrix $\boldsymbol{A} \in \mathcal{R}^{N \times N}$ and a vector $\boldsymbol{b} \in \mathcal{R}^N$ can be done with $O(N \log N)$ ring operations using algorithm 3.

Proof. Assume the recursion step gives the correct result, then $\forall 0 \le i < B$:

$$\begin{aligned}\boldsymbol{c}_i &= \left(Br^{B-1-i}\right)^{-1} \sum_{j=0}^{B-1} \omega^{ij} \boldsymbol{e}_j \\ &= \left(Br^{B-1-i}\right)^{-1} \sum_{j=0}^{B-1} \left(\omega^{ij} \left(\sum_{k=0}^{B-1} r^k \omega^{kj} \boldsymbol{A}_k\right)\left(\sum_{l=0}^{B-1} r^{B-1-l}\omega^{-lj}\boldsymbol{b}_l\right)\right) \\ &= B^{-1} \sum_{j,k,l} r^{i+k-l} \omega^{j(i+k-l)} \boldsymbol{A}_k \boldsymbol{b}_l = \sum_{k,l} r^{i+k-l} \boldsymbol{A}_k \boldsymbol{b}_l \cdot B^{-1} \sum_j \omega^{j(i+k-l)} \\ &= \sum_{k,l} r^{i+k-l} \boldsymbol{A}_k \boldsymbol{b}_l \cdot [l \equiv i+k \pmod B] = \sum_{k=0}^{B-i-1} \boldsymbol{A}_k \boldsymbol{b}_{i+k} + \sum_{k=B-i}^{B-1} f \boldsymbol{A}_k \boldsymbol{b}_{i+k-B}\end{aligned}$$

The final expression is exactly the i^{th} block of the product $\boldsymbol{Ab}$

Next we show that $\forall 0 \le k < B$, the matrix $\boldsymbol{M}_k$ is a $\omega^k r$-circulant $\frac{N}{B} \times \frac{N}{B}$ matrix.

Because $\boldsymbol{A}$ is f-circulant, $\exists (a)_{i=0}^{N-1}$ such that $\boldsymbol{A}_{i,j} = \begin{cases} a_{j-i} & \text{if } i \le j \\ f a_{N+j-i} & \text{otherwise} \end{cases}$.

For $0 \le i < \frac{N}{B}$, define $\alpha_i = \sum_{j=0}^{B-1} \omega^{jk} r^j a_{\frac{N}{B}j+i}$.

Algorithm 3 GenCircMatMult: Multiply an f-circulant matrix with a vector

```
Input: f-circulant matrix $\boldsymbol{A}$, input vector $\boldsymbol{b}$, $f$ and length $N$
Output: product $\boldsymbol{A} \cdot \boldsymbol{b}$
 1: if $N = B$ then                                                ▷ Base Case
 2:     Return $\boldsymbol{A} \cdot \boldsymbol{b}$
 3: end if
 4: Decompose $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_1 & \dots & \boldsymbol{A}_{B-1} \\ f\boldsymbol{A}_{B-1} & \boldsymbol{A}_0 & \dots & \boldsymbol{A}_{B-2} \\ & & \vdots & \\ f\boldsymbol{A}_1 & f\boldsymbol{A}_2 & \dots & \boldsymbol{A}_0 \end{pmatrix}$ where $\boldsymbol{A_i} \in \mathcal{R}^{N/B \times N/B}$. Parse $\boldsymbol{b} = (\boldsymbol{b}_0 \parallel \dots \parallel \boldsymbol{b}_{B-1})^\top$
    where $\boldsymbol{b}_i \in \mathcal{R}^{N/B}$
 5: Let $\omega$ be a primitive $B^{\text{th}}$ root of unity, $r = f^{1/B}$ a $B^{\text{th}}$ root of $f$
 6: for $i \leftarrow 0$ to $B-1$ do
 7:     Compute
            $$\boldsymbol{M}_i = \sum_{j=0}^{B-1} r^j \omega^{ij} \boldsymbol{A}_j, \qquad \boldsymbol{d}_i = \sum_{j=0}^{B-1} r^{B-1-j} \omega^{-ij} \boldsymbol{b}_j$$
 8:     Recursively compute $\boldsymbol{e}_i = \text{GenCircMatMult}(\boldsymbol{M}_i, \boldsymbol{d}_i, r\omega^i, \frac{N}{B})$
 9: end for
10: for $i \leftarrow 0$ to $B-1$ do
11:     Calculate $\boldsymbol{c}_i = \left(Br^{B-1-i}\right)^{-1} \sum_{j=0}^{B-1} \omega^{ij} \boldsymbol{e}_j$
12: end for
13: Return $\boldsymbol{c} = (\boldsymbol{c}_0 \parallel \dots \parallel \boldsymbol{c}_{B-1})^\top$
```

If $0 \le i \le j < \frac{N}{B}$:

$$(\boldsymbol{M}_k)_{i,j} = \sum_{l=0}^{B-1} r^l \omega^{kl} (\boldsymbol{A}_l)_{i,j} = \sum_{l=0}^{B-1} r^l \omega^{kl} \boldsymbol{A}_{i,\frac{N}{B}l+j} = \sum_{l=0}^{B-1} r^l \omega^{kl} a_{\frac{N}{B}l+j-i} = \alpha_{j-i}$$

If $0 \le j < i < \frac{N}{B}$:

$$\begin{aligned}(\boldsymbol{M}_k)_{i,j} &= \sum_{l=0}^{B-1} r^l \omega^{kl} \boldsymbol{A}_{i,\frac{N}{B}l+j} = f a_{N+j-i} + \sum_{l=1}^{B-1} r^l \omega^{kl} a_{\frac{N}{B}l+j-i} \\ &= \omega^k r \left(\sum_{l=0}^{B-2} r^l \omega^{kl} a_{\frac{N}{B}+\frac{N}{B}l+j-i} \right) + \omega^k r \left(r^{B-1} \omega^{k(B-1) a_{\frac{N}{B}+\frac{(B-1)N}{B}+j-i}} \right) \\ &= \omega^k r \left(\alpha_{\frac{N}{B}+j-i} \right)\end{aligned}$$

Therefore $\boldsymbol{M}_k$ is an $\omega^k r$-circulant matrix.

Finally, the divide an conquer nature of the algorithm implies that the time complexity of algorithm 3 is $O(N \log N)$. □

Example 7. We illustrate the recursion step of our algorithm with an example. Let $B = 3$, ω be a primitive 3rd root of unity and $r = \sqrt[3]{f}$ a third root of f. To calculate the product

$$\begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ f\boldsymbol{A}_2 & \boldsymbol{A}_0 & \boldsymbol{A}_1 \\ f\boldsymbol{A}_1 & f\boldsymbol{A}_2 & \boldsymbol{A}_0 \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$$

We let

$$\begin{aligned} \boldsymbol{M}_0 &= \boldsymbol{A}_0 + r\boldsymbol{A}_1 + r^2\boldsymbol{A}_2 & \boldsymbol{d}_0 &= r^2\boldsymbol{b}_0 + r\boldsymbol{b}_1 + \boldsymbol{b}_0 \\ \boldsymbol{M}_1 &= \boldsymbol{A}_0 + r\omega\boldsymbol{A}_1 + r^2\omega^2\boldsymbol{A}_2 & \boldsymbol{d}_1 &= r^2\boldsymbol{b}_0 + r\omega^2\boldsymbol{b}_1 + \omega\boldsymbol{b}_2 \\ \boldsymbol{M}_1 &= \boldsymbol{A}_0 + r\omega^2\boldsymbol{A}_1 + r^2\omega\boldsymbol{A}_2 & \boldsymbol{d}_2 &= r^2\boldsymbol{b}_0 + r\omega\boldsymbol{b}_1 + \boldsymbol{b}_2 \end{aligned}$$

and recursively compute the product $\boldsymbol{e}_0 = \boldsymbol{M}_0\boldsymbol{d}_0$, $\boldsymbol{e}_1 = \boldsymbol{M}_1\boldsymbol{d}_1$, $\boldsymbol{e}_2 = \boldsymbol{M}_2\boldsymbol{d}_2$,. Finally, we let

$$\begin{aligned} \boldsymbol{c}_0 &= \frac{\boldsymbol{e}_0 + \boldsymbol{e}_1 + \boldsymbol{e}_2}{3r^2} \\ \boldsymbol{c}_1 &= \frac{\boldsymbol{e}_0 + \omega\boldsymbol{e}_1 + \omega^2\boldsymbol{e}_2}{3r} \\ \boldsymbol{c}_1 &= \frac{\boldsymbol{e}_0 + \omega^2\boldsymbol{e}_1 + \omega\boldsymbol{e}_2}{3} \end{aligned}$$

The final result is $\boldsymbol{c} = (\boldsymbol{c}_0 \parallel \boldsymbol{c}_1 \parallel \boldsymbol{c}_2)^\top$

Remark 4. Since $\boldsymbol{A}$ is circulant and each block $\boldsymbol{A}_i$ in algorithm 3 is also circulant, it is enough to store only the first row/column of the matrices. Similarly, when we compute the matrices $\boldsymbol{M}_i$, it is sufficient to operate on the first row/column of each $\boldsymbol{A}_i$.

5.2 Coprime Factor Case

Suppose $N = N_1N_2$ where N_1, N_2 are relatively prime. Good-Thomas Prime Factorization [17,28] is a technique to reduce the computation of fourier transform of length N to the computation of 2 dimensional fourier transform of dimension $N_1 \times N_2$. In turn its computation can be facilitated by "nesting" a fast algorithm for 1 dimensionaal fourier transform inside another 1 dimensional fast algorithm. This factorization can also be repeated recursively and reduce the original computation to the computation of a multi-dimensional fourier transform.

In the context of computing cyclic convolutions, we use a variant of Good-Thomas prime Factorization due to Agrawal and Cooley [1]. The main benefit of this approach is that it doesn't require additional roots of unity, thereby suitable to our setting where inexpensive roots of unity are scarce.

We need the notion of a stride permutation.

Definition 5. *(Stride Permutation) Assume $N = N_1N_2$ where N_1, N_2 are relatively prime. Let $0 < E_1, E_2 < N$ be unique integers satisfying:*

$$\begin{aligned} E_1 &\equiv 1 \pmod{N_1} & E_1 &\equiv 0 \pmod{N_2} \\ E_2 &\equiv 0 \pmod{N_1} & E_2 &\equiv 1 \pmod{N_2} \end{aligned}$$

We will also call E_1, E_2 the CRT basis *with respect to $N = N_1N_2$. Define the stride permutation associated with N as*

$$\sigma : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\} \qquad N_2 i + j \mapsto iE_1 + jE_2 \bmod N$$

For any $0 \le i < N_1, 0 \le j < N_2$

Example 8. Suppose $N = N_1N_2 = 3 \times 4 = 12$, we can let $E_1 = 4, E_2 = 9$. The stride permutation becomes:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 0 & 9 & 6 & 3 & 4 & 1 & 10 & 7 & 8 & 5 & 2 & 11 \end{pmatrix}$$

Proposition 7. *Assume $N = N_1N_2$ where N_1, N_2 are relatively prime. Let E_1, E_2 be the CRT basis in definition 5. Let $\boldsymbol{P} = \boldsymbol{P}_\sigma$, $\boldsymbol{P}_{i,j} = \delta_{\sigma(i),j}$ for $0 \le i, j < N$ be the (row)-circulant permutation matrix associated with the stride permutation in definition 5.*

Let ζ_N be a principal N^{th} root of unity, and let $\boldsymbol{V}_N$ denote the $N \times N$ fourier matrix where $(\boldsymbol{V}_N)_{i,j} = \zeta_N^{ij}$. Then

1. *If we let $\zeta_{N_1} := \zeta_N^{E_1}$, $\zeta_{N_2} := \zeta_N^{E_2}$. Then ζ_{N_1}, ζ_{N_2} are resepctively principal ${N_1}^{th}$ and ${N_2}^{th}$ root of unity.*
2. *$\boldsymbol{V}_N = \boldsymbol{P}^{-1}(\boldsymbol{V}_{N_1} \otimes \boldsymbol{V}_{N_2})\boldsymbol{P}$, where $\boldsymbol{V}_{N_1}, \boldsymbol{V}_{N_2}$ are fourier matrices associated with ζ_{N_1}, ζ_{N_2}. $\otimes$ denotes the matrix kronecker product.*

Proof. Item 1 of proposition 7 is obvious from the Chinese remainder theorem

For the second item of proposition 7, we let $\sigma : \{0, \dots, N-1\} \to \{0, \dots, N-1\}$ be the stride permutation with respect to N defined in definition 5. For each $0 \le i, j < N$ there exist unique $0 \le a_1, a_2 < N_1$, $0 \le b_1, b_2 < N_2$ such that

$$i \equiv a_1E_1 + b_1E_2 \pmod N \qquad j \equiv a_2E_1 + b_2E_2 \pmod N$$

Therefore,

$$\begin{aligned}\left(\boldsymbol{P}^{-1}(\boldsymbol{V}_{N_1} \otimes \boldsymbol{V}_{N_2})\boldsymbol{P}\right)_{i,j} &= (\boldsymbol{V}_{N_1} \otimes \boldsymbol{V}_{N_2})_{\sigma^{-1}(i),\sigma^{-1}(j)} = (\boldsymbol{V}_{N_1} \otimes \boldsymbol{V}_{N_2})_{a_1N+b1,a_2N+b_2} \\ &= (\boldsymbol{V}_{N_1})_{a_1,a_2}(\boldsymbol{V}_{N_2})_{b_1,b_2} = \zeta_N^{a_1a_2E_1+b_1b_2E_2} = \zeta_N^{ij}\end{aligned}$$

Since by Chinese remainder theorem $ij \equiv a_1a_2E_1 + b_1b_2E_2 \pmod N$ □

The following proposition describes the main idea of Agrawal Cooley algorithm. We refer the reader to [18, Sect 7.2] for more detail.

Proposition 8. 1. *Under the assumption of proposition 7, the inverse fourier matrix satisfies*

$$\boldsymbol{V}_N^{-1} = \boldsymbol{P}^{-1}\left(\boldsymbol{V}_{N_1}^{-1} \otimes \boldsymbol{V}_{N_2}^{-1}\right)\boldsymbol{P}$$

2. *Let $\boldsymbol{u}, \boldsymbol{v}$ be 2 vectors of length N. Let bmH be the $N \times N$ circulant matrix whose first column is $\boldsymbol{u}$. Let $\boldsymbol{w} = \boldsymbol{u} \circledast \boldsymbol{v} = \boldsymbol{Hv}$ be their circular convolution product. If $\boldsymbol{P}$ is the stride permutation matrix as in proposition 7 Then $\boldsymbol{Pw} = \left(\boldsymbol{PHP}^{-1}\right) \cdot \boldsymbol{Pv}$ and $\boldsymbol{H}_1 = \boldsymbol{PHP}^{-1}$ is an $N_1 \times N_1$ block circulant matrix, each block also a circulant matrix of dimension $N_2 \times N_2$*
3. *Hence the block fourier transform $bmH_2 = (\boldsymbol{V}_{N_1} \otimes \boldsymbol{I}_{N_2})\,\boldsymbol{H}_1\left(\boldsymbol{V}_{N_1}^{-1} \otimes \boldsymbol{I}_{N_2}\right)$ is a block diagonal matrix, each block an $N_2 \times N_2$ circulant matrix*
4. *Finally, the fourier transform $\boldsymbol{H}_3 = (\boldsymbol{I}_{N_1} \otimes \boldsymbol{V}_{N_2})\,\boldsymbol{H}_2\left(\boldsymbol{I}_{N_1} \otimes \boldsymbol{V}_{N_2}^{-1}\right)$ is a diagonal matrix*

Proof. Item 1 of proposition 8 is obtained from the well known mix product property of matrix tensor product.

The first part of item 2 of proposition 8 is obvious. We now prove that $\boldsymbol{H}_1 = \boldsymbol{PHP}^{-1}$ has the desired property. For $0 \le a_1, a_2 < N_1$, $0 \le b_1, b_2 < N_2$ we need to show that

$$(\boldsymbol{H}_1)_{a_1N_2+b_1,a_2N_2+b_2} = (\boldsymbol{H}_1)_{0,(a_1-a_2 \bmod N_1)N_2+(b_2-b_1 \bmod N_2)}$$

But

$$\begin{aligned}&(\boldsymbol{H}_1)_{a_1N_2+b_1,a_2N_2+b_2} = \boldsymbol{H}_{a_1E_1+b_1E_2 \bmod N, a_2E_1+b_2E_2 \bmod N} \\ &= \boldsymbol{u}_{(a_1-a_2)E_1+(b_2-b_1)E_2 \bmod N} = (\boldsymbol{H}_1)_{0,(a_1-a_2 \bmod N_1)N_2+(b_2-b_1 \bmod N_2)}\end{aligned}$$

The claim is proven.

Item 3 and 4 of proposition 8 are direct consequences of the fourier transform. $\boldsymbol{V}_{N_1} \otimes \boldsymbol{I}_{N_2}$ operates at block level and will transform a block-circulant matrix to a block-diagonal one. On the otherhand, $\boldsymbol{I}_{N_1} \otimes \boldsymbol{V}_{N_2}$ operates parallel within each block, and will transform each circulant block into a diagonal block. □

Algorithm 4 instantiates the Good-Thomas Prime Factorization strategy. The correctness follows directly from proposition 8.

Algorithm 4 Good-Thomas Prime Factorization

```
Input: 2 length $N$ sequences $\boldsymbol{u}, \boldsymbol{v}$
Output: Their circular convolution product $\boldsymbol{w} = \boldsymbol{u} \circledast \boldsymbol{v}$
1: Break $N = N_1 N_2$ into 2 coprime factors. Find the CRT basis $E_1, E_2$ and stride permutation matrix
   $\boldsymbol{P}$ as in proposition 7.
2: Compute permutations $\boldsymbol{u}_1 = \boldsymbol{P}\boldsymbol{u}$, $\boldsymbol{v}_1 = \boldsymbol{P}\boldsymbol{v}$
3: Calculate the block fourier transforms:
       $$\boldsymbol{u}_2 = (\boldsymbol{V}_{N_1} \otimes \boldsymbol{I}_{N_2})\,\boldsymbol{u}_1, \qquad \boldsymbol{v}_2 = (\boldsymbol{V}_{N_1} \otimes \boldsymbol{I}_{N_2})\,\boldsymbol{v}_1$$
       ▷ This step can be recursively computed if we can break up $N_1$ to coprime factors
4: Break $\boldsymbol{u}_2 = (\boldsymbol{u}_2^{(0)} \parallel \ldots \parallel \boldsymbol{u}_2^{(N_1-1)})^\top$, $\boldsymbol{v}_2 = (\boldsymbol{u}_v^{(0)} \parallel \ldots \parallel \boldsymbol{u}_v^{(N_1-1)})^\top$ into $N_1$ consecutive blocks
5: for each pair $\boldsymbol{u}_2^{(i)}, \boldsymbol{v}_2^{(i)}$ do
6:     Calculate $\boldsymbol{w}_2^{(i)} = \boldsymbol{u}_2^{(i)} \circledast \boldsymbol{v}_2^{(i)}$ ▷ This step can also be recursively computed if we can break up $N_2$
       to coprime factors
7: end for
8: Let $\boldsymbol{w}_2 = (\boldsymbol{w}_2^{(0)} \parallel \ldots \parallel \boldsymbol{w}_2^{N_1-1})^\top$. Calculate the inverse block fourier transform $\boldsymbol{w}_1 = \left(\boldsymbol{V}_{N_1}^{-1} \otimes \boldsymbol{I}_{N_2}\right) \boldsymbol{w}_2$
9: Return $\boldsymbol{w} = \boldsymbol{P}^{-1}\boldsymbol{w}_1$
```

5.3 Combining the 2 strategies

When we search for smooth divisors of $p^r - 1$ where p is a small prime and $r > 0$ a small integer, we find that such factor $N \mid p^r - 1$ frequently factorizes as $N = q_1 q_2 \ldots q_{n-1} q_n^e$, $q_1 \ldots q_n$ distinct primes. When such case occurs, it is possible to combine both strategies in sections 5.1 and 5.2.

The highlevel idea is to first use Good-Thomas Prime Factorization and the block fourier matrices to convert the length N convolution respectively to a length q_1, length $q_1 q_2$, etc. length $q_1 \ldots q_{n-1}$ block-wise convolution. For each block in the last step, which has size $q_n^e \times q_n^e$, we employ algorithm 3 to calculate the convolution. Finally, we use a series of inverse block fourier matrices associated with Good-Thomas Prime Factorization to obtain the final result.

A more precise mathematical expression of the procedure above is given by proposition 9.

Proposition 9. *Suppose $N = q_1 \ldots q_{n-1} q_n^e$ where $e > 0$ and $q_1 \ldots q_n$ are distinct primes. Define*

$$\begin{array}{lllll} N_0 = 1, & N_1 = q_1, & N_2 = q_1 q_2, & \ldots, & N_{n-1} = q_1 q_2 \ldots q_{n-1} \\ M_0 = N, & M_1 = N/q_1, & M_2 = N/q_1 q_2, & \ldots, & M_{n-1} = N/q_1 \ldots q_{n-1} \end{array}$$

For $1 \leq i \leq n-1$ also let $\boldsymbol{P}_i$ be the stride permutation matrix with respect to the factorization $M_{i-1} = q_i M_i$ (see definition 5). Define

$$\boldsymbol{V}_{(n)} \coloneqq \left(\boldsymbol{V}_{q_1} \otimes \boldsymbol{V}_{q_2} \otimes \ldots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{q_n^e}\right) \left(\boldsymbol{I}_{N_{n-2}} \otimes \boldsymbol{P}_{n-1}\right) \left(\boldsymbol{I}_{N_{n-3}} \otimes \boldsymbol{P}_{n-2}\right) \ldots \left(\boldsymbol{I}_{N_1} \otimes \boldsymbol{P}_2\right) \boldsymbol{P}_1$$

where $\boldsymbol{V}_{q_i}$ is the fourier (Vandermonde) matrix of dimension $q_i \times q_i$ with respect to the Good-Thomas prime factorization $M_{i-1} = q_i M_i$ (q_i plays the role of N_1 in proposition 7).

Let $\boldsymbol{u}, \boldsymbol{v}$ be 2 length N sequences, $\boldsymbol{H}$ the circulant matrix whose first column is $\boldsymbol{u}$. Let their circular convolution $\boldsymbol{w} = \boldsymbol{u} \circledast \boldsymbol{v} = \boldsymbol{H}\boldsymbol{v}$, then

$$\boldsymbol{V}_{(n)}\boldsymbol{w} = \left(\boldsymbol{V}_{(n)}\boldsymbol{H}\boldsymbol{V}_{(n)}^{-1}\right) \cdot \boldsymbol{V}_{(n)}\boldsymbol{v}$$

And $\boldsymbol{V}_{(n)}\boldsymbol{H}\boldsymbol{V}_{(n)}^{-1}$ is a block-diagonal matrix, with $N_{n-1} = q_1 \ldots q_{n-1}$ blocks and each block a circulant matrix of dimension $q_n^e \times q_n^e$.

Proof. We prove proposition 9 by induction on the number of coprime factors n of N.

If $n = 2$, $N = q_1 q_2^e$. Let $N_1 = q_1$, $N_2 = q_2^e$, $\boldsymbol{P}_1$ the stride permutation matrix with respect to $N = N_1 N_2$. Item 1 to 3 in proposition 8 implies

$$\begin{aligned}
&\left(\boldsymbol{V}_{q_1} \otimes \boldsymbol{I}_{q_2^e}\right) \boldsymbol{P}_1 \boldsymbol{w} \\
&= \left(\boldsymbol{V}_{q_1} \otimes \boldsymbol{I}_{q_2^e}\right) \left(\boldsymbol{P}_1 \boldsymbol{H} \boldsymbol{P}_1^{-1}\right) \left(\boldsymbol{V}_{q_1}^{-1} \otimes \boldsymbol{I}_{q_2^e}\right) \cdot \left(\boldsymbol{V}_{q_1} \otimes \boldsymbol{I}_{q_2^e}\right) \boldsymbol{P}_1 \boldsymbol{v} \\
&\implies \boldsymbol{V}_{(2)}\boldsymbol{w} = \left(\boldsymbol{V}_{(2)}\boldsymbol{H}\boldsymbol{V}_{(2)}^{-1}\right) \cdot \boldsymbol{V}_{(2)}\boldsymbol{v}
\end{aligned}$$

And that $\boldsymbol{V}_{(2)}\boldsymbol{H}\boldsymbol{V}_{(2)}^{-1}$ is a block-diagnonal matrix, each block a circulant matrix of dimension $q_2^e \times q_2^e$. This proves the base case.

Assume the assumption holds for the number of coprime factors $1, 2 \ldots n$. Now if $N = q_1 q_2 \ldots q_n q_{n+1}^e$, we can also write $N = q_1 q_2 \ldots q_{n-1} Q_n$, where $Q_n = q_n q_{n+1}^e$. By inductive hypothesis: $\boldsymbol{V}_{(n)}\boldsymbol{w} = \left(\boldsymbol{V}_{(n)}\boldsymbol{H}\boldsymbol{V}_{(n)}^{-1}\right) \cdot \boldsymbol{V}_{(n)}\boldsymbol{v}$, where $\boldsymbol{V}_{(n)}$ is as in proposition 9 except we replace $\boldsymbol{I}_{q_n^e}$ by $\boldsymbol{I}_{Q_n}$. Again by inductive hypothesis $\boldsymbol{V}_{(n)}\boldsymbol{H}\boldsymbol{V}_{(n)}^{-1}$ is a block-diagonal matix, each block a circulant matrix of dimension $Q_n \times Q_n$.

Again Item 1 to 3 in proposition 8 says that for $Q_n = q_n q_{n+1}^e$, P_n the stride permutation matrix with respect to the factorization $Q_n = q_n q_{n+1}^e$, and for any $Q_n \times Q_n$ circulant matrix $\boldsymbol{U}$:

$$\left(\boldsymbol{V}_{q_n} \otimes \boldsymbol{I}_{q_{n+1}^e}\right) \left(\boldsymbol{P}_n \boldsymbol{U} \boldsymbol{P}_n^{-1}\right) \left(\boldsymbol{V}_{q_n}^{-1} \otimes \boldsymbol{I}_{q_{n+1}^e}\right)$$

is a block-diagnoal matrix, each block is circulant of dimension $q_{n+1}^e \times q_{n+1}^e$.

Apply this operator to each circulant block within $\boldsymbol{V}_{(n)}$, we obtain

$$\left(\boldsymbol{I}_{q_1 \ldots q_{n-1}} \otimes \boldsymbol{V}_{q_n} \otimes \boldsymbol{I}_{q_{n+1}^e}\right) \left(\boldsymbol{I}_{q_1 \ldots q_{n-1}} \otimes \boldsymbol{P}_n\right) \left(\boldsymbol{V}_{(n)}\boldsymbol{H}\boldsymbol{V}_{(n)}^{-1}\right) \left(\boldsymbol{I}_{q_1 \ldots q_{n-1}} \otimes \boldsymbol{P}_n^{-1}\right) \left(\boldsymbol{I}_{q_1 \ldots q_{n-1}} \otimes \boldsymbol{V}_{q_n}^{-1} \otimes \boldsymbol{I}_{q_{n+1}^e}\right)$$

is a block diagonal matrix, each block circulant of dimension $q_{n+1}^e \times q_{n+1}^e$. But

$$\begin{aligned}
&\left(\boldsymbol{I}_{q_1\dots q_{n-1}} \otimes \boldsymbol{V}_{q_n} \otimes \boldsymbol{I}_{q_{n+1}^e}\right)\left(\boldsymbol{I}_{q_1\dots q_{n-1}} \otimes \boldsymbol{P}_n\right)\boldsymbol{V}_{(n)}\\
&= \left(\boldsymbol{I}_{q_1\dots q_{n-1}} \otimes \boldsymbol{V}_{q_n} \otimes \boldsymbol{I}_{q_{n+1}^e}\right)\left(\boldsymbol{I}_{q_1\dots q_{n-1}} \otimes \boldsymbol{P}_n\right)\left(\boldsymbol{V}_{q_1} \otimes \boldsymbol{V}_{q_2} \otimes \dots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{Q_n}\right)\\
&\left(\boldsymbol{I}_{N_{n-2}} \otimes \boldsymbol{P}_{n-1}\right)\dots\left(\boldsymbol{I}_{N_1} \otimes \boldsymbol{P}_2\right)\boldsymbol{P}_1\\
&= \left(\boldsymbol{V}_{q_1} \otimes \dots \otimes \boldsymbol{V}_{q_n} \otimes \boldsymbol{I}_{q_{n+1}^e}\right)\left(\boldsymbol{I}_{N_{n-1}} \otimes \boldsymbol{P}_n\right)\dots\left(\boldsymbol{I}_{N_1} \otimes \boldsymbol{P}_2\right)\boldsymbol{P}_1\\
&= \boldsymbol{V}_{(n+1)}
\end{aligned}$$

where $N_{n-1} = q_1 q_2 \dots q_n$ and $Q_n = q_n q_{n+1}^e$. In addition, we use the mix product property of tensor products to move around and combine the components in order to obtain the second last equality. This proves the induction step. □

We briefly recall that the tensor product matrix $\boldsymbol{V}_{q_1} \otimes \dots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{q_n^e}$ can be decomposed into a series of block-wise operations peppered with suitable permutations. We refer the reader to [18, Chapter 2] for more detail and background.

Definition 6. *Let $M, N > 0$, we define the* tensor interchange permutation

$$\begin{aligned}
\tau :\ &\{0, 1, \dots, MN-1\} \to \{0, 1, \dots, MN-1\}\\
\forall 0 \le a < M, 0 \le b < N :\quad & aN + b \mapsto bM + a
\end{aligned}$$

We will let $\boldsymbol{Q} = \boldsymbol{Q}_\tau$, $(\boldsymbol{Q})_{i,j} = \delta_{\tau(i),j}$ be the (row)-permutation matrix associated with τ.

Proposition 10. *Let $M, N > 0$, $\boldsymbol{A}$ an $M \times M$ matrix and $\boldsymbol{B}$ an $N \times N$ matrix. Using the notation in definition 6, we have:*

$$\boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{Q}^{-1}\left(\boldsymbol{B} \otimes \boldsymbol{A}\right)\boldsymbol{Q}$$

Proof. Let $i = a_1 N + b_1$, $j = a_2 N + b_2$ where $0 \le a_1, a_2 < M$, $0 \le b_1, b_2 < N$. Then

$$\begin{aligned}
&\left(\boldsymbol{Q}^{-1}\left(\boldsymbol{B} \otimes \boldsymbol{A}\right)\boldsymbol{Q}\right)_{i,j} = \left(\boldsymbol{B} \otimes \boldsymbol{A}\right)_{\tau(i),\tau(j)} = \left(\boldsymbol{B} \otimes \boldsymbol{A}\right)_{b_1 M + a_1, b_2 M + a_2} = \boldsymbol{B}_{b_1,b_2}\boldsymbol{A}_{a_1,a_2}\\
&= \left(\boldsymbol{A} \otimes \boldsymbol{B}\right)_{a_1 N + b_1, a_2 N + b_2} = \left(\boldsymbol{A} \otimes \boldsymbol{B}\right)_{i,j}
\end{aligned}$$

□

Corollary 1. *Under the notation of definition 6, propositions 9 and 10.*

1. $\boldsymbol{V}_{q_1} \otimes \dots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{q_n^e} = \prod_{i=1}^{n-1}\left(\boldsymbol{I}_{N_{i-1}} \otimes \boldsymbol{V}_{q_i} \otimes \boldsymbol{I}_{M_i}\right)$
2. *Let $\boldsymbol{Q}_i$ denote the tensor product interchange permutation matrix between a $N_{i-1} \times N_{i-1}$ and a $q_i \times q_i$ matrix (see definition 6). Then*

$$\begin{aligned}
\boldsymbol{V}_{q_1} \otimes \dots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{q_n^e} &= \prod_{i=1}^{n-1}\left(\left(\boldsymbol{Q}_i\left(\boldsymbol{V}_{q_i} \otimes \boldsymbol{I}_{N_{i-1}}\right)\right) \otimes \boldsymbol{I}_{M_i}\right)\\
&= \prod_{i=1}^{n-1}\left(\left(\boldsymbol{Q}_i \otimes \boldsymbol{I}_{M_i}\right)\left(\boldsymbol{V}_{q_i} \otimes \boldsymbol{I}_{N/q_i}\right)\left(\boldsymbol{Q}_i^{-1} \otimes \boldsymbol{I}_{M_i}\right)\right)
\end{aligned}$$

3. Let $\boldsymbol{Q}_i'$ denote the tensor product interchange permutation matrix between a $q_i \times q_i$ and a $M_i \times M_i$ matrix (see definition 6). Then

$$\boldsymbol{V}_{q_1} \otimes \ldots \otimes \boldsymbol{V}_{q_{n-1}} \otimes \boldsymbol{I}_{q_n^e} = \prod_{i=1}^{n-1} \left(\boldsymbol{I}_{N_{i-1}} \otimes \left(\boldsymbol{Q}_i' \left(\boldsymbol{I}_{M_i} \otimes \boldsymbol{V}_{q_i} \right) \boldsymbol{Q}_i'^{-1} \right) \right)$$

$$= \prod_{i=1}^{n-1} \left(\left(\boldsymbol{I}_{N_{i-1}} \otimes \boldsymbol{Q}_i' \right) \left(\boldsymbol{I}_{N/q_i} \otimes \boldsymbol{V}_{q_i} \right) \left(\boldsymbol{I}_{N_{i-1}} \otimes \boldsymbol{Q}_i'^{-1} \right) \right)$$

Proof. Item 1 is a direct consequence of mixed product property of tensor products. Item 2 and 3 follow from proposition 10. □

Finally, algorithm 5 shows one way to instantiate the combined strategy. Correctness follows immediately from proposition 10 and corollary 1.

6 Experiments and Conclusion

We implemented algorithm 5 in Sagemath/Python [26] environment. The parameters and configurations of algorithm 5 is chosen in the following manner:

$p = 2$, **Length** $= 2000$: We let $N = 4095 = 13 * 7 * 5 * 3^2 = 2^{12} - 1$. We find a primitve degree 12 polynomial mod 2: $f_{12}(x) = x^{12} + x^7 + x^6 + x^5 + x^3 + x + 1$ and use algorithm 1 to lift the polynomial over prime powers $2^8, 2^{16}, 2^{32}$. Use x as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 2$, **Length** $= 30000$: We let $N = 69615 = 17 * 13 * 7 * 5 * 3^2 \mid 2^{24} - 1$. We find a primitve degree 24 polynomial mod 2: $f_{24}(x) = x^{24} + x^{16} + x^{15} + x^{14} + x^{13} + x^{10} + x^9 + x^7 + x^5 + x^3 + 1$ and use algorithm 1 to lift the polynomial over prime powers $2^8, 2^{16}, 2^{32}$. Use x^{241} as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 2$, **Length** $= 100000$: We let $N = 233415 = 19 * 13 * 7 * 5 * 3^3 \mid 2^{36} - 1$. We find a primitve degree 36 polynomial mod 2: $f_{36}(x) = x^{36} + x^{23} + x^{22} + x^{20} + x^{19} + x^{17} + x^{14} + x^{13} + x^8 + x^6 + x^5 + x + 1$ and use algorithm 1 to lift the polynomial over prime powers $2^8, 2^{16}, 2^{32}$. Use $x^{37*73*109}$ as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 17$, **Length** $= 1000$: We let $N = 2880 = 5 * 3^2 * 2^6 \mid 17^4 - 1$. We find a primitve degree 4 polynomial mod 17: $f_4(x) = x^4 + 7x^2 + 10x + 3$ and use algorithm 1 to lift the polynomial over prime powers $17^2, 17^4, 17^8$. Use x^{29} as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 17$, **Length** $= 80000$: We let $N = 2880 = 29 * 5 * 3^2 * 2^7 \mid 17^8 - 1$. We find a primitve degree 8 polynomial mod 17: $f_8(x) = x^8 + 11x^4 + 12x^3 + 6x + 3$ and use algorithm 1 to lift the polynomial over prime powers $17^2, 17^4, 17^8$. Use x^{41761} as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 31$, **Length** $= 900$: We let $N = 1920 = 5 * 3 * 2^7 \mid 31^4 - 1$. We find a primitve degree 4 polynomial mod 31: $f_4(x) = x^4 + 3x^2 + 16x + 3$ and use algorithm 1 to lift the polynomial over prime powers $31^2, 31^4, 31^8$. Use x^{13*37} as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

$p = 31$, **Length** $= 10000$: We let $N = 24960 = 13 * 5 * 3 * 2^7 \mid 31^4 - 1$. Use again the primitive degree 4 polynomial mod 31: $f_4(x) = x^4 + 3x^2 + 16x + 3$ and algorithm 1 to lift the polynomial over prime powers $31^2, 31^4, 31^8$. Use x^{37} as a principal N^{th} root of unity and finally employ algorithm 5 and algorithm 3.

We compare our implementation with 3 other methods.

- *The Direct Method*: Our base line method is to represent circular as polynomial multiplication (mod $p^m, x^N - 1$). This is very similar to directly doing a matrix vector product where the matrix is circulant. We use Sagemath's generic engine to implement the operations and turn off all optimizations.
- *The Multimodular NTT Method* We use the Multimodular NTT method briefly introduced in section 2. We use Sympy's ntt and intt method to calculate the Classical NTT and finally Sagemath's CRT vector method to compute the Chinese remainder isomorphism.
- *The Flint Method* This is similar to the Direct Method, but now we turn on Sagemath's Flint engine to implement polynomial arithmetics. We let Flint [27] handle various optimizations when doing multiplications.

6.1 Result

Method/Length	**2000**	**30000**	**100000**
Direct	113	22247	239498
Algorithm 5	509	22402	230037
Multimodular-NTT	44	908	4193
Flint	2.9	77	420

Table 1. Average time in ms for convolution when modulus is 2^8. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**2000**	**30000**	**100000**
Direct	331	64288	778338
Algorithm 5	526	28775	337630
Multimodular-NTT	65	1340	6341
Flint	3	117	439

Table 2. Average time in ms for convolution when modulus is 2^{16}. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**2000**	**30000**	**100000**
Direct	608	133040	2633100
Algorithm 5	972	64871	598649
Multimodular-NTT	112	2249	8314
Flint	6.3	139	666

Table 3. Average time in ms for convolution when modulus is 2^{32}. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

6.2 Conclusion

On one hand, experimental data demonstrate that algorithm 5 performs better than the base-line $O(N^2)$ method. This result confirms the feasibility of our strategy and gives an empirical indication that our method has asymptotic complexity $O(N \log N)$.

Method/Length	**1000**	**80000**
Direct	29	149945
Algorithm 5	299	27015
Multimodular-NTT	22	4166
Flint	2.8	243

Table 4. Average time in ms for convolution when modulus is 17^2. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**1000**	**80000**
Direct	83	455928
Algorithm 5	304	29573
Multimodular-NTT	43	6527
Flint	1.6	377

Table 5. Average time in ms for convolution when modulus is 17^4. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

On the otherhand, the data show that algorithm 5 performs consistenly worse than other optimization strategies. We summarize some of the reasons of its inefficiency:

– We use modular polynomial multiplication while other optimizations use integer arithmetic throughout the NTT computations. Hence our method will take up more space and requires much longer time to do one basic multiplication.
– Most optimization strategies employ radix-2 Cooley-Tuckey algorithm while we adopt a mixed radix approach. It is known that mixed radix FFT is concretely less efficient than pure radix-2 FFT.
– Algorithm 5 is written in pure python with Sagemath library while some optimizations are implemented in c/c++ code behind the python interface. Python overhead could account for some of our inefficiencies.

Potential Benefits and room for improvements When the modulus is a prime power, we initially hope to leverage the modular structure of the underlying ring and avoid using Chinese remainder theorem, thereby avoid doing NTT multiple times under different prime modulus. It seems that Item 1 and 2 in the above consideration more than offset the costs of doing NTT multiple times.

One room for improvements may come from using large prime p and very small degree primitive polynomials. The difference between the time it takes to multiply 2 numbers $\pmod{p^m}$ and the time it takes to multiply small degree polynomials $\pmod{p^m, F(x)}$ might not be very large. In this case we may gain some efficiency by reducing the number of NTT we need to perform.

Another room for improvement would come from optimizing the computation of block-fourier transform of small dimension, in other words, the computation of matrix vector product $(\boldsymbol{V}_p \otimes \boldsymbol{I}_M)\boldsymbol{v}$ where $\boldsymbol{V}_p$ is the fourier (Vandermonde) matrix of a small prime p and M the block size. It is the bottleneck of Good-Thomas Prime Factorization computation so our method could benefit from any improvements in the block-wise fourier computation. More generally, a better FFT-style algorithm that doesn't use additional roots of unity will likely benefit our approach.

Method/Length	**1000**	**80000**
Direct	141	1231909
Algorithm 5	374	49495
Multimodular-NTT	66	8520
Flint	3.7	564

Table 6. Average time in ms for convolution when modulus is 17^8. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**900**	**10000**
Direct	69	7262
Algorithm 5	181	3162
Multimodular-NTT	32	662
Flint	1.3	41.6

Table 7. Average time in ms for convolution when modulus is 31^2. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**900**	**10000**
Direct	77	7206
Algorithm 5	190	3386
Multimodular-NTT	43	865
Flint	4.2	47

Table 8. Average time in ms for convolution when modulus is 31^4. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Method/Length	**900**	**10000**
Direct	116	12588
Algorithm 5	244	4312
Multimodular-NTT	66	1302
Flint	4.8	41

Table 9. Average time in ms for convolution when modulus is 31^8. We use 50 test when Length is ≤ 5000 and 20 test data when Length > 5000

Algorithm 5 Circular Convolution over prime power

```
Input: Length $N = q_1 \dots q_{n-1} q_n^e$, where $q_1 \dots q_n$ are distinct small primes; 2 length $N$ sequences $\boldsymbol{u}, \boldsymbol{v}$; a
    principal $N^{\text{th}}$ root of unity $\zeta_N$
Output: Circular convolution $\boldsymbol{w} = \boldsymbol{u} \circledast \boldsymbol{v}$
 1: Determine the permutation matrix
        $$\boldsymbol{P} = \left(\boldsymbol{I}_{N_{n-1}} \otimes \boldsymbol{P}_{n-1}\right)\left(\boldsymbol{I}_{N_{n-3}} \otimes \boldsymbol{P}_{n-2}\right) \dots \left(\boldsymbol{I}_{N_1} \otimes \boldsymbol{P}_2\right) \boldsymbol{P}_1$$
    where $N_1, \dots, N_{n-2}$ and $\boldsymbol{P}_1, \dots, \boldsymbol{P}_{N_{n-2}}$ are as in proposition 9
 2: Calculate the permutations $\boldsymbol{u}_0 = \boldsymbol{P}\boldsymbol{u}$, $\boldsymbol{v}_0 = \boldsymbol{P}\boldsymbol{v}$
 3: for $i \leftarrow 1$ to $n-1$ do
 4:     Permute arguments
            $$\boldsymbol{u}_i^{(1)} = \left(\boldsymbol{Q}_i^{-1} \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{u}_{i-1}, \quad \boldsymbol{v}_i^{(1)} = \left(\boldsymbol{Q}_i^{-1} \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{v}_{i-1}$$
        where $\boldsymbol{Q}_i$ is as in corollary 1
 5:     Break $\boldsymbol{u}_i^{(1)}, \boldsymbol{v}_i^{(1)}$ into $q_i$ blocks. Compute the block fourier transform
            $$\boldsymbol{u}_i^{(2)} = \left(\boldsymbol{V}_{q_i} \otimes \boldsymbol{I}_{N/q_i}\right)\boldsymbol{u}_i^{(1)}, \quad \boldsymbol{v}_i^{(2)} = \left(\boldsymbol{V}_{q_i} \otimes \boldsymbol{I}_{N/q_i}\right)\boldsymbol{v}_i^{(1)}$$
        $\boldsymbol{V}_{q_i}$ is obtained from $\zeta_N$ and factorization $M_{i-1} = q_i M_i$ as in proposition 7
 6:     Again calculate the permutations
            $$\boldsymbol{u}_i = \left(\boldsymbol{Q}_i \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{u}_i^{(2)}, \quad \boldsymbol{v}_i = \left(\boldsymbol{Q}_i \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{v}_i^{(2)}$$
        where $\boldsymbol{Q}_i$ is as in corollary 1
 7: end for
 8: Parse
        $$\boldsymbol{u}_{n-1} = \left(\boldsymbol{u}_{n-1,0} \parallel \dots \parallel \boldsymbol{u}_{n-1,N_{n-1}-1}\right)^{\top}$$
        $$\boldsymbol{v}_{n-1} = \left(\boldsymbol{v}_{n-1,0} \parallel \dots \parallel \boldsymbol{v}_{n-1,N_{n-1}-1}\right)^{\top}$$
    into $N_{n-1}$ consecutive blocks, each of size $q_n^e$
 9: for $i \leftarrow 0$ to $N_{n-1} - 1$ do
10:     Let $\boldsymbol{H}_i$ represent the circulant matrix whose first column is $\boldsymbol{u}_{n-1,i}$
11:     Calculate
            $$\boldsymbol{w}_{n-1,i} = \text{GenCircMatMult}(\boldsymbol{H}_i, \boldsymbol{v}_{n-1,i}, 1, q_n^e)$$
        using algorithm 3                                  ▷ $\boldsymbol{w}_{n-1,i} = \boldsymbol{u}_{n-1,i} \circledast \boldsymbol{v}_{n-1,i}$
12: end for
13: Combine $\boldsymbol{w}_{n-1} = \left(\boldsymbol{w}_{n-1,0} \parallel \dots \parallel \boldsymbol{w}_{n-1,N_{n-1}-1}\right)^{\top}$
14: for $i \leftarrow n-1$ down to 2 do
15:     Calculate the permutation $\boldsymbol{w}_i^{(1)} = \left(\boldsymbol{Q}_i^{-1} \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{w}_i$, where $\boldsymbol{Q}_i$ is as in corollary 1
16:     Break $\boldsymbol{w}_i^{(1)}$ into $q_i$ blocks. Compute the block inverse fourier transform
            $$\boldsymbol{w}_i^{(2)} = \left(\boldsymbol{V}_{q_i}^{-1} \otimes \boldsymbol{I}_{N/q_i}\right)\boldsymbol{w}_i^{(1)}$$
        $\boldsymbol{V}_{q_i}$ is obtained from $\zeta_N$ and factorization $M_{i-1} = q_i M_i$ as in proposition 7
17:     Again calculate the permutation $\boldsymbol{w}_{i-1} = \left(\boldsymbol{Q}_i \otimes \boldsymbol{I}_{M_i}\right)\boldsymbol{w}_i^{(2)}$, where $\boldsymbol{Q}_i$ is as in corollary 1
18: end for
19: Return $\boldsymbol{w} = \boldsymbol{P}^{-1}\boldsymbol{w}_0$, where $\boldsymbol{P}$ is as in line 1
```

References

1. Agarwal, R., Cooley, J.: New algorithms for digital convolution. IEEE Transactions on Acoustics, Speech, and Signal Processing **25**(5), 392–410 (1977)
2. Agarwal, S., Cooley, J.: Fast algorithms for convolution. In: Proceedings of the National Computer Conference. pp. 143–147 (1974)
3. Al Badawi, A., Veeravalli, B., Aung, K.M.M.: Efficient polynomial multiplication via modified discrete galois transform and negacyclic convolution. In: Future of Information and Communication Conference (FICC). pp. 785–802. Springer (2019). https://doi.org/10.1007/978-3-030-03402-3_47
4. Berlekamp, E.R.: Negacyclic codes for the lee metric. In: Bose, R.C., Dowling, T.A. (eds.) Combinatorial Mathematics and Its Applications, pp. 298–316. No. 4 in UNC Monograph Series in Probability and Statistics, University of North Carolina Press, Chapel Hill, NC (1969)
5. Bini, G., Flamini, F.: Finite commutative rings and their applications, vol. 680. Springer Science & Business Media (2012)
6. Bluestein, L.: A linear filtering approach to the computation of discrete fourier transform. IEEE Transactions on Audio and Electroacoustics **18**(4), 451–455 (1970)
7. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals - kyber: A cca-secure module-lattice-based kem. In: Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018). https://doi.org/10.1109/EuroSP.2018.00032, https://doi.org/10.1109/EuroSP.2018.00032
8. Bos, L., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G.: Crystals–dilithium: Digital signatures from module lattices. IACR Cryptol. ePrint Arch. **2017**, 633 (2017), https://eprint.iacr.org/2017/633
9. Brakerski, Z.: Leveled fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS). pp. 309–325. ACM (2012). https://doi.org/10.1145/2090236.2090262, https://doi.org/10.1145/2090236.2090262
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. In: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS). pp. 97–106. IEEE (2011). https://doi.org/10.1109/FOCS.2011.12, https://doi.org/10.1109/FOCS.2011.12
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 409–437. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_15, https://doi.org/10.1007/978-3-319-70694-8_15
12. Cooley, J.W., Lewis, P.A., Welch, P.D.: Historical notes on the fast fourier transform. Proceedings of the IEEE **55**(10), 1675–1677 (1967)
13. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of computation **19**(90), 297–301 (1965). https://doi.org/10.2307/2003354
14. Dougherty, S.T., Şahinkaya, S.: On cyclic and negacyclic codes with one-dimensional hulls and their applications. Advances in Mathematics of Communications **16**(4), 765–780 (2022). https://doi.org/10.3934/amc.2022096, https://doi.org/10.3934/amc.2022096
15. Duhamel, P., Hollmann, H.: 'split radix'fft algorithm. Electronics letters **20**(1), 14–16 (1984)
16. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over ntru. In: Proceedings of the 2018 International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 123–151. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_5, https://doi.org/10.1007/978-3-030-03329-3_5
17. Good, I.J.: The interaction algorithm and practical fourier analysis. Journal of the Royal Statistical Society Series B: Statistical Methodology **20**(2), 361–372 (1958)
18. Lu, R.: Algorithms for discrete Fourier transform and convolution. Springer (1989)

19. Martens, J., Vanwormhoudt, M.: Convolution using a conjugate symmetry property for number theoretic transforms over rings of regular integers. IEEE Transactions on Acoustics, Speech, and Signal Processing **31**(5), 1247–1250 (1983). https://doi.org/10.1109/TASSP.1983.1164198
20. McDonald, B.R.: Finite rings with identity. Marcel Dekker (1974)
21. McGuire, G.: An approach to hensel's lemma. Irish Math Soc. Bulletin **47**, 15–21 (2001)
22. Nussbaumer, H.J.: Fast polynomial transform algorithms for digital convolution. IEEE Transactions on Acoustics, Speech, and Signal Processing **28**(2), 205–215 (1980). https://doi.org/10.1109/TASSP.1980.1163422
23. Rosowski, A.: On fast computation of a circulant matrix-vector product. arXiv preprint arXiv:2103.02605 (2021)
24. Schönhage, A., Strassen, V.: Schnelle multiplikation großer zahlen. Computing **7**(3-4), 281–292 (1971). https://doi.org/10.1007/BF02242355
25. Shoup, V.: A new polynomial factorization algorithm and its implementation. Journal of Symbolic Computation **20**(4), 363–397 (1995)
26. Stein, W., Joyner, D.: Sage: System for algebra and geometry experimentation. Acm Sigsam Bulletin **39**(2), 61–64 (2005)
27. team, T.F.: FLINT: Fast Library for Number Theory (2025), version 3.2.1, https://flintlib.org
28. Thomas, L.H.: Using a computer to solve problems in physics. Applications of digital computers pp. 44–45 (1963)
29. Wikipedia contributors: Galois ring — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Galois_ring&oldid=1181997128 (2023), [Online; accessed 29-April-2025]
30. Wikipedia contributors: Principal root of unity — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Principal_root_of_unity&oldid=1223603190 (2024), [Online; accessed 29-April-2025]
31. Wikipedia contributors: Division algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Division_algorithm&oldid=1283459286 (2025), [Online; accessed 29-April-2025]
32. Wikipedia contributors: Hensel's lemma — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hensel%27s_lemma&oldid=1275481796 (2025), [Online; accessed 29-April-2025]
33. Wikipedia contributors: Resultant — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Resultant&oldid=1280437221 (2025), [Online; accessed 30-April-2025]