Notes on Multiplying Cyclotomic Polynomials on a GPU

Joseph Johnston

Abstract

Lattice cryptography has many exciting applications, from homomorphic encryption to zero knowledge proofs. We explore the algebra of cyclotomic polynomials underlying many practical lattice cryptography constructions, and we explore algorithms for multiplying cyclotomic polynomials on a GPU.

Contents

1 Introduction

We herein present a series of six sections dedicated to describing the algebra behind cyclotomic polynomials as well as algorithms for multiplying cyclotomic polynomials on a GPU. Our motivation is lattice cryptography, though we do not discuss any cryptographic topics. The first four sections focus purely on algebra, the last two sections each explore GPU algorithms. Our only novel theorem is Theorem 13, all others folklore. The sections may be summarized as follows.

1. Section 2 introduces notation for algebra to be reused across the first four sections. Cyclotomic extensions, a natural source for studying cyclotomic polynomials, is studied by first discussing such extension in general, and then narrowing study to such extensions of the field of rational numbers and then finite fields. Theorems in this first section relate to when cyclotomic polynomials are irreducible over these base fields, and also to the Galois groups of these extensions.

2. Section 3 is dedicated to covering two crucial concepts to be utilized in subsequent sections. The first concept is representing the reducibility of cyclotomic polynomials over finite fields, and while we will ultimately only be concerned with prime fields, analysis in this section applies to non-prime fields as well. The second concept is the Chinese remainder theorem, presented in the form of quotient rings. Lastly we combine the two concepts to show how the quotient ring of a finite field modulo a cyclotomic polynomial splits into a direct product of smaller quotient rings.

3. Section 4 explores the algebraic number theory behind our ring of interest, which may be written as $\mathbb{Z}_p[X]/\Phi_n(X)$ where p is a prime and Φ_n is the n'th cyclotomic polynomial. This ring may also be phrased as the ring of integers $\mathbb{Z}[X]/\Phi_n(X)$ of the number field extension $\mathbb{Q}[X]/\Phi_n(X)$ modulo the prime ideal (p) of the ring of integers $\mathbb{Z}$ of the number field $\mathbb{Q}$. The ring phrased as such will be our item of study. First we'll present such rings for more general number field extensions, second we'll narrow focus to Galois extensions, and lastly we'll narrow focus to the said extension $(\mathbb{Q}[X]/\Phi_n(X))/\mathbb{Q}$.

4. Section 5 is focused on representing our ring of interest in a convenient form for efficient algorithmic implementation. Using the results of Section 4 we are able to represent the ring $\mathbb{Z}_p[X]/\Phi_n(X)$ by recursively decomposing it into components until they become irreducible. Conveniently indexing the components is the core topic of the section, and we provide three solutions. The first solution indexes the components using quotient groups in terms of a certain class of subgroup of $\mathbb{Z}_n^\times$. The second solution indexes components via analogous subgroups of the Galois group. The third solution proves the subgroups of interest in the previous two solutions have generators, yielding simpler indexing schemes we will use in our algorithms.

5. Section 6 explores how one may architect a GPU algorithm for applying the Chinese remainder theorem to decompose a cyclotomic polynomial into its irreducible components. We do not yet concern in this section with multiplying the irreducible components by those of another polynomial. We perform decomposition in three steps. First we read the coefficient values from global memory into thread registers. Second we perform Chinese remainder

theorem decomposition *within* threads, meaning only values within the same thread interact. Third we complete decomposition *across* threads, meaning only values in different threads interact.

6. Section 7 improves and extends the initial GPU algorithm from Section 6. We apply the Chinese remainder theorem to decompose a cyclotomic polynomial into irreducible components, but compared to the initial algorithm we increase the amount of decomposition that occurs *within* threads, and decrease the amount that occurs *across* threads. We discuss arranging the resulting irreducible components in a layout suitable for multiplication with the irreducibles of another polynomial. Finally we turn to the multiplication of irreducible components. We end up performing multiplication of each irreducible component entirely within a single thread. Doing so limits the size of the irreducible components we can multiply, but simplifies our options for multiplication. For our motivational purposes the irreducible components are small enough to pose no issue. We explore schoolbook multiplication, minimal time Karatsuba multiplication, and minimal space Karatsuba multiplication.

2 Cyclotomic extensions

We introduce cyclotomic extensions and examine the special cases of extending $\mathbb{Q}$ and $\mathbb{F}_q$, the two most common base fields for such extensions. We also introduce cyclotomic polynomials and explore some basic properties relevant to cyclotomic extensions. Two general sources used but not cited elsewhere are [Mor96] and [Con].

- Let ϕ denote Euler's totient function, that is $\phi(n)$ counts the numbers between 1 and n that are coprime with n.
- Let $\operatorname{ord}_n(x)$ denote the order of element $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ in the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^\times$.
- Let the notation (a, b) for $a, b \in \mathbb{Z}$ denote the greatest common divisor of a and b, that is $\gcd(a, b)$.

For positive integer n, the n'th complex roots of unity are the roots of the polynomial $X^n - 1$ over $\mathbb{C}$, namely $\zeta_k = e^{2\pi i(k/n)}$ for $1 \leq k \leq n$ which form a cyclic group of order n. The *primitive* n'th complex roots of unity are those ζ_k for which $(k, n) = 1$, of which there are $\phi(n)$ by definition of ϕ.

We may extend the notion of complex roots of unity to roots of unity over any field in which $X^n - 1$ has n distinct roots. These roots are the n'th roots of unity denoted $\mu_n = \{\zeta_k\}_{k\in[n]}$. They form a cyclic subgroup of the field's multiplicative group with generators any roots of $X^n - 1$ not roots of $X^m - 1$ for any $m < n$. If $\zeta \in \mu_n$ is a generator, then $\zeta^k \in \mu_n$ has order $n/(n, k)$ and is therefore also a generator precisely when k and n are coprime. There are $\phi(n)$ generators of μ_n by definition of ϕ. We call these generators the *primitive* n'th roots of unity.

2.1 Extensions in general

After presenting the definition of cyclotomic extensions we immediately show such extensions are Galois. The Galois groups of n'th cyclotomic extensions are then shown isomorphic to subgroups of $(\mathbb{Z}/n\mathbb{Z})^\times$. At this point we introduce cyclotomic polynomials which we use in the following two sections, but before proceeding to those sections we prove two central properties of cyclotomic polynomials. The first is the fact that all cyclotomic polynomials have integer coefficients, and as such can be interpreted over any field. The second shows that all n'th cyclotomic polynomials are equivalent to the extent of such interpretation.

Cyclotomic extensions are had by adjoining a complete set of n'th roots of unity to a field. A central assumption we carry throughout exploration of cyclotomic extensions is that if the characteristic of the field is non-zero (i.e. prime), then it does not divide n, otherwise the field cannot hold all n'th roots of unity. To see this, suppose the field has characteristic p and $n = m \cdot p^\ell$. Since p is prime we apply the Frobenius endomorphism to see $X^n - 1 = (X^m - 1)^{p^\ell}$. Then for any $\zeta \in \mu_n$, $\zeta^n = 1$ implies $\zeta^m = 1$ with $m < n$ and therefore no element of μ_n can function as a primitive n'th root of unity in the field. So if we are to assume μ_n behaves as a complete set of n'th roots of unity in a field we must assume the field has characteristic zero or characteristic not dividing n. Henceforth we implicitly make this assumption.

Definition 1. *An n'th* cyclotomic *extension of a field F is obtained by adjoining a set of n'th roots of unity μ_n to F. The roots of unity μ_n may belong to any extension field of F. The field $F(\mu_n)$ is called an n'th cyclotomic field.*

Note that adjoining all n'th roots μ_n is equivalent to adjoining any generator of the set μ_n, that is any primitive n'th root of unity.

Consider the unique polynomial vanishing on exactly the n distinct elements μ_n, that is the polynomial $X^n - 1$. Since $F(\mu_n)$ contains all n roots of $X^n - 1$ (that is μ_n), it's clear that $X^n - 1$ splits in $F(\mu_n)$. Moreover, since the n elements of μ_n are distinct, $X^n - 1$ splits into distinct linear factors in $F(\mu_n)$. With such splitting in mind we now show the extension $F(\mu_n)/F$ is Galois.

Theorem 1 ($F(\mu_n)/F$ is Galois)**.** *For a field F, the extension $F(\mu_n)/F$ is Galois.*

Proof. We show normality and separability of this extension, that is the criteria for an extension to be Galois. In both parts we use the following lemma, the proof of which is not relevant for our purposes. Variations of the lemma can be found in many introductory texts on field theory (e.g. [Mit]).

> Suppose E/F is an algebraic field extension. If the minimal polynomial over F of *some* $\alpha_0 \in E$ splits in E, then the minimal polynomial over F of *every* $\alpha_i \in E$ splits in E. Moreover, if the minimal polynomial of α_0 is separable, then the minimal polynomial of *every* α_i is separable.

Fix some $\zeta \in \mu_n$. The minimal polynomial over F of ζ is a factor of $X^n - 1$. To see this, suppose dividing $X^n - 1$ by the minimal polynomial f_ζ of ζ leaves non-zero remainder r_ζ. Then r_ζ must be of degree less than f_ζ and vanish on ζ (since both $X^n - 1$ and f_ζ vanish on ζ). As such, r contradicts the minimality of f_ζ.

Normality: Since $X^n - 1$ splits in $F(\mu_n)$, so does its factors, including the minimal polynomial over F of $\zeta \in F(\mu_n)$. Applying the lemma above we conclude the minimal polynomial over F of *every* element in $F(\mu_n)$ splits in $F(\mu_n)$, and thus the extension $F(\mu_n)/F$ is normal.

Separability: Since $X^n - 1$ splits into *distinct* linear factors in $F(\mu_n)$, so does its factors, including the minimal polynomial over F of $\zeta \in F(\mu_n)$. Therefore the minimal polynomial of

ζ is separable. Applying the lemma above we conclude the minimal polynomial over F of *every* element in $F(\mu_n)$ is separable, and thus the extension $F(\mu_n)/F$ is separable. $\square$

With $F(\mu_n)/F$ Galois, what can we say about its Galois group? The Galois group of a Galois extension $F(\mu_n)/F$ is the group of automorphisms on $F(\mu_n)$ that fix F. Since the field $F(\mu_n)$ is composed of F and μ_n, an automorphism on $F(\mu_n)$ that fixes F is defined solely by its behavior on μ_n. For every $\sigma \in \text{Gal}(F(\mu_n)/F)$, the next theorem shows this behavior to be $\sigma(\zeta) = \zeta^k$ for all $\zeta \in \mu_n$ and some k coprime with n. Thus σ may be defined piecewise on $F(\mu_n)$ as

$$\sigma(x) = \begin{cases} x^k, & x \in \mu_n \\ x, & x \in F \end{cases}$$

Theorem 2 (Gal$(F(\mu_n)/F)$ is isomorphic to a subgroup of $(\mathbb{Z}/n\mathbb{Z})^\times$)**.** *Each automorphism of Gal$(F(\mu_n)/F)$ is determined by some integer coprime with n. Let σ_k denote the automorphism determined by integer k with $(k, n) = 1$. Then $\sigma_k(\zeta) = \zeta^k$ for all $\zeta \in \mu_n$.*

The map $\sigma_k \to (k + \mathbb{Z})$ of signature Gal$(F(\mu_n)/F) \to (\mathbb{Z}/n\mathbb{Z})^\times$ is an injective group homomorphism. Therefore, Gal$(F(\mu_n)/F)$ is isomorphic to a subgroup of $(\mathbb{Z}/n\mathbb{Z})^\times$.

Proof. Fix some primitive root of unity $\zeta_0 \in \mu_n$, and recall that all generators of μ_n have the form ζ_0^k for some k coprime with n.

Since the automorphisms Gal$(F(\mu_n)/F)$ of the field $F(\mu_n)$ fix the subfield F, they necessarily map $F(\mu_n) \setminus F = \mu_n$ to itself. Moreover, since they are automorphisms of the multiplicative group $F(\mu_n)^\times$, they are group automorphisms of μ_n. As any automorphism of the group μ_n must map generators to generators, for $\sigma \in \text{Gal}(F(\mu_n)/F)$ we must have $\sigma(\zeta_0) = \zeta_0^k$ for some k coprime with n. With any element of μ_n taking the form ζ_0^t for some t, we have

$$\sigma(\zeta_0^t) = \sigma(\zeta_0)^t = (\zeta_0^k)^t = (\zeta_0^t)^k$$

Thus σ is defined by raising all elements of μ_n to k. We associate σ with k and write σ_k. To see the subscript notation σ_k is unique modulo n, suppose $\sigma_{k'} = \sigma_k$. Then $\zeta_0^{k-k'} = 1$ and since ζ_0 has order n, it must be that n divides $k - k'$.

We now show the map $\sigma_k \to (k+\mathbb{Z})$ is an injective group homomorphism from Gal$(F(\mu_n)/F)$ to $(\mathbb{Z}/n\mathbb{Z})^\times$. The fact that $(k, n) = 1$ for all σ_k implies the image of σ_k is contained in $(\mathbb{Z}/n\mathbb{Z})^\times$. For $\sigma_k, \sigma_{k'} \in \text{Gal}(F(\mu_n)/F)$ and $\zeta \in \mu_n$ we have

$$\sigma_k \circ \sigma_{k'}(\zeta) = \sigma_k(\sigma_{k'}(\zeta)) = \zeta^{kk'} = \sigma_{kk'}$$

Denoting the map by ψ we readily see it is a homomorphism as

$$\psi(\sigma_k \circ \sigma_{k'}) = \psi(\sigma_{kk'}) = kk' = \psi(\sigma_k) \cdot \psi(\sigma_{k'})$$

The kernel of the homomorphism is all σ_k such that $k \equiv 1 (\bmod n)$ in which case σ_k is the identity function, that is the identity element of the Galois group. The homomorphism is therefore injective. $\square$

We now introduce cyclotomic polynomials for their roles in cyclotomic extensions of $\mathbb{Q}$ and $\mathbb{F}_q$ as explored in the next two sections. The n'th cyclotomic polynomial is defined as the monic, non-zero polynomial vanishing on the primitive n'th roots of unity in μ_n. Suppose $\zeta_0 \in \mu_n$ is a primitive root of unity.

$$\Phi_n(X) := \prod_{\substack{n\text{'th primitive} \\ \text{roots of unity } \zeta}} (X - \zeta) = \prod_{\substack{1 \le k \le n \\ (k,n)=1}} (X - \zeta_0^k)$$

Since there are $\phi(n)$ primitive n'th roots of unity, Φ_n has degree $\phi(n)$.

We may partition the n'th roots of unity into all sets of primitive d'th roots of unity, one set for every d dividing n. To see this, note that every n'th root of unity generates a subgroup of order d and is therefore a primitive d'th root of unity for exactly one d dividing n. On the other hand, since d divides n every primitive d'th root of unity is also an n'th root of unity. With this partition in mind we may represent $X^n - 1$ as the product of all cyclotomic polynomials $\Phi_d(X)$ for d a divisor of n.

$$X^n - 1 = \prod_{d|n} \Phi_d(X)$$

We may regard this equation as an implicit, recursive definition of cyclotomic polynomials, written explicitly as

$$\Phi_n(X) = \big(X^n - 1\big) \Big/ \prod_{\substack{d|n \\ d \neq n}} \Phi_d(X)$$

A potentially surprising property of cyclotomic polynomials is that they have integer coefficients. Indeed, regardless the n'th roots of unity μ_n used to define Φ_n, even if the *irrational* complex roots of unity, expanding the polynomial Φ_n into coefficient form always yields integer coefficients. Note that integer coefficients can alternatively be interpreted as belonging to any field by identifying them with multiples of the field's additive identity. Thus when considering Φ_n over $F(\mu_n)$ we may say Φ_n has coefficients over F.

Theorem 3 ($\Phi_n[X] \in \mathbb{Z}[X]$)**.** *The n'th cyclotomic polynomial Φ_n has integer coefficients.*

Proof. We show this by induction using the recursive definition of cyclotomic polynomials. The base case in clear with $\Phi_1(X) = X - 1 \in \mathbb{Z}[X]$. For induction consider the equation

$$X^n - 1 = \Phi_n(X) \cdot \prod_{\substack{d|n \\ d \neq n}} \Phi_d(X) = \Phi_n(X) \cdot \gamma(X)$$

The polynomial $X^n - 1$ has integer coefficients, and by induction so does γ. Note that γ is also monic as it is the product of monic polynomials. Then one may observe that in dividing $X^n - 1$ by γ, the long division algorithm in $\mathbb{Z}[X]$ must yield a polynomial also in $\mathbb{Z}[X]$. For a more explicit argument, suppose the long division algorithm in $\mathbb{Z}[X]$ yields quotient $q \in \mathbb{Z}[X]$ and remainder $r \in \mathbb{Z}[X]$ such that $X^n - 1 = q(X)\gamma(X) + r(X)$. Then $q(X)\gamma(X) + r(X) = \Phi_n(X)\gamma(X)$ forces $r(X) = \gamma(X)(\Phi_n(X) - q(X))$ to be zero, otherwise $\deg(r) \ge \deg(\gamma)$. Therefore, $\Phi_n = q \in \mathbb{Z}[X]$. $\square$

We can strengthen the previous result further. We can in fact say that all cyclotomic polynomials are equivalent in the sense that two n'th cyclotomic polynomials defined over different fields share the same roots when interpreted over the same field. As mentioned previously, such cross-field interpretation is possible because cyclotomic polynomials have integer coefficients. As a result, a lot can be said about cyclotomic polynomials oblivious to the field over which they're defined.

We demonstrate equivalence between n'th cyclotomic polynomials by showing they are all equivalent to the n'th cyclotomic polynomial over $\mathbb{C}$. Consider the n'th cyclotomic polynomial in $\mathbb{C}[X]$ defined as vanishing exactly on $e^{2\pi i(k/n)}$ for $1 \leq k \leq n$ with $(n, k) = 1$. Interpreting this polynomial over any n'th cyclotomic field $F(\mu_n)$, we argue it vanishes on the primitive n'th roots of unity in μ_n. Therefore, the following two polynomials are equal:

- The n'th cyclotomic polynomial defined over $F(\mu_n)$ as vanishing on exactly the primitive n'th roots of unity in μ_n.
- The n'th cyclotomic polynomial defined over $\mathbb{C}$ and interpreted over $F(\mu_n)$.

By the transitivity of equivalence, we may then say any two cyclotomic polynomials defined over different fields are equivalent 'up to interpretation.'

Theorem 4 ($\Phi_n \in \mathbb{C}[X] \equiv \Phi_n \in F(\mu_n)[X]$)**.** *Let Φ_n be the n'th cyclotomic polynomial in $\mathbb{C}[X]$. Let $F(\mu_n)$ be an n'th cyclotomic field. Then $\overline{\Phi_n} \in F(\mu_n)[X]$ vanishes on precisely the primitive n'th roots of unity in μ_n, where $\overline{\Phi_n}$ is Φ_n interpreted over $\mathbb{F}(\mu_n)$.*

Proof. Let $\overline{x}$ for $x \in \mathbb{Z}[X]$ denote $x \in F[X]$, that is x from $\mathbb{Z}[X]$ interpreted in $F[X]$. Suppose division of $X^n - 1$ by Φ_n in $\mathbb{Z}[X]$ yields quotient q and remainder r. Then we may write $\overline{X^n - 1} = \overline{\Phi_n}(X) \cdot \overline{q}(X) + \overline{r}(X)$ with relations $\deg(\overline{r}) < \deg(\overline{\Phi_n}) = \phi(n)$ still holding. That is, as Φ_n divides $X^n - 1$ in $\mathbb{Z}[X]$, so $\overline{\Phi_n}$ divides $\overline{X^n - 1} = X^n - \overline{1}$ in $F[X]$. Note that $X^n - \overline{1}$ splits in $F(\mu_n)$ with roots μ_n, and therefore its factor $\overline{\Phi_n}$ also splits in $F(\mu_n)$ with roots some subset of μ_n. Since the subset must be of size $\phi(n) = \deg(\overline{\Phi_n})$, we are left to show that all roots of $\overline{\Phi_n}$ are primitive n'th roots of unity.

Supposing $\overline{\Phi_n}(\zeta) = 0$ for $\zeta \in \mu_n$, we must show ζ is a primitive n'th root of unity. We will refer to the following equation as the 'cyclotomic decomposition' of $\zeta^m - \overline{1}$ for some m.

$$\zeta^m - \overline{1} = \prod_{d|m} \overline{\Phi_d}(\zeta)$$

Suppose ζ is not a primitive n'th root of unity. Then ζ is a primitive m'th root of unity for some proper divisor m of n. The cyclotomic decomposition of $\zeta^m - \overline{1}$ suggests $\overline{\Phi_d}(\zeta) = 0$ for some $d|m$ which is a proper divisor of n. We now contradict this statement by recalling that $X^n - \overline{1}$ has no repeated roots, and therefore ζ can only vanish on one of its factors. That is, $\overline{\Phi_d}(\zeta) = 0$ can only occur for one $d|n$ in the cyclotomic decomposition of $\zeta^n - \overline{1}$. Since we assume $\overline{\Phi_n}(\zeta) = 0$, it must be that $\overline{\Phi_d}(\zeta) \neq 0$ for all proper divisors d of n. □

2.2 Extensions of $\mathbb{Q}$

First we examine the reducibility of Φ_n over $\mathbb{Q}$, then see what it means for the Galois group $\mathbb{Q}(\mu_n)/\mathbb{Q}$. We also mention the ring of integers of $\mathbb{Q}(\mu_n)$. Note that since the field $\mathbb{Q}$ has characteristic zero, we have met the assumption stated at the start of the prior section to ensure μ_n functions as a full set of n'th roots of unity over $\mathbb{Q}$.

Theorem 5 (Φ_n is irreducible over $\mathbb{Q}$)**.** *The n'th cyclotomic polynomial Φ_n is irreducible over $\mathbb{Q}$.*

Proof. See [Wei] for several of the most historical proofs. Here we give a proof that goes back as far as Dedekind in 1857 ([Mil22]).

Suppose $\Phi_n(X) = f(x)g(x) \in \mathbb{Z}[X]$ with f irreducible over $\mathbb{Q}$. We will prove that for all primes p not dividing n, if primitive n'th root of unity ζ is a root of f, then so is ζ^p. Therefore, for all $\phi(n)$ values k with $(k, n) = 1$ we can say ζ^k is also a root of f, and thus f has degree $\phi(n)$ so it must be Φ_n itself. To justify the latter conclusion, consider how any k has a prime decomposition, and to say $(k, n) = 1$ is to say no prime in the decomposition divides n. To show f vanishes on ζ^k, we raise ζ consecutively to each prime in the decomposition. Each time we do so, we are raising a primitive n'th root of unity on which f vanishes to a prime not dividing n, so we may conclude the result is another primitive n'th root of unity on which f vanishes.

Now we prove that for prime p not diving n, ζ^p is a root of f. Since ζ^p is an n'th primitive root of unity, we have $\Phi_n(\zeta^p) = 0$ in which case either $f(\zeta^p) = 0$ or $g(\zeta^p) = 0$. Suppose the latter for sake of contradiction. Since f is irreducible, and we assume it vanishes on ζ, it must be the minimal polynomial of ζ over $\mathbb{Q}$ and so must divide $g(x^p)$. Now f and g have rational coefficients because they are products of the minimal polynomials of the primitive n'th roots of unity over $\mathbb{Q}$. Then by a variant of what's known as Gauss' Lemma , by the fact that $f(x)g(x) = \Phi_n(x)$ it must be that f and g and thus $g(x^p)$ have integer coefficients. With quotient $q \in \mathbb{Q}[X]$, $f \in \mathbb{Z}[X] \subseteq \mathbb{Q}[X]$ and $g(x^p) \in \mathbb{Z}[X]$, we apply Gauss' Lemma again to the equation $f(x)q(x) = g(x^p)$ and conclude $q \in \mathbb{Z}[X]$. We may then pass the equation $f(x)q(x) = g(x^p)$ through the homomorphism of reducing modulo p with signature $\mathbb{Z}[X] \to \mathbb{Z}_p[X]$. With $\overline{g}(x^p) = \overline{g}(x)^p$ by the Frobenius automorphism, we have $\overline{f}(\zeta)\overline{q}(\zeta) = \overline{g}(\zeta)^p = 0$. Considering the equation $\overline{g}(\zeta)^p = 0$ in $\mathbb{C}$ we conclude $\overline{g}(\zeta) = 0$ since $\mathbb{C}$ has no zero divisors.

Now $\overline{\Phi_n}$ divides $\overline{X^n - 1} = X^n - \overline{1}$ and $\overline{\Phi_n} = \overline{f}(X)\overline{g}(X)$, so we may write

$$X^n - \overline{1} = \frac{X^n - \overline{1}}{\overline{\Phi_n}(X)} \cdot \overline{f}(X) \cdot \overline{g}(X)$$

By $\overline{f}(\zeta) = \overline{g}(\zeta) = 0$ we see ζ is a double root of $X^n - \overline{1}$. But $X^n - \overline{1}$ has non-zero derivative nX^n since we assume p doesn't divide n, and therefore $X^n - \overline{1}$ has no repeated roots. Thus a contradiction. □

The irreducibility of Φ_n over $\mathbb{Q}$ allows us to determine the degree of the extension $\mathbb{Q}(\mu_n)/\mathbb{Q}$. Recall that Φ_n over $\mathbb{Q}(\mu_n)$ has $\phi(n)$ roots, namely the primitive n'th roots of unity. Since Φ_n is irreducible over $\mathbb{Q}$, it must be the minimal polynomial for these primitive n'th roots of unity. Therefore the extension $\mathbb{Q}(\mu_n)/\mathbb{Q}$ has degree $\phi(n)$. To see this, note that the field $\mathbb{Q}(\mu_n)$ is equivalent to the field $\mathbb{Q}[X]/\Phi_n(X)$ with equality had by identifying X with any primitive n'th root of unity. The field $\mathbb{Q}[X]/\Phi_n(X)$ extends $\mathbb{Q}$ with degree $\deg(\Phi_n) = \phi(n)$ and the claim follows.

The identity $[\mathbb{Q}(\mu_n) : \mathbb{Q}] = \phi(n)$ allows us to now fully determine the Galois group.

Corollary 1 (Gal$(\mathbb{Q}(\mu_n)/\mathbb{Q}) \cong (\mathbb{Z}/n\mathbb{Z})^\times$)**.** *The Galois group Gal$(\mathbb{Q}(\mu_n)/\mathbb{Q})$ is isomorphic to the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^\times$.*

Proof. We will not prove it here, but a basic fact of Galois theory is that the order of the Galois group equals the degree of the Galois extension. Having just established that $\mathbb{Q}(\mu_n)/\mathbb{Q}$ has degree $\phi(n)$, we conclude

$$\left|\text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})\right| = \left[\mathbb{Q}(\mu_n) : \mathbb{Q}\right] = \phi(n)$$

We showed in Theorem 2 that Gal$(\mathbb{Q}(\mu_n)/\mathbb{Q})$ is isomorphic to a subgroup of $(\mathbb{Z}/n\mathbb{Z})^\times$. Since the group $(\mathbb{Z}/n\mathbb{Z})^\times$ has order $\phi(n)$ we conclude the subgroup must be $(\mathbb{Z}/n\mathbb{Z})^\times$ itself. □

Before moving on to the case of finite fields, we mention another important theorem of cyclotomic extensions of $\mathbb{Q}$. The ring of integers of the field $\mathbb{Q}(\mu_n)$ is the ring $\mathbb{Z}[\mu_n]$. The ideals of the ring $\mathbb{Z}[\mu_n]$ are the 'ideal lattices' at the foundation of modern lattice cryptography. Note that $\mathbb{Z}[\mu_n]$ is equivalent to $\mathbb{Z}[X]/\Phi_n(X)$.

Theorem 6 ($\mathcal{O}_{\mathbb{Q}(\mu_n)} = \mathbb{Z}[\mu_n]$)**.** *The ring of integers of* $\mathbb{Q}(\mu_n)$ *is* $\mathbb{Z}[\mu_n]$.

Proof. The proof is involved, see [Ngu]. □

2.3 Extensions of $\mathbb{F}_q$

In opposite order of the previous section, we first examine the Galois group $\mathbb{F}_q(\mu_n)/\mathbb{F}$, then see what it means for the reducibility of Φ_n over $\mathbb{F}_q$. Since finite fields have non-zero characteristic, recall that if μ_n is to function as a full set of n'th roots of unity in a finite field, the field characteristic cannot divide n. We continue to implicitly make this assumption below.

We already know that Gal$(F_q(\mu_n)/F_q)$ is isomorphic to a subgroup of $(\mathbb{Z}/n\mathbb{Z})^\times$. We now argue this subgroup in generated by q.

Theorem 7 (Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ is isomorphic to $\langle q+\mathbb{Z}\rangle \subseteq (\mathbb{Z}/n\mathbb{Z})^\times$)**.** *For prime power q, the group Gal*$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ *is isomorphic to the subgroup* $\langle q+\mathbb{Z}\rangle$ *of* $(\mathbb{Z}/n\mathbb{Z})^\times$ *and thus has order* $\mathrm{ord}_n(q)$.

Proof. We will show the q'th power map is an element of the Galois group, and then show the q'th power map in fact generates the Galois group. Using this knowledge we determine the image of the Galois group in $(\mathbb{Z}/n\mathbb{Z})^\times$.

Before proceeding we mention the Frobenius map. Suppose $q = p^\ell$. The Frobenius map is the p'th power map, which is an endomorphism of $\mathbb{F}_q(\mu_n)$. In fact, it is an automorphism of $\mathbb{F}_q(\mu_n)$ because it is injective and $\mathbb{F}_q(\mu_n)$ is finite. To see injectivity, note the kernel is all $x \in \mathbb{F}_q(\mu_n)$ such that $x^p = 0$, and since $\mathbb{F}_q(\mu_n)$ contains no zero-divisors this only holds for $x = 0$.

To show that the q'th power map belongs to Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$, we show it is an automorphism fixing the field precisely equal to $\mathbb{F}_q$. The map is an automorphism because it is the ℓ-depth composition of the Frobenius map, itself an automorphism. The q'th power map clearly fixes the additive identity of $\mathbb{F}_q$, and it also fixes the multiplicative group $\mathbb{F}_q^\times$ because group order $q-1$ implies $x^{q-1} = 1$ for $x \in \mathbb{F}_q^\times$. Thus the q'th power map fixes $\mathbb{F}_q$, and to see that it *only* fixes the q elements of $\mathbb{F}_q$, note that $X^q - X$ has no more than q roots. Furthermore, we can say the same about all automorphisms generated by the q'th power map. That is, the subgroup of Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ generated by the q'th power map also fixes $\mathbb{F}_q$.

We now invoke the fundamental theorem of Galois theory to argue that two different subgroups of Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ must fix different fields. By definition, the goup Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ fixes the field $\mathbb{F}_q$, but so does the subgroup generated by the q'th power map. Therefore, these groups are in fact the same, that is the q'th power map generates Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$.

Finally we determine the image of the Galois group in $(\mathbb{Z}/n\mathbb{Z})^\times$. Recall that the isomorphism from Gal$(\mathbb{F}_q(\mu_n)/\mathbb{F}_q)$ to a subgroup of $(\mathbb{Z}/n\mathbb{Z})^\times$ takes the form $\sigma_k \to (k+\mathbb{Z})$ where σ_k is the k'th power map on μ_n and the identity map on $\mathbb{F}_q$. To say the q'th power map generates the Galois group is to say σ_q generates the Galois group. Since σ_q maps to $(q+\mathbb{Z})$, we may conclude that the image is generated by $(q + \mathbb{Z})$. Therefore the Galois group maps to the subgroup $\langle q + \mathbb{Z} \rangle$ of $(\mathbb{Z}/n\mathbb{Z})^\times$, which has order $\mathrm{ord}_n(q)$. $\square$

Again invoking the fact that the order of the Galois group is equal to the degree of the Galois extension, we conclude the extension $\mathbb{F}_q(\mu_n)/\mathbb{F}_q$ has degree $\mathrm{ord}_n(q)$. Now we utilize this fact to examine how Φ_n splits in the field $\mathbb{F}_q$.

Theorem 8 (Reducibility of Φ_n over $\mathbb{F}_q$)**.** *Over the field $\mathbb{F}_q$ the polynomial Φ_n splits into $\phi(n)/\mathrm{ord}_n(q)$ monic, irreducible factors each of degree $\mathrm{ord}_n(q)$.*

Proof. By definition, Φ_n over $\mathbb{F}_q(\mu_n)$ has degree $\phi(n)$ and vanishes on the $\phi(n)$ primitive n'th roots of unity. Therefore every irreducible factor of Φ_n over $\mathbb{F}_q$ is the minimal polynomial of one or more primitive n'th roots of unity.

Fix a primitive n'th root of unity ζ with minimal polynomial f_ζ (an irreducible factor of Φ_n over $\mathbb{F}_q$). The field $\mathbb{F}_q(\zeta)$ is a cyclotomic extension of $\mathbb{F}_q$. As implied by the previous theorem, the extension $\mathbb{F}_q(\zeta)/\mathbb{F}_q$ has degree $\mathrm{ord}_n(q)$. But this extension is equivalent to $\mathbb{F}_q[X]/f_\zeta(X)$ which has degree $\deg(f_\zeta)$. Therefore all irreducible factors of Φ_n over $\mathbb{F}_q$ have degree $\mathrm{ord}_n(q)$. Since Φ_n has degree $\phi(n)$, there are $\phi(n)/\mathrm{ord}_n(q)$ such factors. $\square$

We restate the previous theorem for the special case in which Φ_n is irreducible over $\mathbb{F}_q$.

Corollary 2 (Irreducibility of Φ_n over $\mathbb{F}_q$)**.** *In the field $\mathbb{F}_q$ the polynomial Φ_n is irreducible if and only if $\mathrm{ord}_n(q) = \phi(n)$.*

3 Reducibility and CRT

This section is dedicated to presenting two auxiliary notions that will play a role in investigating Section 4. In the first section we present how cyclotomic polynomials split over finite fields. In the second section we present the Chinese Remainder Theorem. In the third section we show how the tools from the previous two sections can be combined, though at the moment in this project we have no relevant context in which to apply this result.

3.1 Representation of Φ_n over $\mathbb{F}_q$

In this section we introduce the Möbius function along with its inversion formula, then show how it can be used to represent the n'th cyclotomic polynomial. Using this representation, we can represent one cyclotomic polynomial in terms of another. We exploit this relationship to arrive at our primary result for this section, that is how cyclotomic polynomials can split over finite fields in any number of related ways (the factors not always irreducible). We now proceed through these parts one after the other with no interleaving commentary.

Definition 2 (Möbius function)**.** *Let whole number n have prime decomposition $\prod_{i=1}^{\ell} p_k^{e_k}$ where $e_k > 0$ for all k. That is, k enumerates the ℓ prime factors of n. Then the möbius function of n is given as*

$$\mu(n) = \begin{cases} (-1)^\ell, & \forall k, e_k = 1 \\ 0, & \exists k, e_k > 1 \end{cases}$$

Lemma 1 (Möbius inversion formula)**.** *For a function f consider the following definitions for all integers $n > 0$.*

$$F_+(n) \coloneqq \sum_{d|n} f(d),\ F_\times(n) \coloneqq \prod_{d|n} f(d)$$

Then the Möbius inversion formula (which we will not prove) is

$$f(n) = \sum_{d|n} \mu(d) \cdot F_+\left(\frac{n}{d}\right) = \prod_{d|n} F_\times\left(\frac{n}{d}\right)^{\mu(d)}$$

Note that for every $d|n$ with quotient q, we also have $q|n$ with quotient d. The formulas may thus be rewritten as

$$f(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) \cdot F_+(d) = \prod_{d|n} F_\times(d)^{\mu(n/d)}$$

Lemma 2 (Φ_n by Möbius inversion)**.**

$$\Phi_n(X) = \prod_{d|n} \left(X^{n/d} - 1\right)^{\mu(d)} = \prod_{d|n} \left(X^d - 1\right)^{\mu(n/d)}$$

Proof. Recall from Section 2 that Φ_n may be expressed as

$$X^n - 1 = \prod_{d|n} \Phi_d(X)$$

The results follow by applying the Möbius inversion formula with $f(d)$ corresponding to Φ_d and $F_\times(n)$ corresponding to $X^n - 1$. □

Theorem 9 ($\Phi_n(X) = \Phi_m(X^{n/m})$)**.** *Suppose n and m have prime decompositions $\prod_k p_k^{e_k}$ and $\prod_k p_k^{f_k}$ respectively, with $e_k \geq f_k \geq 1$ for all k. Then $\Phi_n(X) = \Phi_m(X^{n/m})$.*

Proof. If $d|n$ but $d \nmid m$ then with prime decomposition $d = \prod_k p_k^{g_k}$ there exists some k such that $g_k > f_k \geq 1$, in which case $\mu(d) = 0$. Using Möbius inversion for the first and last equations, we write

$$\begin{aligned}
\Phi_n(X) &= \prod_{d|n} (X^{n/d} - 1)^{\mu(d)} \\
&= \prod_{d|n,\ d|m} ((X^{n/m})^{m/d} - 1)^{\mu(d)} \prod_{d|n,\ d\nmid m} (X^{n/d} - 1)^0 \\
&= \Phi_m(X^{n/m})
\end{aligned}$$

□

Corollary 3 (Reducibility of Φ_n over $\mathbb{F}_q$ with respect to m)**.** *Suppose n and m have prime decompositions $\prod_k p_k^{e_k}$ and $\prod_k p_k^{f_k}$ respectively, with $e_k \geq f_k \geq 1$ for all k. With q some prime power, suppose $q \equiv 1 (\mathrm{mod} m)$.*

Then Φ_n over $\mathbb{F}_q$ splits into $\phi(m)$ monic factors each of degree n/m in the following form:

$$\Phi_n(X) = \prod_{k=1}^{\phi(m)} \left(X^{n/m} - \zeta_k\right)$$

where the ζ_k are the $\phi(m)$ primitive m'th roots of unity in the group $\mathbb{F}_q^{\times}$. Moreover, these factors are irreducible if and only if $\mathrm{ord}_n(q) = n/m$, in which case $\phi(m) = \phi(n)/\mathrm{ord}_n(q)$.

Proof. Consider the finite field $\mathbb{F}_q$, with cyclic multiplicative group of order $q-1$. Since m divides $q-1$ by assumption, the multiplicative group must have a cyclic subgroup of order m, containing the primitive m'th roots of unity $\{\zeta_k\}_{k\in[\phi(m)]}$, meaning $\zeta_k^i = 1$ if and only if $i \equiv 0 (\bmod m)$. We may then use the definition of the m'th cyclotomic polynomial over $\mathbb{F}_q$ to write

$$\Phi_m(X) = \prod_{k=1}^{\phi(m)} (X - \zeta_k)$$

Noting our assumptions meet the criteria, we may apply the transformation $\Phi_n(X) = \Phi_m(X^{n/m})$ from Theorem 9 to get

$$\Phi_n(X) = \prod_{k=1}^{\phi(m)} \left(X^{n/m} - \zeta_k\right)$$

Now the reducibility of Φ_n over $\mathbb{F}_q$ from Theorem 8 asserts that Φ_n splits into $\phi(n)/\mathrm{ord}_n(q)$ monic, irreducible factors each of degree $\mathrm{ord}_n(q)$. In the equation above, Φ_n has split into $\phi(m)$ factors each of degree n/m. Then these factors are irreducible precisely when $\mathrm{ord}_n(q) = n/m$. □

3.2 Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) was originally used in the context of modular arithmetic. Here we first describe a more general version of the Chinese Remainder Theorem in the context of arbitrary commutative rings, but restricted to the case of two ideals. Then we generalize from the case of two ideals to the case of any finite number of ideals. Keep in mind we are presenting the Chinese Remainder Theorem as something of independent interest from the representation of cyclotomic polynomials in the first section.

Lemma 3 (Binary Chinese Remainder Theorem)**.** *For commutative ring R, let $I, J \subseteq R$ be co-prime ideals, meaning $I + J = R$. Then we have*

$$R/(IJ) \cong R/I \times R/J$$

with corresponding isomorphism $x + IJ \to (x + I, x + J)$.

Proof. Consider the natural homomorphism $\psi\colon R \to R/I \times R/J$ with $\psi(x) = (x+I, x+J)$. We will show that ψ has kernel $\ker(\psi) = IJ$ and image $\mathrm{img}(\psi) = R/I \times R/J$. Hence upon invoking what's called the 'First Isomorphism Theorem' for rings, we conclude

$$R/\ker(\psi) \cong \mathrm{img}(\psi) \implies R/(IJ) \cong R/I \times R/J$$

Since $I + J = R$, there must be $i \in I$ and $j \in J$ such that $i + j = 1$. We will exploit i and j below without reintroduction.

Kernel: The kernel is all $x \in R$ such that $x + I = I$ and $x + J = J$, that is the set $I \cap J$. For $x \in I \cap J$, we have $x(i + j) = xi + xj \in IJ$ as $ix, xj \in IJ$, and thus $I \cap J \subseteq IJ$. To show $IJ \subseteq I \cap J$ we don't need the fact the I and J are coprime. For $x = i_1 j_1 + \cdots + i_n j_n \in IJ$ with $i_k \in I$ and $j_k \in J$ we observe that each $i_k j_k \in I \cap J$ and since $I \cap J$ is itself an ideal, as an additive subgroup it is closed under addition, that way $x \in I \cap J$. We conclude $\ker(\psi) = I \cap J = IJ$.

Image: To prove $\operatorname{img}(\psi) = R/I \times R/J$ we must find a pre-image for any $(\alpha + I, \beta + J)$. The pre-image $\alpha i + \beta j$ satisfies as

$$\begin{aligned}
&\psi(\alpha i + \beta j)\\
&\qquad = (\alpha i + \beta j + I, \alpha i + \beta j + J)\\
&\qquad = (\beta j + I, \alpha i + J)\\
&\qquad = (\beta j + \beta i + I, \alpha i + \alpha j + J)\\
&\qquad = (\beta + I, \alpha + J)
\end{aligned}$$

Last we must show that the homomorphism $f\colon R/IJ \to R/I \times R/J$ sending $x + IJ$ to $\psi(x) = (x + I, x + J)$ is an isomorphism. Let $\pi\colon R \to R/IJ$ be the projection homomorphism $x \to x + IJ$. Then $f \circ \pi = \psi$, and since ψ is surjective it must be that f (and π) is surjective. The kernel of f is all $x + IJ$ such that $x \in I$ and $x \in J$, that is $I \cap J$ which we know (from examining the kernel of ψ) is equal to IJ. Since f is injective and surjective it is an isomorphism. □

Remark 1. *As is shown in the proof of the previous lemma, when two ideals I and J are coprime we have the equality $IJ = I \cap J$. The isomorphism for the binary case of the Chinese Remainder Theorem can then be written as*

$$R/(I \cap J) \cong R/I \times R/J$$

Similarly, for the general case shown in the following lemma we may write

$$R/(I_1 \cap \cdots \cap I_n) \cong R/I_1 \times \cdots \times R/I_n$$

Theorem 10 (Chinese Remainder Theorem)**.** *For commutative ring R, let $I_1, \ldots, I_n \subseteq R$ be mutually coprime ideals, meaning $I_j + I_k = R$ for all $1 \leq j < k \leq n$. Then we have*

$$R/(I_1 \ldots I_n) \cong R/I_1 \times \cdots \times R/I_n$$

with corresponding isomorphism $x + (I_1 \ldots I_n) \to (x + I_1, \ldots, x + I_n)$.

Proof. Two ideals are coprime if and only if their sum contains the multiplicative identity. Then by assumption $1 \in I_j + I_n$ for all $j \in [n-1]$, and in particular there exist $\alpha_j \in I_j,\ \beta_j \in I_n$ for all $j \in [n-1]$ such that $\alpha_j + \beta_j = 1$. Their product expands as

$$1 = \prod_{j=1}^{n-1} (\alpha_j + \beta_j) = \left(\prod_{j=1}^{n-1} \alpha_j\right) + \gamma \in \left(\prod_{j=1}^{n-1} I_j\right) + I_n$$

where $\gamma \in I_n$ because every term of γ contains some $\beta_j \in I_n$. Thus $(I_1 \dots I_{n-1})$ and I_n are coprime.

Then the binary case of the Chinese Remainder Theorem from Lemma 3 followed by induction implies

$$\begin{aligned} R/(I_1 \dots I_n) &= R/((I_1 \dots I_{n-1})I_n) \\ &\cong R/(I_1 \dots I_{n-1}) \times R/I_n \\ &\cong (R/I_1 \times \dots \times R/I_{n-1}) \times R/I_n \\ &\cong R/I_1 \times \dots \times R/I_n \end{aligned}$$

We may also apply the binary case of the Chinese Remainder Theorem to conclude the following homomorphism is an isomorphism

$$x + (I_1 \dots I_{n-1})I_n \to (x + (I_1 \dots I_{n-1}), x + I_n)$$

By induction $x + (I_1 \dots I_{n-1}) \to (x + I_1, \dots, x + I_{n-1})$ is an isomorphism, so composing with the previous isomorphism yields the desired isomorphism

$$x + (I_1 \dots I_{n-1}I_n) \to (x + I_1, \dots, x + I_{n-1}, x + I_n)$$

□

3.3 An example representing $\mathbb{F}_q[X]/\Phi_n(X)$

The tools developed in the other two sections are used Section 4 in this project, and as far as we have developed them so far they have no overlapping use. But before concluding this section we can illustrate how we may combine them in a straightforward and useful way. In particular, below we show how cyclotomic polynomials can be represented over finite fields using both the reducibility result shown in the first section, as well as the generalized Chinese Remainder Theorem.

Lemma 4 (Representation of $\mathbb{F}_q[X]/\Phi_n(X)$)**.** *Let $\mathbb{F}_q$ be a finite field of prime power order q, and suppose we have m and n as discussed in Corollary 3 regarding the reducibility of Φ_n over $\mathbb{F}_q$ with respect to m, but for the special case that $\operatorname{ord}_n(q) = n/m$. Then we have*

$$\mathbb{F}_q[X]/\Phi_n(X) = \mathbb{F}_q[X] \Big/ \prod_{k=1}^{\phi(m)} \left(X^{n/m} - \zeta_k\right) \cong \prod_{k=1}^{\phi(m)} \frac{\mathbb{F}_q[X]}{X^{n/m} - \zeta_k}$$

with ζ_k the $\phi(m)$ primitive m'th roots of unity in the group $\mathbb{F}_q^\times$.

Proof. The first equality holds by the reducibility of Φ_n over $\mathbb{F}_q$ with respect to m from Corollary 3. The isomorphism holds by the Chinese Remainder Theorem from Theorem 10 provided the $\phi(m)$ factors of Φ_n generate mutually coprime ideals. We now show this is indeed the case.

Each factor $X^{n/m} - \zeta_k$ is irreducible since we assume $\operatorname{ord}_n(q) = n/m$, and therefore each ring $\mathbb{F}_q/(X^{n/m} - \zeta_k)$ in the direct product is a (finite) field. From commutative ring theory we know an ideal I is maximal if and only if the quotient ring R/I is a field. Therefore, the ideals generated by the factors $X^{n/m} - \zeta_k$ are maximal. If I and J are distinct, maximal ideals, then $I + J$ is an ideal larger than both I and J and hence must be R, that is I and J are coprime. As maximal ideals $(X^{n/m} - \zeta_k)$ and $(X^{n/m} - \zeta_{k'})$ are distinct for all $k \neq k'$, we have it that the factors of Φ_n generate mutually coprime ideals. □

4 Ring of interest

This section examines a certain kind of ring in three sections of increasing specificity. The kind of ring of interest may seem rather arbitrary, but such rings are a central topic in algebraic number theory. For ring of integers $\mathcal{O}_L$ of number field L, our ring of interest is $\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L$ where $\mathfrak{p}$ is a prime ideal of the ring of integers $\mathcal{O}_K$ of some number field K contained in L. The nature of this ring is largely determined by how the ideal $\mathfrak{p} \subseteq \mathcal{O}_K$ splits into prime ideals in $\mathcal{O}_L$, and indeed that turns out to be the core topic of our investigation.

In the first section we present definitions and basic results for the general case of number fields K and L in which we assume nothing particular about the extension L/K. Here we will make use of the Chinese Remainder Theorem covered in Section 3. In the second section we focus on the special case that the extension L/K is Galois. The increased specificity yields certain simplifications, but those simplifications yield additional definitions and concepts to cover. In the third section we further specialize to the case that $K = \mathbb{Q}$ and $L = \mathbb{Q}[X]/\Phi_n(X)$, that is when L/K is an n'th cyclotomic extension. Here we will make use of the reducibility of cyclotomic polynomials covered in Section 3. In all sections we make claims standard to algebraic number theory without providing proof. For proofs one may consult books such as [Mil20] and [Mar18].

4.1 The general case

Let K and L be number fields with corresponding rings of integers $\mathcal{O}_K$ and $\mathcal{O}_L$, and consider the extension L/K. Rings of integers of number fields are 'Dedekind domains,' those are integral domains in which every (non-zero and proper) ideal factors into a (unique) product of prime ideals. Moreover, every prime ideal in a Dedekind domain is maximal.

Remark 2. *To say an ideal $\mathfrak{a}$ divides an ideal $\mathfrak{b}$ is to say there exists a quotient ideal $\mathfrak{q}$ such that $\mathfrak{a}\mathfrak{q} = \mathfrak{b}$. This is equivalent to the statement $\mathfrak{a} \supseteq \mathfrak{b}$. If it's counter-intuitive that the divisor is larger than the dividend, consider how ideals in $\mathbb{Z}$ are multiples of integers, so the smaller the integer the larger the ideal.*

Let $\mathfrak{p}$ be an ideal in $\mathcal{O}_K$, and note that $\mathfrak{p}\mathcal{O}_K$ is an ideal in $\mathcal{O}_L$. We say a prime ideal $\mathfrak{P}$ in $\mathcal{O}_L$ *lies over* $\mathfrak{p}$ (or $\mathfrak{p}$ *lies under* $\mathfrak{P}$) if $\mathfrak{P}|\mathfrak{p}\mathcal{O}_K$ (or we abuse notation and write $\mathfrak{P}|\mathfrak{p}$ and refer to prime ideals as 'primes'). Every prime $\mathfrak{p} \subseteq \mathcal{O}_K$ lies under some prime $\mathfrak{P} \subseteq \mathcal{O}_L$, and every prime $\mathfrak{P} \subseteq \mathcal{O}_L$ lies over some *unique* prime $\mathfrak{p} \subseteq \mathcal{O}_K$.

Continuing with $\mathfrak{P}$ and $\mathfrak{p}$, since they are prime and thus maximal ideals, the quotient rings $\mathcal{O}_L/\mathfrak{P}$ and $\mathcal{O}_K/\mathfrak{p}$ are fields (and in fact finite). This is because for any commutative ring R, an ideal I is maximal if and only if R/I is a field. Moreover, since $\mathfrak{P} \subseteq \mathfrak{p}\mathcal{O}_L$ the field $\mathcal{O}_L/\mathfrak{P}$ is a finite field extension of $\mathcal{O}_K/\mathfrak{p}$. The degree of the field extension $[\mathcal{O}_L/\mathfrak{P} : \mathcal{O}_K/\mathfrak{p}]$ is the *inertia degree* of $\mathfrak{P}$ over $\mathfrak{p}$, and denoted $f(\mathfrak{P}|\mathfrak{p})$.

Suppose the ideal $\mathfrak{p}\mathcal{O}_L \subseteq \mathcal{O}_L$ has unique prime factorization

$$\mathfrak{p}\mathcal{O}_L = \prod_{i=1}^{g} \mathfrak{P}_i^{e_i}$$

The exponents e_i are called *ramification indices*, and the *ramification index* e_i of $\mathfrak{P}_i$ over $\mathfrak{p}$ is denoted $e(\mathfrak{P}_i|\mathfrak{p}) = e_i$. Likewise we have the inertia degree f_i of $\mathfrak{P}_i$ over $\mathfrak{p}$ as $f(\mathfrak{P}_i|\mathfrak{p}) = [\mathcal{O}_L/\mathfrak{P}_i :$

$\mathcal{O}_K/\mathfrak{p}]$. The ramification indices and inertia degrees of the factorization satisfy the fundamental relation

$$\sum_{i=1}^{g} e_i f_i = \sum_{i=1}^{g} e(\mathfrak{P}_i|\mathfrak{p}) f(\mathfrak{P}_i|\mathfrak{p}) = \sum_{\mathfrak{P}|\mathfrak{p}} e(\mathfrak{P}|\mathfrak{p}) f(\mathfrak{P}|\mathfrak{p}) = [L : K]$$

Several definitions describe the nature of the factorization:

- If $e_i > 1$ we say $\mathfrak{p}$ *ramifies* in L, and more specifically $\mathfrak{P}_i$ is *ramified* over $\mathfrak{p}$. Otherwise, with all $e_i = 1$ we say $\mathfrak{p}$ is *unramified* in L.
- If $g = [L : K]$ (in which case $e_i, f_i = 1$) we say $\mathfrak{p}$ *splits completely* (or is *totally split*) in L.
- If $g = 1$, then in the case $e_1 = 1$ we say $\mathfrak{p}$ is *inert* in L, while in the case $f_1 = 1$ we say $\mathfrak{p}$ is *totally ramified* in L.

Having introduced the setting, in the two subsections that follow we will introduce two basic results. The first result is how one may go about finding the prime factors of $\mathfrak{p}\mathcal{O}_L$making a weak assumption on $\mathfrak{p}$. The second result is how one may represent our ring of interest $\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L$ having found the prime factors of $\mathfrak{p}\mathcal{O}_L$.

4.1.1 Finding the factors

We present a common technique for finding the prime factorization of prime ideal $\mathfrak{p} \in \mathcal{O}_K$ in terms of prime ideals of $\mathcal{O}_L$. First we must fix some $\alpha \in \mathcal{O}_L$ such that $L = K[\alpha] \cong K[X]/(h(x))$where $h \in K[X]$ is the minimal polynomial of α over K. Then $\mathcal{O}_K[\alpha]$ is a subring of $\mathcal{O}_L$ with finite index $[\mathcal{O}_L : \mathcal{O}_K[\alpha]]$. This technique for factoring $\mathfrak{p}$ in $\mathcal{O}_L$ works provided the prime integer that lies under $\mathfrak{p}$ does not divide $[\mathcal{O}_L : \mathcal{O}_K[\alpha]]$.

Now h, while defined in $K[X]$, is actually in $\mathcal{O}_K[X]$ because $\alpha \in \mathcal{O}_L$ is the root of some polynomial in $\mathbb{Z}[X]$ divisible by $h \in K[X]$ which prevents h from having coefficients in $K \setminus \mathcal{O}_K$. Suppose $\overline{h} = h(\mathrm{mod}\mathfrak{p})$ factors over $\mathcal{O}_K/\mathfrak{p}$ as

$$\overline{h}(X) = \prod_{i=1}^{g} \overline{h_i}(X)^{e_i} \mod \mathfrak{p}$$

where $\overline{h_i}$ are distinct, monic, and irreducible in $(\mathcal{O}_K/\mathfrak{p})[X]$. Then the prime decomposition of $\mathfrak{p}$ in $\mathcal{O}_L$ is

$$\mathfrak{p}\mathcal{O}_L = \prod_{i=1}^{g} \mathfrak{P}_i^{e_i}$$

where $\mathfrak{P}_i = (\mathfrak{p}, h_i(\alpha)) = \mathfrak{p}\mathcal{O}_L + h_i(\alpha)\mathcal{O}_L$ is the ideal in $\mathcal{O}_L$ generated by $\mathfrak{p}$ and $h_i(\alpha)$. Moreover, we have the following field isomorphism

$$\frac{\mathcal{O}_L}{(\mathfrak{p}, h_i(\alpha))} \cong \frac{(\mathcal{O}_K/\mathfrak{p})[X]}{(\overline{h_i}(X))}$$

The inertia degree $f(\mathfrak{P}_i|\mathfrak{p}) = [\mathcal{O}_L/\mathfrak{P}_i : \mathcal{O}_K/\mathfrak{p}]$ is then seen to be $\deg(\overline{h_i}) = \deg(h_i)$.

4.1.2 Using the Chinese Remainder Theorem

We will now exploit prime factorization to invoke the Theorem 10. Since each $\mathfrak{P}_i$ in the factorization is maximal, so are the powers $\mathfrak{P}_i^{e_i}$. That is, if ideals I and J are coprime ($I + J = R$) then so are their powers ($I^k + J^{k'} = R$). Suppose otherwise, that there exists some maximal ideal K such that $I^k + J^{k'} = K$. Then since I (or argue using J) is prime, $I^k \subseteq K$ implies $I \subseteq K$ which contradicts the maximality of I. The criteria for the Chinese Remainder Theorem requires that the ideals $\mathfrak{P}_i^{e_i}$ be coprime. For this we use the fact that any two distinct, maximal ideals are coprime. For justification, note that if I and J are distinct, maximal ideals, then $I + J$ is an ideal larger than both I and J and hence must be the full ring. Now we may invoke the Chinese Remainder Theorem for the ring $\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L$.

$$\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L = \mathcal{O}_L \Big/ \prod_{i=1}^{g} \mathfrak{P}_i^{e_i} \cong \prod_{i=1}^{g} \mathcal{O}_L/\mathfrak{P}_i^{e_i}$$

Consider what this looks like having used the Section 4.1.1 method of factorizing $\mathfrak{p}\mathcal{O}_L$:

$$\frac{\mathcal{O}_L}{\mathfrak{p}\mathcal{O}_L} \cong \prod_{i=1}^{g} \frac{\mathcal{O}_L}{(\mathfrak{p}, h_i(\alpha))^{e_i}} \cong \prod_{i=1}^{g} \frac{(\mathcal{O}_K/\mathfrak{p})[X]}{(\overline{h_i}(X))^{e_i}}$$

4.2 The Galois case

Continuing with L/K and the factorization of $\mathfrak{p}\mathcal{O}_L$ into primes $\mathfrak{P}_i$, the ramification indices and inertia degrees are simple in the case that L/K is Galois. In particular, all $e_i = e$ are equal, and all $f_i = f$ are equal, and as such, the fundamental relation becomes

$$efg = [L : K]$$

Since automorphisms on L induce automorphisms on subrings, we have $\sigma(\mathcal{O}_L) = \mathcal{O}_L$ for any $\sigma \in \text{Gal}(L/K)$. Also noting that $\sigma(\mathfrak{p}) = \mathfrak{p}$ because σ fixes K, we may apply σ to the prime decomposition of $\mathfrak{p}\mathcal{O}_L$ to get

$$\mathfrak{p}\mathcal{O}_L = \sigma(\mathfrak{p}\mathcal{O}_L) = \prod_{i=1}^{g} \sigma(\mathfrak{P}_i)^e$$

As an automorphism, σ maps prime ideals to prime ideal, so $\{\sigma(\mathfrak{P}_i)\}_{i\in[g]}$ must be the unique prime factorization of $\mathfrak{p}\mathcal{O}_L$ and thus coincide with $\{\mathfrak{P}_i\}_{i\in[g]}$. In other words, σ acts on the prime ideals by permuting them. Moreover, the action is *transitive*, meaning for every two prime ideals $\mathfrak{P}_i$ and $\mathfrak{P}_{i'}$ lying over $\mathfrak{p}$ there exists some $\sigma \in \text{Gal}(L/K)$ such that $\sigma(\mathfrak{P}_i) = \mathfrak{P}_{i'}$. Furthermore, to say the action is *simply transitive* is to say there exists a unique such σ.

For some prime ideal $\mathfrak{P} \subseteq \mathcal{O}_L$ the *decomposition group* of $\mathfrak{P}$ is the subgroup of Gal(L/K) that fixes $\mathfrak{P}$, that is

$$D(\mathfrak{P}) = \{\sigma \in \text{Gal}(L/K) \mid \sigma(\mathfrak{P}) = \mathfrak{P}\}$$

As with every Galois subgroup there is an associated field, the field that is fixed by exactly the automorphisms of the subgroup. The *decomposition field* associated with $D(\mathfrak{P})$ is

$$L_D(\mathfrak{P}) = \{x \in L \mid \forall \sigma \in D(\mathfrak{P}),\ \sigma(x) = x\}$$

In the case $\sigma \in D(\mathfrak{P})$, since σ is invariant on $\mathfrak{P}$ and thus its cosets, we may consider $\overline{\sigma} = \sigma(\bmod \mathfrak{P})$ as an automorphism on $\mathcal{O}_L/\mathfrak{P}$. Note by fixing K, any $\sigma \in \text{Gal}(L/K)$ also fixes $\mathcal{O}_K/\mathfrak{p}$. Therefore, with any $\sigma \in D(\mathfrak{P})$ inducing an automorphism on $\mathcal{O}_L/\mathfrak{P}$ that fixes subfield $\mathcal{O}_K/\mathfrak{p}$, we have the natural map $D(\mathfrak{P}) \to \text{Gal}((\mathcal{O}_L/\mathfrak{P})/(\mathcal{O}_K/\mathfrak{p}))$ defined as $\sigma \to \overline{\sigma}$. The kernel of this homomorphism, a subgroup of $D(\mathfrak{P})$, is referred to as the *inertia* group of $\mathfrak{P}$, and takes the form

$$E(\mathfrak{B}) = \{\sigma \in D(\mathfrak{P}) \mid \forall x \in \mathcal{O}_L,\ \sigma(x) \equiv x \mod \mathfrak{P}\}$$

Analogous to the decomposition field, we define the *inertia field* associated with $E(\mathfrak{P})$ as

$$L_E(\mathfrak{P}) = \{x \in L \mid \forall \sigma \in E(\mathfrak{P}),\ \sigma(x) = x\}$$

The natural map from $D(\mathfrak{P})$ to $\text{Gal}((\mathcal{O}_L/\mathfrak{P})/(\mathcal{O}_K/\mathfrak{p}))$ is actually surjective, and thus we have a group isomorphism $D(\mathfrak{P})/E(\mathfrak{P}) \cong \text{Gal}((\mathcal{O}_L/\mathfrak{P})/(\mathcal{O}_K/\mathfrak{p}))$. Since the latter Galois group is cyclic and generated by the Frobenius element, so is $D(\mathfrak{P})/E(\mathfrak{P})$ and we denote it as $F(\mathfrak{P})$. We have the following automorphism group orders:

- $|\text{Gal}(L/K)| = e(\mathfrak{P}|\mathfrak{p})f(\mathfrak{P}|\mathfrak{p})g$
- $|D(\mathfrak{P})| = e(\mathfrak{P}|\mathfrak{p})f(\mathfrak{P}|\mathfrak{p})$
- $|E(\mathfrak{P})| = e(\mathfrak{P}|\mathfrak{p})$
- $|F(\mathfrak{P})| = f(\mathfrak{P}|\mathfrak{p})$

The following diagram illustrates the changes in ramification indices and inertia degrees as we step through the fields from K to L.

$$\begin{tikzcd}
 & L \arrow[dd, equal] & \mathfrak{P} \arrow[dd, equal] & & \\
{[L : L_E] = e} & & & e(\mathfrak{P}|\mathfrak{P} \cap L_E) = e(\mathfrak{P}|\mathfrak{p}) & f(\mathfrak{P}|\mathfrak{P} \cap L_E) = 1 \\
 & L_E \arrow[dd, equal] & \mathfrak{P} \cap L_E \arrow[dd, equal] & & \\
{[L_E : L_D] = f} & & & e(\mathfrak{P} \cap L_E|\mathfrak{P} \cap L_D) = 1 & f(\mathfrak{P} \cap L_E|\mathfrak{P} \cap L_D) = f(\mathfrak{P}|\mathfrak{p}) \\
 & L_D \arrow[dd, equal] & \mathfrak{P} \cap L_D \arrow[dd, equal] & & \\
{[L_D : K] = g} & & & e(\mathfrak{P} \cap L_D|\mathfrak{p}) = 1 & f(\mathfrak{P} \cap L_D|\mathfrak{p}) = 1 \\
 & K & \mathfrak{p} & &
\end{tikzcd}$$

The word ‘decomposition’ reflects how in the bottom layer the prime decomposes into g parts. The word ‘inertia’ reflects how in the middle layer the prime remains inert with inertia degree 1, not ramifying until the top layer.

4.2.1 The abelian Galois case

In the case that the Galois group is abelian, the decomposition groups for all $\mathfrak{P}_i$ lying over $\mathfrak{p}$ coincide. Let $\mathfrak{P}$ be any such $\mathfrak{P}_i$. It suffices to show that $D(\mathfrak{P}) = D(\tau(\mathfrak{P}))$ for all τ in the Galois group. This is because by the transitivity of the Galois action every prime lying above $\mathfrak{p}$ takes the form $\tau(\mathfrak{P})$ for some τ. Equality of $D(\mathfrak{P})$ and $D(\tau(\mathfrak{P}))$ is seen set-membership-wise as

$$\begin{aligned}
&\sigma \in D(\tau(\mathfrak{P})) \\
&\iff \sigma(\tau(\mathfrak{P})) = \tau(\mathfrak{P}) \\
&\iff (\tau^{-1} \circ \sigma \circ \tau)(\mathfrak{P}) = \mathfrak{P} \\
&\iff \sigma(\mathfrak{P}) = \mathfrak{P} \\
&\iff \sigma \in D(\mathfrak{P})
\end{aligned}$$

Since all decomposition groups $D(\mathfrak{P}_i)$ are the same (normal) subgroup of Gal(L/K), we may consider the quotient group Gal$(L/K)/D(\mathfrak{P})$ of order $efg/ef = g$ where $\mathfrak{P}$ is again any $\mathfrak{P}_i$. We will now show that this quotient group acts simply transitively on the primes lying above $\mathfrak{p}$. To do so we show that the map $\sigma \to \sigma(\mathfrak{P})$ with domain Gal$(L/K)/D(\mathfrak{P})$ is injective, hence for each prime $\mathfrak{P}'$ lying above $\mathfrak{p}$ there is a unique $\sigma \in$ Gal$(L/K)/D(\mathfrak{P})$ such that $\sigma(\mathfrak{P}) = \mathfrak{P}'$. For two cosets $\tau \circ D(\mathfrak{P})$ and $\tau' \circ D(\mathfrak{P})$ had by some coset representatives τ and τ' we have

$$\begin{aligned}
&(\tau' \circ D)(\mathfrak{P}) = (\tau \circ D)(\mathfrak{P}) \\
&\implies \tau'(D(\mathfrak{P})) = \tau(D(\mathfrak{P})) \\
&\implies \tau'(\mathfrak{P}) = \tau(\mathfrak{P}) \\
&\implies (\tau^{-1} \circ \tau')(\mathfrak{P}) = \mathfrak{P} \\
&\implies \tau^{-1} \circ \tau' \in D(\mathfrak{P})
\end{aligned}$$

and hence by the equivalence relation of cosets τ and τ' belong to the same coset.

The simple transitivity of Gal$(L/K)/D(\mathfrak{P})$ allows us to express the prime factorization of $\mathfrak{p}\mathcal{O}_L$ as

$$\mathfrak{p}\mathcal{O}_L = \prod_{\sigma \in \text{Gal}(L/K)/D(\mathfrak{P})} \sigma(\mathfrak{P})^e$$

and along with the Chinese Remainder Theorem we may write our ring of interest as

$$\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L \cong \prod_{\sigma \in \text{Gal}(L/K)/D(\mathfrak{P})} \mathcal{O}_L/\sigma(\mathfrak{P})^e$$

4.3 The cyclotomic case

We will now examine the special case of cyclotomic fields, that is number fields of the form $L = \mathbb{Q}(\mu_n)$ (extending $K = \mathbb{Q}$) where μ_n are the n'th complex roots of unity. The extension $L/K = \mathbb{Q}(\mu_n)/\mathbb{Q}$ is Galois, and we briefly examined such extensions in Section 2. The minimal polynomial over $\mathbb{Q}$ of the generators of μ_n (the primitive n'th complex roots of unity) is Φ_n, the n'th cyclotomic polynomial. The ring of integers of $\mathbb{Q}$ is $\mathbb{Z}$, and the ring of integers of $\mathbb{Q}(\mu_n)$ turns out to be $\mathbb{Z}[\mu_n]$.

Remark 3. *For a ring R the notations $R[\alpha]$ and $R(\alpha)$ indeed carry different meanings, though they coincide in certain cases. The ring $R[\alpha]$ denotes the ring had by adjoining α to R, which is the smallest* ring *containing both α and R. The ring $R(\alpha)$, only really used when R is a field,* *denotes the ring had by adjoining α* and $1/\alpha$ *to R, that is the smallest* field *containing both α and R.*

If α is the root of some polynomial h over R with invertible constant coefficient, then $1/\alpha$ exists in $R[\alpha]$ and thus $R[\alpha] = R(\alpha)$ since

$$\alpha \cdot \frac{h_{\deg(h)}\alpha^{\deg(h)-1} + \cdots + h_1}{-h_0} = 1$$

Therefore in the case that R is a field, the two notations coincide if and only if α is algebraic over R.

In our case, since $\mathbb{Q}$ is a field and μ_n vanishes on $\Phi_n \in \mathbb{Q}[X]$ we have $\mathbb{Q}[\mu_n] = \mathbb{Q}(\mu_n)$, while on the other hand $\mathbb{Z}[\mu_n] \neq \mathbb{Z}(\mu_n)$.

Recall our goal here is to investigate the ring $\mathcal{O}_L/\mathfrak{p}\mathcal{O}_L$ where $\mathfrak{p}$ is a prime ideal of $\mathcal{O}_K$. In the cyclotomic case, since $\mathcal{O}_K = \mathbb{Z}$ we have $\mathfrak{p} = (p) \subseteq \mathbb{Z}$ generated by some prime number $p \in \mathbb{Z}$. Our ring of interest is then $\mathbb{Z}[\mu_n]/(p)\mathbb{Z}[\mu_n]$. To clarify, we have the following correspondences:

- **Number fields:** $K = \mathbb{Q}$ and $L = \mathbb{Q}(\mu_n)$
- **Rings of integers:** $\mathcal{O}_K = \mathbb{Z}$ and $\mathcal{O}_L = \mathbb{Z}[\mu_n]$.
- **Primes:** $\mathfrak{p} = (p) \subseteq \mathbb{Z}$ for some prime number $p \in \mathbb{Z}$, and the primes $\mathfrak{P}_i \subseteq \mathbb{Z}[\mu_n]$ lying above $\mathfrak{p}$ will be found in the following subsection.

4.3.1 Cyclotomics as a general case

Now we apply the tools from the general case discussed in Section 4.1 to the cyclotomic case, namely how to find the prime factors of $(p)\mathbb{Z}[\mu_n]$ and how to represent $\mathbb{Z}[\mu_n]/(p)\mathbb{Z}[\mu_n]$ in terms of those prime factors using the Chinese Remainder Theorem.

To find the factorization of prime ideal $\mathfrak{p} = (p) \in \mathcal{O}_K$ into prime ideals of $\mathcal{O}_L = \mathbb{Z}[\mu_n]$, we will use the method presented Section 4.1.1. First we fix some primitive n'th root of unity $\alpha \in \mu_n$. Then we have $L = \mathbb{Q}(\alpha) = \mathbb{Q}[\alpha] = K[\alpha]$ since L is an algebraic extension of K. Furthermore, we have the isomorphism $\mathbb{Q}(\alpha) \cong \mathbb{Q}[X]/(\Phi_n(X))$. To complete the criteria for using the method, note that p (the prime number lying under (p)) does not divide $[\mathcal{O}_L : \mathcal{O}_K[\alpha]] = [\mathbb{Z}[\alpha] : \mathbb{Z}[\alpha]] = 1$.

We know from the discussion of reducibility of cyclotomic polynomials over finite fields from Corollary 3 that Φ_n over the finite field $\mathcal{O}_K/\mathfrak{p} = \mathbb{Z}/(p)$ factors into irreducibles as

$$\Phi_n(X) \mod (p) = \prod_{i=1}^{\phi(r)} (X^{n/r} - \zeta_i) \mod (p)$$

where ζ_i are the primitive r'th roots of unity in the group $(\mathbb{Z}/(p))^\times$ for $r = n/\mathrm{ord}_n(p)$. The method then yields the prime ideal decomposition of (p) in $\mathbb{Z}[\alpha]$ as

$$(p)\mathbb{Z}[\alpha] = \prod_{i=1}^{\phi(r)} \left(p, \alpha^{n/r} - \zeta_i\right)$$

That is, $(p)\mathbb{Z}[\alpha]$ splits completely into prime ideals generated by both p and $\alpha^{n/r} - \zeta_i$ which we denote $\mathfrak{P}_i = (p, \alpha^{n/r} - \zeta_i)$. Moreover, the finite field $\mathbb{Z}[\alpha]/(p, \alpha^{n/r} - \zeta_i)$ is isomorphic to

$(\mathbb{Z}/(p))[X]/(X^{n/r}-\zeta_i)$ and hence $f((p, X^{n/r}-\zeta_i)|(p)) = \deg(X^{n/r}-\zeta_i) = \mathrm{ord}_n(p)$. Notice how these inertia degrees are $\mathfrak{P}$-invariant, and also notice how the ramification indices are also $\mathfrak{P}$-invariant as $(p)\mathbb{Z}[\alpha]$ splits completely. Recall that these invariants are expected since we are in a Galois extension.

We may use this prime factorization to apply the Chinese Remainder Theorem and express our ring of interest as

$$\frac{\mathbb{Z}[\mu_n]}{(p)\mathbb{Z}[\mu_n]} \cong \prod_{i=1}^{\phi(r)} \frac{\mathbb{Z}[\alpha]}{(p, \alpha^{n/r}-\zeta_i)} \cong \prod_{i=1}^{\phi(r)} \frac{(\mathbb{Z}/(p))[X]}{(X^{n/r}-\zeta_i)}$$

This is a special case of the more general result for representing $\mathbb{F}_q[X]/\Phi_n(X)$ from Lemma 4.

4.3.2 Cyclotomics as a Galois case

Lastly we apply the tools from the Galois case in Section 4.2 to the cyclotomic case. Keep in mind throughout that $\mathfrak{P}_i = (p, X^{n/r}-\zeta_i)$. As expected, we observed in the previous subsection that all inertia degrees and ramification indices coincide. The fundamental identity in the cyclotomic case becomes

$$efg = 1 \cdot \mathrm{ord}_n(p) \cdot \phi(n/\mathrm{ord}_n(p)) = \phi(n) = [\mathbb{Q}(\mu_n) : \mathbb{Q}]$$

Moreover, our list of automorphism group orders becomes

- $|\mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})| = efg = \phi(n)$
- $|D(\mathfrak{P}_i)| = ef = 1 \cdot \mathrm{ord}_n(p) = \mathrm{ord}_n(p)$
- $|E(\mathfrak{P}_i)| = e = 1$
- $|F(\mathfrak{P}_i)| = f = \mathrm{ord}_n(p)$

Recall that the natural map $\sigma \to \sigma(\bmod\ (p))$ from $F(\mathfrak{P}_i) := D(\mathfrak{P}_i)/E(\mathfrak{P}_i)$ to $\mathrm{Gal}((\mathbb{Z}[\mu_n]/\mathfrak{P}_i)/(\mathbb{Z}/(p)))$ is surjective. Since $E(\mathfrak{P}_i)$ is the trivial group with order 1, the natural map is in fact an isomorphism between $D(\mathfrak{P}_i)$ and $\mathrm{Gal}((\mathbb{Z}[\mu_n]/\mathfrak{P}_i)/(\mathbb{Z}/(p)))$. As listed above, $D(\mathfrak{P}_i)$ has order $\mathrm{ord}_n(p)$, and as a subgroup of $\mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})$ it is isomorphic to a subgroup of $(\mathbb{Z}/(n))^\times$. With these constraints in mind, we may guess that $D(\mathfrak{P}_i)$ is isomorphic to the subgroup of $(\mathbb{Z}/(n))^\times$ generated by p, which we denote $\langle p \rangle$. Indeed, from the theory of finite fields we know the Frobenius automorphism $x \to x^p$ generates the Galois group of any finite field extension.

Since the Galois group is abelian (as it's isomorphic to $(\mathbb{Z}/(n))^\times$), we may use the abelian Galois case discussed in Section 4.2.1 to express the prime factorization in terms of a single prime as

$$(p)\mathbb{Z}[\mu_n] = \prod_{\sigma \in \mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/D(\mathfrak{P})} \sigma((p, X^{n/r}-\zeta))$$

where ζ is any primitive $(n/\mathrm{ord}_n(p))$'th root of unity in the group $(\mathbb{Z}/(p))^\times$, and $\mathfrak{P}$ is any of the primes $\mathfrak{P}_i$. Once again using the Chinese Remainder Theorem we may write our ring of interest as

$$\frac{\mathbb{Z}[\mu_n]}{(p)\mathbb{Z}[\mu_n]} \cong \prod_{\sigma \in \mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/D(\mathfrak{P})} \frac{\mathbb{Z}[\alpha]}{\sigma((p, \alpha^{n/r}-\zeta))} \cong \prod_{\sigma \in \mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/D(\mathfrak{P})} \frac{(\mathbb{Z}[\mu_n]/(p))[X]}{\sigma((X^{n/r}-\zeta))}$$

where for the last isomorphism we use the fact that $\sigma((p, \alpha^{n/r} - \zeta)) = (p, \sigma(\alpha^{n/r} - \zeta))$ since σ fixes $p \in \mathbb{Q}$.

5 Tree form factorization

In Section 4 we saw how our ring of interest may be expressed in two particular ways. In the general case of cyclotomics discussed in Section 4.3.1, we observed the representation

$$\prod_{i=1}^{\phi(r)} \frac{(\mathbb{Z}/(p))[X]}{(X^{n/r} - \zeta_i)} \tag{1}$$

where $r = n/\mathrm{ord}_n(p)$, $\mathrm{ord}_r(p) = 1$, and ζ_i are the $\phi(r)$ primitive r'th roots of unity. In the galois case of cyclotomic as discussed in Section 4.3.2, we observed the representation

$$\prod_{\sigma \in \mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/D(\mathfrak{P})} \frac{(\mathbb{Z}[\mu_n]/(p))[X]}{\sigma((X^{n/r} - \zeta))} \tag{2}$$

for the same values of r, where $\mathbb{Q}(\mu_n)/\mathbb{Q}$ is the n'th cyclotomic field, ζ is any primitive r'th root of unity, and $D(\mathfrak{P})$ is the decomposition group of the prime ideal $\mathfrak{P} = (X^{n/r} - \zeta)$. We only focused, however, on how the ideal (p) splits into *prime* ideals of the cyclotomic ring of integers $\mathbb{Z}[\mu_n]$. We are now interested rather in how (p) splits more generally into *reducible* (non-prime) ideals before splitting into irreducible (prime) ideals. In other words, the two decompositions Equation (1) and Equation (2) are fully-split decompositions, whereas now we're interested in more general partially-split decompositions. Our motivation is the full convenience of such representations.

If one recalls from Section 4.1.1, to determine the splitting of (p) in $\mathbb{Z}[\mu_n]$ we instead turn to the splitting of Φ_n over $\mathbb{Z}/(p)$. The method may be applied, however, not only to identify how the full splitting into prime ideals occurs, but to identify any partial splitting into non-prime ideals as well. Specifically, if Φ_n splits over $\mathbb{Z}/(p)$ into not necessarily irreducible factors h_i as

$$\Phi_n(X) \bmod (p) = \prod_i h_i(X) \bmod (p)$$

then (p) splits in $\mathbb{Z}[\mu_n]$ such that our ring representation becomes

$$\frac{(\mathbb{Z}/(p))[X]}{(\Phi_n(X))} = \prod_i \frac{(\mathbb{Z}/(p))[X]}{(h_i(X))}$$

While this kind of decomposition is similar to Equation (1), we can also derive decompositions similar to Equation (2) by analyzing how (subgroups of) the Galois group of $\mathbb{Q}(\mu_n)/\mathbb{Q}$ acts on these ideals $(h_i(X)) \subseteq (\mathbb{Z}/(p))[X]/(\Phi_n(X))$. In fact, instead of focusing on the ring representation directly, we will only focus on the ideals $(h_i(X))$, how the Galois group acts on them, and how they compose to form $(\Phi_n(X))$ as

$$(\Phi_n(X)) = \prod_i (h_i(X))$$

In particular, the following two sections focus on different ways to index the ideals $(h_i(X))$. In the first section we use indexing similar to Equation (1), and in the second section we use indexing similar to Equation (2).

5.1 Indexing via H_m sets

Let $n = \prod_k p_k^{f_k}$ and $m = \prod_k p_k^{e_k}$ for primes p_k and powers $e_k \geq f_k \geq 1$. We begin with a lemma that holds for such n and m.

Lemma 5 ($\phi(n)/\phi(m) = n/m$)**.** *For n and m as described we have $\phi(n)/\phi(m) = n/m$.*

Proof. By the properites of Euler's totient function ϕ we have

$$\phi(n) = \prod_k \phi(p_k^{e_k}) = \prod_k p_k^{e_k}\left(1 - \frac{1}{p_k}\right)$$
$$\phi(n) = \prod_k \phi(p_k^{f_k}) = \prod_k p_k^{f_k}\left(1 - \frac{1}{p_k}\right)$$

Therefore $\phi(n)/\phi(m)$ reduces to

$$\prod_k \frac{p_k^{e_k}}{p_k^{f_k}} = \frac{n}{m}$$

□

Next define $H_m \subseteq \mathbb{Z}_n^\times$ to be the subgroup

$$H_m \coloneqq \left\{x \in \mathbb{Z}_n^\times \mid x \equiv 1 \bmod m\right\}$$

Note H_n is the trivial subgroup. A important identity for H_m is $\mathbb{Z}_n^\times / H_m \cong \mathbb{Z}_m^\times$, which comes from the First Isomorphism Theorem with respect to the homomorphism $(x \to x \bmod m)\colon \mathbb{Z}_n^\times \to \mathbb{Z}_m^\times$ with kernel H_m and image $\mathbb{Z}_m^\times$. From this isomorphism and Lemma 5 we conclude $|H_m| = \phi(n)/\phi(m) = n/m$.

Our first form for indexing factors of $(\Phi_n(X))$ is the form we've used in previous sections, that is by enumerating the primitive m'th roots of unity ζ_i as

$$(\Phi_n(X)) = \prod_{i=1}^{\phi(m)} (X^{n/m} - \zeta_i)$$

assuming $\mathrm{ord}_m(p) = 1$, and indeed *let us assume henceforth that this holds by assumption whenever we split* Φ_n. We may instead index the factors in the direct product by selecting any primitive m'th root of unity ζ and exponentiating over $\mathbb{Z}_m^\times$ as

$$(\Phi_n(X)) = \prod_{i \in \mathbb{Z}_m^\times} (X^{n/m} - \zeta^i)$$

Alternatively, consider the isomorphism $\mathbb{Z}_m^\times \cong \mathbb{Z}_n^\times / H_m$ together with the map $(x \to x \cdot H_m)\colon \mathbb{Z}_m^\times \to \mathbb{Z}_n^\times / H_m$. Using $\mathbb{Z}_n^\times / H_m$ as an alternative set to $\mathbb{Z}_m^\times$ for exponentiating ζ is well defined because H_m is the subgroup of $\mathbb{Z}_n^\times$ that fixes the (primitive) m'th roots of unity on exponentiation. By the fact that H_m, and hence its cosets, are fixed on the (primitive) m'th root of unity ζ, exponentiating ζ by any coset representatives in a given coset yield the same result. We may thus write

$$(\Phi_n(X)) = \prod_{i \in \mathbb{Z}_n^\times / H_m} (X^{n/m} - \zeta^i) \tag{3}$$

We have so far represented $(\Phi_n(X))$ in a decomposition using a new indexing set $\mathbb{Z}_n^\times / H_m$. We can extend our use of H_m to further decompose the factors $(X^{n/m} - \zeta^i)$, and continuing as such we can recursively decompose $(\Phi_n(X))$ down to it's prime ideals.

Theorem 11 (Decomposition by H_m sets)**.** *Consider a sequence $m_1, \dots, m_t$ such that all m_k share the same prime factors as n, and $m_k | m_{k+1}$ for $k \in [t-1]$ and $m_t | n$. With ζ a primitive m_t'th root of unity, we have*

$$(\Phi_n(X)) = \prod_{i_1 \in \mathbb{Z}_n^\times / H_{m_1}} \prod_{i_2 \in H_{m_1} / H_{m_2}} \cdots \prod_{i_t \in H_{m_{t-1}} / H_{m_t}} (X^{n/m_t} - \zeta^{i_1 \cdots i_t})$$

Proof. We will prove a single iteration of this recursion, and to do so we simplify notation. Given a pair m_k and m_{k+1}, we alias with $\ell := m_i$ and $m := m_{i+1}$ noting $\ell | m$. Furthermore, let ζ be a primitive m'th root of unit, and let $\eta := \zeta^{m/\ell}$. We'll prove that η is a primitive ℓ'th root of unity and for $i \in \mathbb{Z}_n^\times / H_\ell$ we have

$$(X^{n/\ell} - \eta^i) = \prod_{j \in H_\ell / H_m} (X^{n/m} - \zeta^{ij}) \tag{4}$$

This decomposition may be applied directly after decomposition Equation (3). To then recurse on this decomposition, note that with representative $i \in \mathbb{Z}_n^\times / H_\ell$ and representative $j \in H_\ell / H_m$ we have representative $ij \in \mathbb{Z}_n^\times / H_m$.

To prove that η is a primitive ℓ'th roof of unity, we argue $\eta^x = 1$ if and only if ℓ divides x. The equation

$$\eta^x = \left(\zeta^{m/\ell}\right)^x = 1$$

holds if and only if m divides xm/ℓ, since ζ is a primitive m'th root of unity. But m divides $m(x/\ell)$ if and only if ℓ divides x.

To prove the decomposition Equation (4), we show the generating polynomials of both sides are the same in degree and any root of the right side is a root of the left side. Regarding degree, the left side has degree n/ℓ while the right side has degree

$$(n/m) \frac{|H_\ell|}{|H_m|} = (n/m) \frac{n/\ell}{n/m} = n/\ell$$

Regarding roots, any X satisfying $X^{n/m} = \zeta^{ij}$ for some $j \in H_\ell / H_m$ satisfies $X^{n/\ell} = \eta^i$. Take $X^{n/m} = \zeta^{ij}$ and raise both sides to m/ℓ getting

$$X^{n/\ell} = \left(\zeta^{m/\ell}\right)^{ij} = \eta^{ij} = \eta^i$$

where $\zeta^{m/\ell} = \eta$ by definition of η, and $\eta^{ij} = \eta^i$ since $j \subseteq H_m$ fixes η as a (primitive) ℓ'th root of unity. □

5.2 Indexing via G_m sets

Recall that the Galois group of the n'th cyclotomic field $\mathbb{Q}(\mu_n)/\mathbb{Q}$ is isomorphic to $\mathbb{Z}_n^\times$ via the map

$$(k \to \sigma_k) \colon \mathbb{Z}_n^\times \to \text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})$$

where σ_k leaves $\mathbb{Q}$ fixed while raising any element of μ_n to the power k. In the context of $(\Phi_n(X))$ over $\mathbb{Z}/(p)$, that means leaving the base field $\mathbb{Z}/(p)$ fixed and raising X to the power k.

We would like to exploit this isomorphism to express the decomposition by H_m sets in Theorem 11 using subgroups of the Galois group. Define $G_m \subseteq \text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})$ to be the Galois subgroup isomorphic to $H_m \subseteq \mathbb{Z}_n^\times$.

Theorem 12 (Decomposition by G_m sets)**.** *Consider a sequence $m_1, \dots, m_t$ such that all m_k share the same prime factors as n, and $m_k | m_{k+1}$ for $k \in [t-1]$ and $m_t | n$. With ζ a primitive m_t'th root of unity, we have*

$$(\Phi_n(X)) = \prod_{\sigma_1 \in Gal(\mathbb{Q}(\mu_n)/\mathbb{Q})/G_{m_1}} \prod_{\sigma_2 \in G_{m_1}/G_{m_2}} \cdots \prod_{\sigma_t \in G_{m_{t-1}}/G_{m_t}} (\sigma_t \circ \cdots \circ \sigma_1)(X^{n/m_t} - \zeta)$$

Proof. As we did in proving decomposition by H_m sets in Theorem 11, we will prove this equation by a single iteration of recursion with simplified notation. Given a pair m_k and m_{k+1}, we alias with $\ell := m_k$ and $m := m_{k+1}$ noting $\ell | m$. Furthermore, let ζ be a primitive m'th root of unit, and let $\eta := \zeta^{m/\ell}$. We'll prove that for $\sigma_i \in \text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/G_\ell$ we have

$$\sigma_i(X^{n/\ell} - \eta) = \prod_{\sigma_j \in G_\ell / G_m} (\sigma_j \circ \sigma_i)(X^{n/m} - \zeta) \tag{5}$$

To recurse on this decomposition, note that with $\sigma_i \in \text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/G_\ell$ and $\sigma_j \in G_\ell/G_m$ we have $(\sigma_j \circ \sigma_i) \in \text{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q})/G_m$. We also note, having proved it previously, that η is a primitive ℓ'th root of untiy.

We now argue for the following equation

$$\sigma_k((X - \alpha)) = (X^k - \alpha) = (X - \alpha^{1/k})$$

where $1/k$ is the inverse of k in the group $\mathbb{Z}_n^\times$. The first equation follows by definition of σ_k. The second equation will be proven by showing each ideal contains the other. As any root of $X - \alpha^{1/k}$ is a root of $X^k - \alpha$, we have $(X^k - \alpha)$ contained in $(X - \alpha^{1/k})$. For the other direction, let $k' = 1/k$ denote the inverse of k in the group $\mathbb{Z}_n^\times$. Since we are considering these ideals within the ring $(\mathbb{Z}[X]/(p))/(\Phi_n(X))$ where $X^n = 1$, we have $X^{k'k} = X$. Then any root of $X^k - \alpha$ is a root of

$$(X^k)^{k'} - \alpha^{k'} = X - \alpha^{k'} = X - \alpha^{1/k}$$

so $(X - \alpha^{1/k})$ is contained in $(X^k - \alpha)$.

With this equation in mind, we may rewrite the decomposition Equation (5) as

$$(X^{n/\ell} - \eta^{1/i}) = \prod_{j \in H_\ell / H_m} (X^{n/m} - \zeta^{(1/i)(1/j)})$$

With $i' \in \mathbb{Z}_n^\times / H_\ell$ the inverse to i in any subgroup of $\mathbb{Z}_n^\times$ we may further rewrite the decomposition as

$$(X^{n/\ell} - \eta^{i'}) = \prod_{j' \in H_\ell / H_m} (X^{n/m} - \zeta^{i'j'})$$

Since we are indexing over a group (that is H_ℓ / H_m), for every j that appeared in the group previously there is now a unique inverse j', so the two are interchangeable. This equation, however, was already proven in Equation (4), so our proof is complete. □

5.3 Indexing via $\langle m+1 \rangle$ sets

It turns out that in many cases H_m is generated by $\langle m+1 \rangle$. This is the case provided m doesn't have a single factor of 2 in it's prime decomposition, that is 2 may have multiplicity 0 or $k > 1$. We will prove this statement below. Without reintroducing notation, we may rewrite the two decompositions proven above in this alternative form. The first decomposition due to Theorem 11 becomes

$$(\Phi_n(X)) = \prod_{i_1 \in \mathbb{Z}_n^\times / \langle m_1+1 \rangle} \prod_{i_2 \in \langle m_1+1 \rangle / \langle m_2+1 \rangle} \cdots \prod_{i_t \in \langle m_{t-1}+1 \rangle / \langle m_t+1 \rangle} (X^{n/m_t} - \zeta^{i_1 \cdots i_t})$$

The second decomposition due to Theorem 12 becomes

$$(\Phi_n(X)) = \prod_{\sigma_1 \in \mathrm{Gal}(\mathbb{Q}(\mu_n)/\mathbb{Q}) / \left\langle \sigma_{m_1+1} \right\rangle} \prod_{\sigma_2 \in \left\langle \sigma_{m_1+1} \right\rangle / \left\langle \sigma_{m_2+1} \right\rangle} \cdots \prod_{\sigma_t \in \left\langle \sigma_{m_{t-1}+1} \right\rangle / \left\langle \sigma_{m_t+1} \right\rangle} (\sigma_t \circ \cdots \circ \sigma_1)(X^{n/m_t} - \zeta)$$

The following theorem demonstrates when $\langle m+1 \rangle$ generates H_m.

Theorem 13 (When $\langle m+1 \rangle$ generates H_m). *Suppose $n = \prod_i p_i^{e_i}$ and $m = \prod_i p_i^{f_i}$ with $e_i \geq f_i \geq 1$. Moreover, suppose that if $p_i = 2$ then $f_i \geq 2$. Then $\langle m+1 \rangle$ generates the subgroup $H_m \subseteq \mathbb{Z}_n^\times$.*

Proof. As a subgroup of H_m, we know $\langle m+1 \rangle$ has order dividing n/m, and we are left to show it has order *at least* $n/m = \prod_i p_i^{e_i - f_i}$. To do so we'll show that raising $m+1$ to any of the powers $\prod_i p_i^{g_i}$ for $g_i = e_i - f_i$ or $g_i = e_i - f_i - 1$ results in modulo 1 *only* for the trivial case that $\forall i, g_i = e_i - f_i$ in which case $\prod_i p_i^{g_i} = n/m$.

Our core lemma is that power $\prod_i p_i^{g_i}$ for $g_i \geq 0$ of $m+1$ takes the form

$$1 + d \prod_i p_i^{f_i + g_i}$$

for some d coprime with all p_i, that is $(d, \prod_i p_i) = 1$. In the case that $\forall i, g_i = e_i - f_i$ this reduces to

$$1 + d \prod_i p_i^{e_i} = 1 + dn$$

which has residue 1 modulo n. In all other cases, however, there exists at least one j such that $g_j = e_j - f_j - 1$ in which case the expression reduces to

$$1 + d \cdot p_j^{e_j - 1} \prod_{i \neq j} p_i^{f_i + g_i}$$

As n does not divide $p_j^{e_j-1}\prod_{i\neq j} p_i^{f_i+g_i}$ and d is coprime with n, the expression has residue not equal to 1 modulo n.

Now to prove the lemma, we note that the power 1 of $m+1$ takes this form with $\forall i, g_i = 0$ and $d = 1$. For induction, we consider raising $(1 + d\prod_i p_i^{f_i+g_i})$ to power p_j to get power $(1 + d \cdot p_j^{f_i+(g_i+1)} \prod_{i\neq j} p_i^{f_i+g_i})$ and demonstrate it indeed takes the form

$$1 + d'\left(p_j^{f_i+(g_i+1)} \prod_{i\neq j} p_i^{f_i+g_i}\right)$$

for some d' with $(d', \prod_i p_i) = 1$.

We compute the power with binomial expansion as follows.

$$\begin{aligned}
\left(1 + d\prod_i p_i^{f_i+g_i}\right)^{p_j} &= \sum_{k=0}^{p_j} \left(d\prod_i p_i^{f_i+g_i}\right)^k \\
&= 1 + p_j\left(d \cdot p_j^{f_j+g_j}\prod_{i\neq j} p_i^{f_j+g_j}\right) \\
&\quad + \sum_{k=2}^{p_j}\binom{p_j}{k} d^k \cdot p_j^{f_j+g_j+1} \cdot p_j^{(k-1)(f_j+g_j)-1}\prod_{i\neq j} p_i^{k(f_i+g_i)} \\
&= 1 + p_j^{f_j+(g_j+1)}\left(d\prod_{i\neq j} p_i^{f_j+g_j}\right) \\
&\quad + \sum_{k=2}^{p_j}\binom{p_j}{k} d^k \cdot p_j^{(k-1)(f_j+g_j)}\prod_{i\neq j} p_i^{f_i+g_i}\prod_{i\neq j} p_i^{(k-1)(f_i+g_i)} \\
&= 1 + p_j^{f_j+(g_j+1)}\prod_{j\neq j} p_i^{f_i+g_i}\left(d + \sum_{k=2}^{p_j}\binom{p_j}{k} d^k \cdot p_j^{(k-1)(f_j+g_j)-1}\prod_{i\neq j} p_i^{(k-1)(f_i+g_i)}\right) \\
&= 1 + d'\left(p_j^{f_j+(g_j+1)}\prod_{j\neq j} p_i^{f_i+g_i}\right)
\end{aligned}$$

for

$$d' := d + \sum_{k=2}^{p_j}\binom{p_j}{k} d^k \cdot p_j^{(k-1)(f_j+g_j)-1}\prod_{i\neq j} p_i^{(k-1)(f_i+g_i)}$$

Now we must argue that $(d', \prod_i p_i) = 1$. Write $d' = d + T$, and recall that by induction d is coprime with $\prod_i p_i$. On the other hand, T is a sum consisting of $p_j - 1$ terms, and each term is divisible by p_i for $i \neq j$ because $(k-1)(f_i + g_i) \geq 1$ because $k \geq 2$, $f_i \geq 1$, and $g_i \geq 0$. We now also argue that each term of T is divisible by p_j. First note this holds for all terms $k \geq 3$. For the case $k = 2$, note the term is

$$\frac{p_j(p_j-1)}{2} d^2 \cdot p_j^{f_j+g_j-1}\prod_{i\neq j} p_i^{f_i+g_i}$$

and we split analysis by condition on p_j.

- Suppose $p_j = 2$. Then $f_j + g_j - 1 \geq 1$ as $f_j \geq 2$, thus the term is divisible by p_j.
- Suppose $p_j \neq 2$. Then p_j is odd, so $(p_j - 1)/2$ is a whole number and the binomial coefficient $p_j(p_j - 1)/2$ is divisible by p_j.

Thus for every prime p_i (including $i = j$), d is coprime with p_i while T is divisible by p_i. We wish to conclude that $(d', p_i) = 1$ for all i. Suppose $T = p_i \cdot q_i$ for quotient q_i, so $d' = d + p_i \cdot q_i$. Suppose for sake of contradiction that $d' = Q_i \cdot p_i$. Then $p_i(Q_i - q_i) = d$, contradicting $(d, p_i) = 1$, and therefore $(d', p_i) = 1$. As such we have d' coprime with $\prod_i p_i$ as promised. $\square$

There is, however, a simpler way to write these decompositions in terms of $\langle m+1 \rangle$ and $\langle \sigma_{m+1} \rangle$. We can show that the representative $(m_k + 1)$ has order m_{k+1}/m_k in the subgroup $H_{m_k}/H_{m_{k+1}}$, which itself has order m_{k+1}/m_k. Therefore the first m_{k+1}/m_k powers of $(m_k + 1)$ generate elements in H_{m_k} that represent each of the m_{k+1}/m_k cosets of $H_{m_k}/H_{m_{k+1}}$. To see this, note the t'th power of $(m_k + 1)$ takes the form

$$T \cdot m_k^2 + t \cdot m_k + 1$$

for some expression T. The smallest t for which this expression reduces to $1 \bmod m_{k+1}$ is $t = m_{k+1}/m_k$, thus m_{k+1}/m_k is the order of representative $(m_k + 1)$ in the subgroup $H_{m_k}/H_{m_{k+1}}$.

Having established that powers of $(m_k + 1)$ may serve in place of the quotient groups, we may replace the quotient groups with simple integer enumeration (also replacing $\mathbb{Z}_n^\times / \langle m_1 + 1 \rangle$ with $\mathbb{Z}_{m_1}^\times$). The first decomposition now becomes

$$(\Phi_n(X)) = \prod_{i_1 \in \mathbb{Z}_{m_1}^\times} \prod_{i_2=0}^{m_2/m_1 - 1} \cdots \prod_{i_t=0}^{m_t/m_{t-1}-1} (X^{n/m_t} - \zeta^{(m_1+1)^{i_1} \cdots (m_t+1)^{i_t}})$$

The second decomposition now becomes

$$(\Phi_n(X)) = \prod_{i_1 \in \mathbb{Z}_{m_1}^\times} \prod_{i_2=0}^{m_2/m_1 - 1} \cdots \prod_{i_t=0}^{m_t/m_{t-1}-1} (\sigma_t^{i_t} \circ \cdots \circ \sigma_2^{i_2} \circ \sigma_1^{i_1})(X^{n/m_t} - \zeta)$$

where σ_k corresponds to $m_k + 1$.

6 An initial GPU algorithm

This section and the final section following are each dedicated to reasoning about an abstract GPU algorithm for multiplying ring elements in our ring of interest $\mathbb{Z}_p[X]/\Phi_n(X)$, where p is a prime not dividing n and $\Phi_n(X)$ is the n'th cyclotomic polynomial with n a power of 2. Indeed, we restrict to n a power of 2 because the smaller the prime factors of n (2 is optimal) the better for our purposes. There are three reasons for this:

- We want to minimize the growth of the coefficients of ring elements on multiplication, and therefore on polynomial reduction modulo Φ_n. The growth of coefficients on reduction modulo Φ_n is in proportion to the sparsity of Φ_n, which is in proportion to the size of the prime factors of n.

- We want to most efficiently compute the Chinese Remainder Theorem transformation (CRT) in $\mathbb{Z}_p[X]/\Phi_n(X)$, meaning we want $\Phi_n(X)$ to split into sets of polynomials of degrees n/m for as many m as possible provided m shares the same prime factors as n and $m \leq n/\mathrm{ord}_n(p)$. Each such set enables an additional level of recursion for the CRT, so the more sets the more efficient the transformation. The larger the prime factors of n, the fewer the possibilities for m.

- We want to choose a structured prime p to enable tricks for fast modular arithmetic on modern processors. As modern processors operate on binary data, a natural choice is having $p = \Phi_\ell(2)$ for some sparse cyclotomic polynomial Φ_ℓ. The sparsity of Φ_ℓ enables tricks for fast modular reduction, while the fact that Φ_ℓ vanishes on the ℓ'th primitive roots of unity means some powers of 2 represent roots of unity enabling fast twiddle-factor multiplication. In order to control the splitting of Φ_n in $\mathbb{Z}_p[X]$ we need to control $\mathrm{ord}_n(p)$ and with $p = \Phi_\ell(2)$ composed of sums of powers of 2 (for $\ell < 105$) this can basically only be done with n a power of 2.

While the first two priorities are essential to our purposes, they only ask for small prime factors of n, not necessarily 2. While the last priority, on the other hand, is not essential since other efficient number systems exist apart from 'cyclotomic primes,' the last priority indeed restricts to n a power of 2. At the moment we have little reason *not* to choose n a power of 2, so we will continue in this section with n indeed a power of 2.

We approach the problem of multiplying two ring elements by first decomposing the ring elements into irreducible components and then multiplying those components. This section will present an algorithm for decomposing a ring element into irreducibles. The first subsection is about reading the input. The second subsection is about applying decomposition *within* each GPU thread, while the third subsection is about applying decomposition *across* GPU threads when decomposition within threads is no longer applicable.

It turns out there's a faster solution to apply decomposition across GPU threads, and the section following this one develops this second solution. The first solution we develop herein for decomposition within threads, however, is also applied in the solution of the subsequent section. The material of the first three subsections of this section are thus relevant for the second solution. The material of the fourth subsection is also informative, but since we find the alternative solution presented in the subsequent section to be heuristically of higher performance, we continue no further with the material of the fourth subsection. Specifically, upon completing decomposition the components must be multiplied, and the strategy for multiplication depends on the decomposition algorithm. We will develop multiplication algorithms only for the decomposition solution of the subsequent section, and the algorithms are presented therein.

6.1 Introduction

With n a power of 2, the cyclotomic polynomial Φ_n takes the shape

$$\Phi_n(X) = X^D + 1$$

for degree $D := n/2$ (also a power of 2). Let $S > 1$ be a divisor of D. Let T and U also be divisors of D such that $STU = D$. Note that with D a power of 2, so are S, T, and U. We will use these divisors to form a radix number system that enumerates the D coefficients of a polynomial from the set $\mathbb{Z}_p[X]/\Phi_n(X)$. The following set is equal to $[D]-1 \coloneqq \{0,\ldots,D-1\}$

$$\{sTU + tU + u \mid s \in [S]-1,\ t \in [T]-1,\ u \in [U]-1\} \tag{6}$$

where this is first sight of the non-standard notation $[Z]-1$ as a variant of the standard notation $[Z]$ representing $\{1,\ldots,Z\}$.

Suppose we wish to multiply elements $C, R \in \mathbb{Z}_p[X]/\Phi_n(X)$, where C is given in power basis form, while R can be represented in any basis. We will actually not fully multiply these elements as suggested initially. Rather, we'll decompose C using the CRT (Chinese Remainder Theorem) decomposition, assume that R has already been decomposed, multiply the irreducible components, and leave that as our multiplication result rather than inverting back to coefficient form. It turns out this process is sufficient for our purposes in lattice cryptography. Our algorithm will consist of two primary stages. First we compute the CRT form of C, and then we multiply that by the precomputed CRT form of R.

In our GPU context, we will have T 'threads' executing in parallel to compute the multiplication of C and R. In the second subsection, following this first subsection, we will read the input polynomial C into the registers of the T threads, assuming R is loaded separately at some point or is hardcoded into the program. In the third subsection we will focus on computing as much in parallel as possible with no cross-thread communication. In particular, we will show how T threads in parallel can decompose C into a CRT representation of S component polynomials, with each thread holding U coefficients of each polynomial. In the fourth subsection, we will focus on how the threads can cross-communicate in order to decompose further to the CRT representation of irreducible components.

6.2 Reading into registers

Our first task is reading the D coefficients of C into the registers of the T threads. We will do this by having each thread read SU coefficients into its registers. The reading will occur in S steps, at each step each thread reads U values. Specifically, at step $s \in [S]-1$, thread t will read the values at indices

$$\{sTU + tU + u \mid u \in [U]-1\}$$

Each thread may read its SU values into an S-major, U-minor ordered array.

The reason we choose to read according to this enumeration is twofold. First, at each step we want the threads to read from consecutive memory locations as this enables memory reads to be batched across threads. Second, we want threads to read values which can be partially decomposed before needing any cross-thread communication. A thread will decompose S values to another S values, and there are U sets of such S values. It may seem logical then given a fixed reading capacity SU to maximize S and set $U = 1$, thus allowing as much independent thread decomposition as possible. The only reason we may want $U > 1$ is because the values to be read are the coefficients of C which are small, and likely small enough that each thread can read multiple of them in a single read. To determine the amount a thread can read in total, we determine U as the amount a thread can read in a single step, and then we determine S as the number of steps a thread can perform before its registers are full. But maximizing SU may actually not be the best option, as smaller register-space footprint allows more threads to run concurrently. Trial and error determine optimal parameters.

Let us mention an alternative that we will not pursue. Instead of threads decomposing independently followed by cross-thread communication, we could begin with cross-thread communication, and then allow each thread to run independently to completion. The reason we don't pursue this, for now, is it requires adjacent threads to read from non-adjacent memory locations in each step (though across steps a thread reads from adjacent memory locations). It could be that some modern GPUs can still manage to batch such non-adjacent memory accesses, but for now we leave this approach alone. As it turns out, however, this alternative can be had by applying the exact algorithm we discuss in this section apart this first step of reading values into registers. While our intention is $S \gg U$ one may implement the alternative model by having $S \ll U$, in particular with $S = 0$ in which case there is no per-thread decomposition before cross-thread decomposition, and rather all per-thread decomposition can occur after cross-thread decomposition has completed.

6.3 Decomposing within threads

In this subsection we will establish what exactly we wish to decompose, and then we will present the decomposition. For the purpose of implementation, we then reformulate the decomposition using arrays. Last we examine how many times the decomposition may be applied.

6.3.1 What to decompose

Let us write C using our enumeration Equation (6), and to avoid verbose subscripts, we index the coefficients using square brackets. While it may be most intuitive to sum over $[S]-1$ on the outside, $[T]-1$ in the middle and $[U]-1$ on the inside, it turns out we will work with them in the reverse order.

$$\sum_{i=0}^{D-1} C[i]X^i = \sum_{u\in[U]-1} \sum_{t\in[T]-1} \sum_{s\in[S]-1} C[sTU + tU + u]X^{sTU+tU+u} \tag{7}$$

Our goal is to decompose this polynomial using the CRT decomposition. Of the several formulations of the CRT decomposition we've developed, we will use the decomposition by H_m sets from Theorem 11. Recall such a decomposition is parameterized by an increasing sequence of values, each dividing the next, and the last dividing n. With n a power of 2, our sequence will be

$$m_1 = 2^1,\ m_2 = 2^2,\ \ldots,\ m_k = 2^k,\ m_{k+1} = 2^{k+1} \le n$$

We also need η, a primitive $m_{k+1} = 2^{k+1}$'th root of unity. As discussed at the end of the previous section, $m_k + 1$ has the same order as $H_{m_k}/H_{m_{k+1}}$, namely $m_{k+1}/m_k = 2^{k+1}/2^k = 2$, so we will replace enumeration over $H_{m_k}/H_{m_{k+1}}$ with enumeration over powers of m_k+1 to get enumeration set $\{1, m_k + 1\} = \{1, 2^k + 1\}$. Before writing C using the decomposition, let us review the decomposition applied to Φ_n

$$\begin{aligned}
\Phi_n(X) &= \prod_{i_0\in\mathbb{Z}_{m_1}^\times}\ \prod_{i_1\in H_{m_1}/H_{m_2}} \cdots \prod_{i_k\in H_{m_k}/H_{m_{k+1}}} X^{n/m_{k+1}} - \eta^{i_0\cdot i_1\cdots i_k} \\
&= \prod_{i_0\in\mathbb{Z}_2^\times}\ \prod_{i_1\in\{1,2^1+1\}} \cdots \prod_{i_k\in\{1,2^k+1\}} X^{(2D)/2^{k+1}} - \eta^{i_0\cdot i_1\cdots i_k} \\
&= \prod_{i_1\in\{1,2^1+1\}} \cdots \prod_{i_k\in\{1,2^k+1\}} X^{D/2^k} - \eta^{i_1\cdots i_k} \\
&= \prod_{i_1\in\{0,1\}} \cdots \prod_{i_k\in\{0,1\}} X^{D/2^k} - \eta^{\epsilon(i_1,\dots i_k)} \\
&= \prod_{\boldsymbol{i}\in\{0,1\}^k} X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})} \\
\epsilon(\boldsymbol{i}) \coloneqq \epsilon(i_1,\dots i_k) &\coloneqq \prod_{z=1}^{k} (1 + i_z \cdot 2^z)
\end{aligned}$$

where we are overloading ϵ to accept either k bit-values or a vector of k bits. Note these bits are indexed with $[k]$ rather than $[k]-1$. We will use such vectors to identify components in the direct product decomposition. But for convenience we will map them to the natural numbers and typically index with the natural numbers instead. For binary vector $\boldsymbol{i} = (i_1,\dots i_{|\boldsymbol{i}|}) \in \{0,1\}^{|\boldsymbol{i}|}$ of length $|\boldsymbol{i}| \geq k$ we define α_k as

$$\alpha_k(\boldsymbol{i}) \coloneqq \alpha_k(i_1,\dots i_{|\boldsymbol{i}|}) \coloneqq \sum_{z=0}^{k-1} i_{k-z}\cdot 2^z$$

noting how α_k ignores the last $|\boldsymbol{i}|-k$ coordinates of $\boldsymbol{i}$. The function α_k maps the first k coordinates of $\boldsymbol{i}$ to the set $[2^k]-1$, with i_1 as the most significant digit and i_k as the least significant digit. For $i \in [2^{k_1+k_2}]-1$ with $k_2 \geq 0$ one may think of $\alpha_{k_1}(\boldsymbol{i})$ as the right-shift of i by k_2 bits. We will use $\boldsymbol{i} \in \{0,1\}^k$ and $i = \alpha_k(\boldsymbol{i}) \in [2^k]-1$ interchangeably as one can be recovered from the other. For example, the component with index $i \in [2^k]-1$ of the decomposition corresponds to factor $X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}$ where $\boldsymbol{i}$ is the unique pre-image of i under α_k. A useful property we will use is that

$$\alpha_{k+1}(i_1,\dots,i_k,i_{k+1}) = 2\cdot\alpha_k(i_1,\dots,i_k) + i_{k+1}$$

or for $i \in [2^k]-1$ and $b \in \{0,1\}$ one may write $\alpha_{k+1}(\boldsymbol{i}, b) = 2i + b$.

As a special case of this factorization of Φ_n, we can recognize that with $k=0$ our sequence is $m_1 = 2$, our primitive 2nd root of unity is $\eta = -1$, and we recover

$$\Phi_n(X) = \prod_{i_0\in\mathbb{Z}_2^\times} X^{n/2} - (-1)^{i_0} = X^D + 1$$

We can use this factorization of Φ_n to represent our corresponding direct product decomposition. Let C_k denote the corresponding decomposition of depth $k \geq 0$ with the component polynomials $C_k^{(i)}$ for $i \in [2^k] - 1$

$$C_k^{(i)}(X) \coloneqq C(X) \bmod \left(X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}\right) \in \frac{\mathbb{Z}_p[X]}{X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}}$$
$$C_k(X) \coloneqq \prod_{i \in [2^k]-1} C_k^{(i)}(X) \in \prod_{i \in [2^k]-1} \frac{\mathbb{Z}_p[X]}{X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}}$$

where we're using the symbol $\prod$ over the $C_k^{(i)}$ to denote direct product rather than arithmetic product.

Our goal is to compute $C_k^{(i)}$ for $i \in [2^k] - 1$ for the largest k applicable (to be discussed at the end of the subsection). For clarity on what computing $C_k^{(i)}$ entails, we expand it using a variation of our enumeration Equation (6). Previously we enumerated the D coefficients of C with most significant digit $s \in [S] - 1$, such that with lower digits $t \in [T] - 1$ and $u \in [U] - 1$ the digit combinations span $[STU] - 1 = [D] - 1$. Now we'd like to generalize by enumerating the $D/2^k$ coefficients of $C_k^{(i)}$, so we alter our most significant digit to be $s \in [S/2^k] - 1$, such that with identical lower digits the digit combinations now span $[(S/2^k)TU] - 1 = [D/2^k] - 1$. Also, instead of summing over $[S] - 1$ as the *outside* sum as in Equation (7) we will instead sum over $[S/2^k] - 1$ as the *inside* sum, and furthermore we give this inside sum the name $\overline{C}_k^{(i)}$.

$$\overline{C}_k^{(i)}(X, t, u) \coloneqq \sum_{s \in [S/2^k]-1} C_k^{(i)}[sTU + tU + u] X^{sTU+tU+u}$$
$$C_k^{(i)}(X) = \sum_{u \in [U]-1} \sum_{t \in [T]-1} \overline{C}_k^{(i)}(X, t, u)$$

Note the latter equation reduces to Equation (7) for the case $k = 0$. We write C_k in terms of our newly defined $\overline{C}_k^{(i)}$ as

$$C_k(X) = \prod_{i \in [2^k]-1} \sum_{u \in [U]-1} \sum_{t \in [T]-1} \overline{C}_k^{(i)}(X, t, u)$$
$$= \sum_{u \in [U]-1} \sum_{t \in [T]-1} \prod_{i \in [2^k]-1} \overline{C}_k^{(i)}(X, t, u)$$

We may henceforth in this subsection focus on the inner direct product, consisting of the 2^k polynomials (in X) $\overline{C}_k^{(i)}$. This direct product is parameterized by t and u, and indeed each thread t will compute its own set of direct products, one direct product for each $u \in [U] - 1$. Once all T threads have completed their U direct products, we may sum over $[U] - 1$ and $[T] - 1$ to obtain C_k. Decomposing C_k further when it is a sum distributed as such among threads will be the topic of the next subsection, and it will require cross-thread communication.

6.3.2 How to decompose with polynomials

The direct product $\prod_{i \in [2^k]-1} \overline{C}_k^{(i)}(X, t, u)$ is what we wish to decompose, meaning that given this direct product for some appropriate k, we wish to compute it for $k+1$. For the full decomposition, we begin with $k = 0$ and proceed in steps by incrementing k until we can decompose no further. At step k we have a direct product of 2^k polynomials $\overline{C}_k^{(i)}$ for $i \in [2^k] - 1$ such that

$$\prod_{i \in [2^k]-1} \overline{C}_k^{(i)}(X) \in \prod_{i \in [2^k]-1} \frac{\mathbb{Z}_p[X]}{X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}}$$

We wish to decompose it into a direct product of 2^{k+1} polynomials $\overline{C}_{k+1}^{(i')}$ for $i' \in [2^{k+1}] - 1$ such that

$$\prod_{i' \in [2^{k+1}]-1} \overline{C}_{k+1}^{(i')}(X) \in \prod_{i' \in [2^{k+1}]-1} \frac{\mathbb{Z}_p[X]}{X^{D/2^{k+1}} - \zeta^{\epsilon(\boldsymbol{i'})}} \tag{8}$$

where ζ is a primitive 2^{k+2}'th root of unity satisfying $\zeta^2 = \eta$. This decomposition occurs by every component $X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}$ splitting into two components $X^{D/2^k} - \zeta^{\epsilon(\boldsymbol{i'})}$ for $\boldsymbol{i'} = (\boldsymbol{i}, b)$ with $b \in \{0, 1\}$, also written $i' = 2i + b$. To see this splitting, note that $\zeta^{2^{k+1}} = -1$, and also that $\epsilon(\boldsymbol{i}, b) = \epsilon(\boldsymbol{i})(1 + b2^{k+1})$. Then we may write

$$\begin{aligned}
&(X^{D/2^{k+1}} - \zeta^{\epsilon(\boldsymbol{i},0)})(X^{D/2^{k+1}} - \zeta^{\epsilon(\boldsymbol{i},1)}) \\
&\qquad = X^{D/2^k} - X^{D/2^{k+1}}\zeta^{\epsilon(\boldsymbol{i})}(\zeta^{1+0} + \zeta^{1+2^{k+1}}) + \zeta^{\epsilon(\boldsymbol{i})(1+0)}\zeta^{\epsilon(\boldsymbol{i})(1+2^{k+1})} \\
&\qquad = X^{D/2^k} - X^{D/2^{k+1}}\zeta^{\epsilon(\boldsymbol{i})}(\zeta + \zeta(-1)) + \zeta^{2\cdot\epsilon(\boldsymbol{i})}(-1) \\
&\qquad = X^{D/2^k} - \eta^{\epsilon(\boldsymbol{i})}
\end{aligned}$$

We will decompose $\overline{C}_k^{(i)}$ by replacing $X^{D/2^{k+1}}$ with the two values $\zeta^{\epsilon(\boldsymbol{i},b)}$ for $b \in \{0, 1\}$. But as ζ^{1+0} and $\zeta^{1+2^{k+1}} = \zeta(-1)$ are additive inverses, we will instead use the two values $(-1)^b\zeta^{\epsilon(\boldsymbol{i})}$. Our decomposition may be written as follows with $t \in [T] - 1$, $u \in [U] - 1$, and $i \in [2^k] - 1$ (the range for k to be discussed last in this subsection)

$$\begin{aligned}
\overline{C}_k^{(i)}(X, t, u) &= \sum_{s \in [S/2^k]-1} C_k^{(i)}[sTU + tU + u]X^{sTU+tU+u} \\
&= \sum_{s \in [S/2^{k+1}]-1} C_k^{(i)}[sTU + tU + u]X^{sTU+tU+u} \\
&\quad + \sum_{s \in [S/2^{k+1}]-1} C_k^{(i)}[(s + S/2^{k+1})TU + tU + u]X^{(s+S/2^{k+1})TU+tU+u} \\
&= \sum_{s \in [S/2^{k+1}]-1} C_k^{(i)}[sTU + tU + u]X^{sTU+tU+u} \\
&\quad + X^{D/2^{k+1}} \cdot \sum_{s \in [S/2^{k+1}]-1} C_k^{(i)}[D/2^{k+1} + sTU + tU + u]X^{sTU+tU+u} \\
&= \sum_{s \in [S/2^{k+1}]-1} \Big(\\
&\quad C_k^{(i)}[sTU + tU + u] + (-1)^b\zeta^{\epsilon(\boldsymbol{i})}C_k^{(i)}[D/2^{k+1} + sTU + tU + u] \\
&\Big)X^{sTU+tU+u}
\end{aligned}$$

We have decomposed $\overline{C}_k^{(i)}$ into two polynomials that correspond to the two factors $X^{D/2^{k+1}} + \zeta^{\epsilon(\boldsymbol{i},b)}$ within our desired direct product decomposition Equation (8). Therefore we may assign the two polynomials $\overline{C}_k^{(2i+b)}$ to the two decomposed results

$$\begin{aligned}\overline{C}_{k+1}^{(2i+b)}(X,t,u) &= \sum_{i\in[S/2^{k+1}]-1} C_{k+1}^{(2i+b)}[sTU+tU+u]X^{sTU+tU+u} \\ &= \sum_{i\in[S/2^{k+1}]-1} \Big(\\ &C_k^{(i)}[sTU+tU+u] + (-1)^b\zeta^{\epsilon(\boldsymbol{i})}C_k^{(i)}[D/2^{k+1}+sTU+tU+u] \\ &\Big)X^{sTU+tU+u}\end{aligned}$$

As the polynomials are equal, so are their coefficients, yielding our decomposition formula $\forall s \in [D/2^{k+1}]-1$

$$C_{k+1}^{(2i+b)}[sTU+tU+u] = \tag{9}$$
$$C_k^{(i)}[sTU+tU+u] + (-1)^b\zeta^{\epsilon(\boldsymbol{i})}C_k^{(i)}[D/2^{k+1}+sTU+tU+u] \tag{10}$$

We have written the formula using the coefficients of $C_k^{(i)}$ and $C_{k+1}^{(2i+b)}$. But the formula can also be written using the coefficients of $\overline{C}_k^{(i)}$ and $\overline{C}_{k+1}^{(2i+b)}$, though they must be parameterized by a t and u in that coefficient $sTU+tU+u$ of $C_k^{(i)}(X)$ and $\overline{C}_k^{(i)}(X,t,u)$ coincide, and similarly for $C_{k+1}^{(2i+b)}$ and $\overline{C}_{k+1}^{(2i+b)}$. Our use of $\overline{C}_k^{(i)}$ is to identify it as an important polynomial had via $C_k^{(i)}$, and to identify it without summing over $[S/2^k]-1$ each time. Now that we have a decomposition formula in coefficient form, we will henceforth, whenever using coefficient form, use coefficients of $C_k^{(i)}$ rather than those of $\overline{C}_k^{(i)}$.

6.3.3 How to decompose with arrays

The formula in Equation (10) is the decomposition formula one may use to decompose $\overline{C}_k^{(i)}$. In it's current form this formula applies independently for each t and u. We will now show how thread t can, for each invocation k of the decomposition, use an array $A_k^{(t)}$ to encode $\overline{C}_k^{(i)}(X,t,u)$ for all $i \in [2^k]-1$ and $u \in [U]-1$. In practice, thread t would not have a separate array $A_k^{(t)}$ for each k but rather have a single array $A^{(t)}$ mutated in-place.

Recall our choice to use coefficients of $C_k^{(i)}(X)$ given t and u values rather than the same and equal coefficients of $\overline{C}_k^{(i)}(X,t,u)$. Given this choice and the fact that arrays will encode coefficients, we proceed with $C_k^{(i)}(X)$ rather than $\overline{C}_k^{(i)}(X,t,u)$, though we will note how an array indeed encodes $\overline{C}_k^{(i)}(X,t,u)$. Suppose we wish to enumerate all $STU/2^k$ coefficients of all 2^k polynomials $C_k^{(i)}$ for $i \in [2^k]-1$ in component-major order. Then our enumeration set is

$$\left\{ i(S/2^k)TU + sTU + tU + u \;\middle|\; \begin{array}{l} i \in [2^k]-1,\ s \in [S/2^k]-1 \\ t \in [T]-1,\ u \in [U]-1 \end{array} \right\}$$

Note that for $k = 0$ this enumeration reduces to enumeration Equation (6). We use this enumeration set to connect the enumeration of components and their coefficients to arrays and their entries. When enumerating components we use $i \in [2^k]-1$ and therefore when enumerating the coefficients of components we discard i and its scaling factor $(S/2^k)TU$ from the enumeration to recover $C_k^{(i)}[sTU + tU + u]$. Similarly, when enumerating arrays we use t since there's one array for each thread. Therefore when enumerating the entries of arrays we discard t and its scaling factor T from the enumeration to get $A_k^{(t)}[i(S/2^k)U + sU + u]$. We thus define our array entries as

$$\forall i \in [2^k] - 1,\ \forall s \in [S/2^k] - 1,\ \forall u \in [U] - 1 :$$
$$A_k^{(t)}[i(S/2^k)U + sU + u] := C_k^{(i)}[sTU + tU + u]$$

Note how array t indeed encodes exactly the SU coefficients of $\overline{C}_k^{(i)}(X, t, u)$.

Given the definition of $A_k^{(t)}$, we now may adapt our decomposition formula Equation (10) for the encoding of $A_k^{(t)}$. The two split halves of $\overline{C}_k^{(i)}(X, t, u)$, that is its upper and lower halves, are had by replacing $s \in [S/2^k] - 1$ with $s \in [S/2^{k+1}] - 1$ and respectively replacing s with $b(S/2^{k+1}) + s$ for $b \in \{0, 1\}$. Doing the same here we have

$$\forall i \in [2^k] - 1,\ \forall s \in [S/2^{k+1}] - 1,\ \forall u \in [U] - 1 : \tag{11}$$
$$A_k^{(t)}[(2i + b)(S/2^{k+1})U + sU + u] := C_k^{(i)}[b(D/2^{k+1}) + sTU + tU + u] \tag{12}$$

We may now construct an analogous formula to Equation (10) by copying Equation (10) and making appropriate translations using our defining equation for arrays along with Equation (12)

$$\forall i \in [2^k] - 1,\ \forall b \in \{0, 1\},\ \forall s \in [S/2^{k+1}] - 1,\ \forall t \in [T] - 1,\ \forall u \in [U] - 1 :$$
$$A_{k+1}^{(t)}[(2i + b)(S/2^{k+1})U + sU + u] =$$
$$A_k^{(t)}[(2i)(S/2^{k+1})U + sU + u] + (-1)^b \zeta^{\epsilon(\boldsymbol{i})} A_k^{(t)}[(2i + 1)(S/2^{k+1})U + sU + u]$$

6.3.4 How many times to decompose

Lastly for this subsection we consider how many applications of the decomposition threads may apply before the polynomial becomes irreducible, noting that $\log(S)$ is an upper bound because the decomposition is applied to sets of size S. The degree of irreducibles is $\mathrm{ord}_n(p) - 1$, so by considering $X^{D/2^{k+1}}$ we conclude the decomposition may only be applied when $D/2^{k+1} \geq \mathrm{ord}_n(p)$, that is when $k < \log(D/\mathrm{ord}_n(p))$. One may also observe the degree of the polynomial to be up to $(S/2^k - 1)TU + (T - 1)U + (U - 1) = D/2^k - 1$, confirming that the decomposition is *non-applicable* when $D/2^k - 1 \leq \mathrm{ord}_n(p) - 1$, that is $k \geq \log(D/\mathrm{ord}_n(p))$. Each thread may then apply the decomposition no more than $\log(D/\mathrm{ord}_n(p))$ times with $k = \{0, \ldots, \log(D/\mathrm{ord}_n(p)) - 1\}$. Depending on S, however, one may need to stop short of all $\log(D/\mathrm{ord}_n(p))$ applications. If $k \geq \log(S)$ we cannot split the enumeration set $[S/2^k] - 1 = \{0\}$ in half. Therefore one may apply the decomposition a maximum of $\min\{\log(D/\mathrm{ord}_n(p)), \log(S)\}$ times for $k = \{0, \ldots, \min\{\log(D/\mathrm{ord}_n(p)), \log(S)\} - 1\}$. In practice we will always choose $S \leq D/\mathrm{ord}_n(p)$ because we strategically choose the set S in order to decompose, and decomposition doesn't apply for $S > D/\mathrm{ord}_n(p)$. We may finally state that we will apply the decomposition exactly $\log(S)$ times for $k = \{0, \ldots, \log(S) - 1\}$.

6.4 Decomposing across threads

After applying the decomposition from the previous subsection $\log(S)$ times within each thread, each thread t holds the set of polynomials $\overline{C}^{(i)}_{\log(S)}$ for $i \in [S] - 1$ satisfying

$$\begin{aligned}
\overline{C}^{(i)}_{\log(S)}(X, t, u) &= C^{(i)}_{\log(S)}[tU + u]X^{tU+u} \\
C^{(i)}_{\log(S)}(X) &= \sum_{u\in[U]-1} \sum_{t\in[T]-1} \overline{C}^{(i)}_{\log(S)}(X, t, u) \\
&= \sum_{u\in[U]-1} \sum_{t\in[T]-1} C^{(i)}_{\log(S)}[tU + u]X^{tU+u}
\end{aligned}$$

Each polynomial $C^{(i)}_{\log(S)}$ is of degree TU yet may still be decomposed into $TU/\mathrm{ord}_n(p)$ polynomials of degree $\mathrm{ord}_n(p)$. Having already applied the decomposition $\log(S)$ times out of the maximum $\log(D/\mathrm{ord}_n(p))$ possible times, the maximum number of times we may further apply the decomposition is

$$\log(D/\mathrm{ord}_n(p)) - \log(S) = \log(D/S) - \log(\mathrm{ord}_n(p)) = \log(TU) - \log(\mathrm{ord}_n(p))$$

For convenience we define $\ell := k - \log(S)$, and for $i \in [2^k] - 1 = [S2^\ell] - 1$ we focus on decomposing the inner sum for each $u \in [U] - 1$

$$\sum_{t\in[T/2^\ell]-1} C^{(i)}_{\log(S)+\ell}[tU + u]X^{tU+u}$$

in steps $\ell \in \{0, \ldots, \log(TU)-\log(\mathrm{ord}_n(p))-1\}$ analogous to our previous steps $k \in \{0, \ldots, \log(S)-1\}$.

6.4.1 How to decompose with polynomials

Writing the decomposition analogous to before for $i \in [S2^\ell] - 1$ yields

$$\begin{aligned}
&\sum_{t\in[T/2^\ell]-1} C^{(i)}_{\log(S)+\ell}[tU+u]X^{tU+u} \\
&\qquad = \sum_{t\in[T/2^{\ell+1}]-1} C^{(i)}_{\log(S)+\ell}[tU+u]X^{tU+u} \\
&\qquad\quad + \sum_{t\in[T/2^{\ell+1}]-1} C^{(i)}_{\log(S)+\ell}[(t+T/2^{\ell+1})U+u]X^{(t+T/2^{\ell+1})U+u} \\
&\qquad = \sum_{t\in[T/2^{\ell+1}]-1} C^{(i)}_{\log(S)+\ell}[tU+u]X^{tU+u} \\
&\qquad\quad + X^{TU/2^{\ell+1}} \cdot \sum_{t\in[T/2^{\ell+1}]-1} C^{(i)}_{\log(S)+\ell}[TU/2^{\ell+1}+tU+u]X^{tU+u} \\
&\qquad = \sum_{t\in[T/2^{\ell+1}]-1} \Big(\\
&\qquad\quad C^{(i)}_{\log(S)+\ell}[tU+u] + (-1)^b\zeta \cdot C^{(i)}_{\log(S)+\ell}[TU/2^{\ell+1}+tU+u] \\
&\qquad\quad \Big)X^{tU+u}
\end{aligned}$$

where as before we have replaced $X^{TU/2^{l+1}} = X^{D/2^{k+1}}$ with $(-1)^b\zeta$ for $b \in \{0,1\}$ where ζ is a primitive $S2^{l+2} = 2^{k+2}$'th primitive root of unity. We derive the decomposition formula $\forall t \in [T/2^\ell] - 1$ in terms of coefficients as

$$C^{(2i+b)}_{\log(S)+\ell+1}[tU+u] = \tag{13}$$
$$C^{(i)}_{\log(S)+\ell}[tU+u] + (-1)^b\zeta \cdot C^{(i)}_{\log(S)+\ell}[TU/2^{\ell+1}+tU+u] \tag{14}$$

6.4.2 How to decompose with arrays

Our task now is to adapt this decomposition formula to arrays. The decomposition using polynomials is independent of how exactly coefficients are distributed among threads and layed out in the arrays. When adapting the decomposition to arrays, these details are crucial and require some examination. Our state when we begin cross-thread communication consists of threads $t \in [T] - 1$ each holding an array $A^{(t)}_{\log(S)}$ which encodes S sets of U elements. Each set of U elements belongs to a different component $C^{(i)}_{\log(S)}$ for $i \in [S] - 1$. Upon decomposing ℓ times further, our direct product decomposition will consist of $2^k = S2^\ell$ components. Each thread, however, will continue to hold an array that encodes only S sets of U elements, and therefore can encode a set of U elements for only S out of all $S2^\ell$ components. Naturally, then, we will partition the $S2^\ell$ components into 2^ℓ groups with S components in each group. Along with partitioning the components, we will also partition the threads into 2^ℓ groups with $T/2^\ell$ threads in each group. Note both partitions have 2^ℓ groups and indeed each group of threads will hold one group of components.

Our new notation will consist of an identifier for each group, and identifiers for each component and each thread in a given group. Let us define for $\boldsymbol{i}$ with $|\boldsymbol{i}| \geq \log(S)$

$$\begin{aligned}\beta_\ell(\boldsymbol{i}) &\coloneqq \beta(i_1, \ldots, i_{\log(S)}, i_{\log(S)+1} \ldots, i_{\log(S)+\ell}) \\ &\coloneqq \sum_{z=1}^{\ell} i_{\log(S)+z} \cdot (T/2^z)\end{aligned}$$

noting $\beta_\ell(\boldsymbol{i})$ depends on only the last ℓ coordinates. A useful property we will use is that for $i \in [S2^\ell] - 1$ and $b \in \{0, 1\}$

$$\beta_{\ell+1}(\boldsymbol{i}, b) = \beta_\ell(\boldsymbol{i}) + b(T/2^{\ell+1}) \tag{15}$$

We will use β_ℓ to identify the 2^ℓ groups, implying that component i belongs to the group with index $\beta_\ell(\boldsymbol{i})$. The index of thread $t \in [T/2^\ell] - 1$ in group $\beta_\ell(\boldsymbol{i})$, among all T threads, may be calculated as $\beta_\ell(\boldsymbol{i}) + t$, and we use this notation to identify the thread's corresponding array $A^{(\beta_\ell(\boldsymbol{i})+t)}_{\log(S)+\ell}$. We will use $\alpha_{\log(S)}$, on the other hand, to identify the S components in a group. Recall that

$$\alpha_{\log(S)}(\boldsymbol{i}) = \sum_{z=0}^{\log(S)-1} i_{\log(S)-z} \cdot 2^z$$

noting $\alpha_{\log(S)}(\boldsymbol{i})$ depends on only the first $\log(S)$ coordinates. We will have $\alpha_{\log(S)}(\boldsymbol{i})$ the index of component $i \in [S2^\ell]-1$ within group $\beta_\ell(\boldsymbol{i})$. Given a component $C^{(i)}_{\log(S)+\ell}$, and given the coefficient index $tU + u$ at which to access the component, let us identify how to access this coefficient using arrays. With i the component index, we know $\beta_\ell(\boldsymbol{i})$ to be the group index. With $tU + u$ the coefficient index, we know $\beta_\ell(\boldsymbol{i}) + t$ to be the index of the thread that holds this coefficient. Lastly we must determine at what index of the array we may access the coefficient. Array $A^{(\beta_\ell(\boldsymbol{i})+t)}_{\log(S)+\ell}$ holds S sets of U elements, each set belonging to a component in the group $\beta_\ell(\boldsymbol{i})$. We target the i'th component, which is the $\alpha_{\log(S)}(\boldsymbol{i})$'th component in the group $\beta_\ell(\boldsymbol{i})$, so we target the $\alpha_{\log(S)}(\boldsymbol{i})$'th set of U elements in the array. We then define $A^{(\beta_\ell(\boldsymbol{i})+t)}_{\log(S)+\ell}$ as

$$\begin{gathered}\forall i \in [S2^\ell] - 1,\ \forall t \in [T/2^\ell] - 1,\ \forall u \in [U] - 1 : \\ A^{(\beta_\ell(\boldsymbol{i})+t)}_{\log(S)+\ell}[\alpha_{\log(S)}(\boldsymbol{i})U + u] \coloneqq C^{(i)}_{\log(S)+\ell}[tU + u]\end{gathered}$$

At this point we've defined how our arrays (for $k \geq \log(S)$) represent the components. To arrive at the decomposition formula Equation (14) in terms of arrays, we must see how the splitting occurs in array form. The two split halves of $C^{(i)}_{\log(S)+\ell}$ are had by replacing $t \in [T/2^\ell] - 1$ with $t \in [T/2^{\ell+1}] - 1$ and respectively replacing t with $b(T/2^{\ell+1} + t)$ for $b \in \{0, 1\}$. Doing the same here we have

$$\forall i \in [S2^\ell] - 1,\ \forall t \in [T/2^{\ell+1}] - 1,\ \forall u \in [U] - 1 : \tag{16}$$

$$A^{(\beta_{\ell+1}(\boldsymbol{i},b)+t)}_{\log(S)+\ell}[\alpha_{\log(S)}(\boldsymbol{i})U + u] = C^{(i)}_{\log(S)+\ell}[b(TU/2^{l+1}) + tU + u] \tag{17}$$

where we have used the fact that, with $\boldsymbol{i'} := (\boldsymbol{i}, b)$, we have

$$\begin{aligned}\beta_\ell(\boldsymbol{i}) + b(T/2^{l+1}) &= \sum_{j=1}^{\ell} i_{\log(S)+j} \cdot (T/2^j) + b(T/2^{l+1}) \\ &= \sum_{j=1}^{\ell+1} i'_{\log(S)+j} \cdot (T/2^j) \\ &= \beta_{\ell+1}(\boldsymbol{i'}) = \beta_{\ell+1}(\boldsymbol{i}, b)\end{aligned}$$

We may now construct an analogous formula to Equation (14) by copying Equation (14) and making appropriate translations using our defining equation for arrays along with Equation (17)

$$\begin{aligned}&\forall i \in [S2^\ell] - 1,\ \forall t \in [T/2^{\ell+1}] - 1,\ \forall u \in [U] - 1 : \\ &A^{(\beta_{\ell+1}(\boldsymbol{i'})+t)}_{\log(S)+\ell+1}[\alpha_{\log(S)}(\boldsymbol{i})U + u] = \\ &\qquad A^{(\beta_{\ell+1}(\boldsymbol{i},0)+t)}_{\log(S)+\ell}[\alpha_{\log(S)}(\boldsymbol{i})U + u] + (-1)^b \zeta \cdot A^{(\beta_{\ell+1}(\boldsymbol{i},1)+t)}_{\log(S)+\ell}[\alpha_{\log(S)}(\boldsymbol{i})U + u]\end{aligned}$$

where $\boldsymbol{i'} := (\boldsymbol{i}, b)$. This decomposition formula may be applied $\log(TU) - \log(\mathrm{ord}_n(p))$ times. We can compare this to the corresponding formula for decomposing within individual threads. There the two source arrays were the same, and we accessed them at different locations. Here, on the other hand, the two source arrays are different, and we access them at the same location.

6.4.3 For implementation

We can simplify the array formula for implementation. Note that the ℓ least significant bits of i determine $\beta_{\ell+1}(\boldsymbol{i}, b)$ for $b \in \{0, 1\}$, while the $\log(S)$ most significant bits determine $\alpha_{\log(S)}(\boldsymbol{i})$. Let us then split i into it's ℓ least significant bits and $\log(S)$ most significant bits, by representing i with $s \in [S] - 1$ and $r \in [2^\ell] - 1$ as $i = s2^\ell + r$. Note that $i_{\log(S)+z} = r_{\ell-z}$ for $z \in [\ell]$ as we will use this fact below.

Not only is $\alpha_{\log(S)}(\boldsymbol{i})$ determined by s, but the two are in fact equal. To see this, recall that when introducing α_k, we mentioned in passing that with $i \in [2^{\log(S)+\ell}]$ we have $\alpha_{\log(S)}(\boldsymbol{i})$ equal to the right-shift of i by ℓ bits, which is indeed equal to s. Thus we can replace $\alpha_{\log(S)}(\boldsymbol{i})$ with s.

Now let us focus on simplifying the array identifier $\beta_{\ell+1}(\boldsymbol{i}, b) + t$. Applying property Equation (15) to $\beta_{\ell+1}(\boldsymbol{i}, b)$ followed by factoring out $T/2^{\ell+1}$ we may write

$$\begin{aligned}\beta_{\ell+1}(\boldsymbol{i}, b) + t &= \beta_\ell(\boldsymbol{i}) + b(T/2^{\ell+1}) + t \\ &= \left(2\sum_{z=1}^{\ell} i_{\log(S)+z} \cdot 2^{\ell-z} + b\right)(T/2^{\ell+1}) + t \\ &= \left(2\sum_{z=1}^{\ell} r_{\ell-z} \cdot 2^{\ell-z} + b\right)(T/2^{\ell+1}) + t \\ &= (2r + b)(T/2^{\ell+1}) + t\end{aligned}$$

Having rewritten $\alpha_{\log(S)}(\boldsymbol{i})$ as s, and rewritten $\beta_{\ell+1}(\boldsymbol{i}, b) + t$ as $(2r + b)(T/2^{\ell+1}) + t$, we may now rewrite our array formula as follows for $b \in \{0, 1\}$.

$$\begin{aligned} &\forall t \in [T/2^{\ell+1}] - 1,\ \forall r \in [2^{\ell}] - 1,\ \forall s \in [S] - 1,\ \forall u \in [U] - 1 : \\ &A_{\log(S)+\ell+1}^{((2r+b)(T/2^{\ell+1})+t)}[sU + u] \\ &\qquad = A_{\log(S)+\ell}^{((2r)(T/2^{\ell+1})+t)}[sU + u] + (-1)^b \zeta \cdot A_{\log(S)+\ell}^{((2r+1)(T/2^{\ell+1})+t)}[sU + u] \end{aligned}$$

While this formula may not look much simpler than our pervious formula, it is more suitable for thread $\tau \in [T] - 1$ to determine with which other thread it must exchange information in round ℓ of the decomposition. Thread $\tau = (2r + b)(T/2^{\ell+1}) + t$ must exchange information with thread $\sigma = (2r + (1 - b))(T/2^{\ell+1}) + t$, that is the thread with address had by bit-flipping the $(T/2^{\ell+1})$'th bit of τ. Upon exchanging information, thread τ merges the two terms in the formula using $(-1)^b$, while σ merges using $(-1)^{1-b}$.

7 An improved GPU algorithm

We discuss decomposition in Section 7.1, arranging irreducible components for multiplication in Section 7.2, and finally three algorithms for multiplication in Section 7.3.

7.1 A second solution

The GPU algorithm for decomposition within and across threads of the previous section may be followed by multiplication of components. One way to do so is to make set $U \geq \mathrm{ord}_n(p)$ and then with $SU/\mathrm{ord}_n(p)$ irreducible components within each thread one may apply any multiplication algorithm without concern for further cross-thread communication. If $U < \mathrm{ord}_n(p)$ one must instead develop some cross-thread multiplication algorithm. In this section, however, we don't pursue either of these paths, but rather develop an alternative solution for decomposition within and across thread, and then we develop multiplication algorithms specialized for our new decompositions.

Let us articulate why we have developed an alternative decomposition solution. Our decomposition solution of the previous section is most efficient when decomposing within threads, and more costly when decomposing across threads. There occur $\log(S)$ decompositions within threads, and $\log(T)$ decompositions across threads, so one strategy is to minimize T and maximize S. But this strategy is at odds with the fact that concurrency scales with T, that is we prefer more threads with less inputs over less threads with more inputs. It turns out there's a solution to this conflict, allowing $\log(S)$ decompositions to occur within threads more than once, the effect being less needed cross-thread communication.

The strategy is an extension of our 'decomposing within threads' strategy of Section 6.3. The idea is once k reaches $\log(S)$ and decomposition within threads is no longer applicable, we swap data between threads such that each thread is loaded with another SU values which can again be decomposed locally within the thread, providing another $\log(S)$ iterations of decomposition. We continue this process until all decomposition has occurred, ultimately requiring roughly $\log(T)/\log(S)$ rather than $\log(T)$ rounds of cross-thread communication. We will formalize this sketch, but with two differences. First, since the purpose of this approach is to maximize the number of decompositions within threads, and noting that a thread holding SU values can perform

$\log(S)$ decompositions, we will set $U = 1$. Second, the number of decompositions that may be applied in total may not be a multiple of S, so we generalize to allow a sequence of S values $S_0 = S, S_1, \ldots, S_j \ldots, S_{J-1}$ satisfying $\prod_{j\in[J]-1} S_j = D/\mathrm{ord}_n(p)$.

Let us define symbols relevant to generalized S_j values. For each S_j we have a variable $k_j \in [\log(S_j)] - 1$ analogous to k with respect to S of the previous section. We define the following as generalizations of D and T.

$$D_0 \coloneqq D, \forall j > 0,\ D_j \coloneqq D_{j-1}/S_{j-1}$$
$$T_0 \coloneqq T, \forall j > 0,\ T_j \coloneqq D_j/S_j$$

We also define for convenience $L_j \coloneqq \sum_{h=0}^{j-1} \log(S_h)$. In round j we start with $k_j = 0$ and $\prod_{h\in[j]-1} S_h$ components. As k_j increases we have $(\prod_{h\in[j]-1} S_h)2^{k_j}$ components. We will extend our notation from the previous section (using $i \in [2^{k_0}]$) to enumerate these $(\prod_{h\in[j]-1} S_h)2^{k_j}$ components as $r2^{k_j} + i$ for $r \in [D/D_j] - 1$ and $i \in [2^{k_j}] - 1$. Here r is a different variable than in the previous section, and without conflict since the two solutions for cross-thread communication are exclusive.

In the first subsection that follows we develop the new decomposition formula for application within threads, first presenting the polynomial form and then adapting it to array form. In the second subsection that follows we develop the cross-thread communication mechanism we need in order to locate coefficients within threads appropriately for our decomposition formula.

7.1.1 Decomposition for round j

In the previous section, both when decomposing within threads and across threads, we implicitly applied the same four-step process. Before doing the same here we outline the process as follows.

1. First we write the decomposition formula in polynomial form.
2. Second we specify how the arrays of the threads encode the polynomial components.
3. Third we see how splitting components into their lower and upper halves translates to splitting array entries into two halves.
4. Fourth we use steps 2 and 3 to translate the decomposition formula from step 1 into array form.

First we write the decomposition formula in polynomial form. We already derived the decomposition formula in Equation (10) of Section 6, but we will now rewrite it in our new context.

$$\begin{aligned}&\forall r \in [D/D_j] - 1,\ \forall i \in [2^{k_j}] - 1, \forall b \in \{0,1\},\ \forall s \in [S_j/2^{k_j}] - 1,\ \forall t \in [T_j] - 1 :\\ &\quad C_{L_j+k_j+1}^{(r2^{k_j+1}+(2i+b))}[sT_j + t] =\\ &\qquad C_{L_j+k_j}^{(r2^{k_j}+i)}[sT_j + t] + (-1)^b \zeta^{\epsilon((\boldsymbol{r},\boldsymbol{i}))} C_{L_j+k_j}^{(r2^{k_j}+i)}[D_j/2^{k_j+1} + sT_j + t] \qquad (18)\end{aligned}$$

Second we specify how the arrays of the threads encode the components. We enumerate the $T = T_0$ threads as $\tau \coloneqq rT_j + t$ for $r \in [D/D_j] - 1$ and $t \in [T_j] - 1$. Analogous to our process in Section 6.3.3 of decomposing with arrays, we enumerate the $D_j/2^{k_j}$ coefficients of all D/D_j components $r2^{k_j}+i$ in component-major order as

$$\left\{ (r2^{k_j}+i)(S_j/2^{k_j})T_j + sT_j + t \;\middle|\; \begin{array}{l} r \in [D/D_j]-1,\ i \in [2^{k_j}]-1 \\ s \in [S_j/2^{k_j}]-1,\ t \in [T_j]-1 \end{array} \right\}$$

We use this enumeration set to connect the components and their coefficients to arrays and their entries. When enumerating components we use $r2^{k_j}+i$ and therefore when enumerating the coefficients of components we discard $r2^{k_j}+i$ along with its scaling factor $(S_j/2^{k_j})T_j$ from the enumeration to recover $C_{L_j+k_j}^{(r2^{k_j}+i)}[sT_j+s]$. Similarly, when enumerating arrays we use rT_j+t, and therefore when enumerating array entries we discard digits r and t along with corresponding scaling factors 2^{k_j} and T_j from the enumeration to get $A_{L_j+k_j}^{(rT_j+t)}[i(S_j/2^{k_j})+s]$. We thus define our array entries as

$$\begin{gathered} \forall r \in [D/D_j]-1,\ \forall i \in [2^{k_j}]-1,\ \forall s \in [S_j/2^{k_j}]-1,\ \forall t \in [T_j]-1 : \\ A_{L_j+k_j}^{(rT_j+t)}[i(S_j/2^{k_j})+s] \coloneqq C_{L_j+k_j}^{(r2^{k_j}+i)}[sT_j+t] \end{gathered}$$

Third we see how splitting $C_{L_j+k_j}^{(r2^{k_j}+i)}$ into lower and upper halves translates to array form. To do so we note that s is the most significant digit of the coefficient enumeration, and thus it is responsible for separating the upper and lower halves. We separate s into its lower and upper halves by replacing the left enumerating set below with the equal right enumerating set.

$$\left\{ s \mid s \in [S_j/2^{k_j}]-1 \right\} = \left\{ bS_j/2^{k_j+1}+s \mid s \in [S_j/2^{k_j+1}]-1,\ b \in \{0,1\} \right\}$$

This change of enumeration yields

$$\begin{gathered} \forall r \in [D/D_j]-1,\ \forall i \in [2^{k_j}]-1,\ \forall s \in [S_j/2^{k_j+1}]-1,\ \forall b \in \{0,1\},\ \forall t \in [T_j]-1 : \\ A_{L_j+k_j}^{(rT_j+t)}[(2i+b)(S_j/2^{k_j+1})+s] = C_{L_j+k_j}^{(r2^{k_j}+i)}[bD_j/2^{k_j+1}+sT_j+t] \end{gathered}$$

Fourth we use our defining equation for arrays along with the previous equation to translate the decomposition formula Equation (18) from polynomial form to array form.

$$\begin{gathered} \forall r \in [D/D_j]-1,\ \forall i \in [2^{k_j}]-1,\ \forall b \in \{0,1\},\ \forall s \in [S_j/2^{k_j}]-1,\ \forall t \in [T_j]-1 : \\ A_{L_j+k_j+1}^{(rT_j+t)}[(2i+b)(S_j/2^{k_j+1})+s] = \\ A_{L_j+k_j}^{(rT_j+t)}[(2i)(S_j/2^{k_j+1})+s] + (-1)^b \zeta^{\epsilon((\boldsymbol{r},\boldsymbol{i}))} A_{L_j+k_j}^{(rT_j+t)}[(2i+1)(S_j/2^{k_j+1})+s] \end{gathered}$$

7.1.2 Translation for round j

So far in this section we have focused on a single round j and the formula for $\log(S_j)$ decompositions to be applied within threads. We must now examine how to arrive at the appropriate array encoding for j in the first place, in order to apply these decompositions. Specifically, suppose we have completed decompositions for $j-1$ for $j>0$ and must apply cross-thread communication to arrive at the starting configuration to apply decompositions for j with $k_j=0$. We will not discuss arriving at the starting configuration for $j = 0$ since that is the topic of reading from global memory covered in previous section. Henceforth in this subsection we assume $j > 0$.

Let us write the state of our array encoding having completed decompositions for $j - 1$. With $k_{j-1} = \log(S_{j-1})$ we have

$$\forall r \in [D/D_{j-1}] - 1,\ \forall i \in [S_{j-1}] - 1,\ \forall t \in [T_{j-1}] - 1 :$$
$$A_{L_j}^{(rT_{j-1}+t)}[i] = C_{L_j}^{(rS_{j-1}+i)}[t]$$

where we have used $L_{j-1} + \log(S_{j-1}) = L_j$. On the other hand, to begin decompositions for j we need the following state of array encodings. With $k_j = 0$ we have

$$\forall r \in [D/D_j] - 1,\ \forall s \in [S_j] - 1,\ \forall t \in [T_j] - 1 :$$
$$A_{L_j}^{(rT_j+t)}[s] = C_{L_j}^{(r)}[sT_j + t]$$

We must be careful in reading these equations, because the arrays for $j - 1$ in the first equation are not necessarily the same as the arrays for j in the second equation. The two sets of arrays share the same names with equal subscripts and superscript sets (of size T), but for $j - 1$ the arrays have length S_{j-1} while for j they have length S_j. We may assume for simplicity that they are the same arrays with length $\max\{S_{j-1}, S_j\}$.

Our goal is to match up the array entries for $j - 1$ and j, depicting how the source array values for $j - 1$ must be mapped to the destination array values for j. To do so we must equate the right sides of the equations, that is the two different enumerations of the equal set of polynomials $C_{L_j}^{(\cdot)}[\cdot]$. We make three changes of variable.

- Replace $t \in [T_{j-1}] - 1$ in the first equation with $sT_j + t$ for $s \in [S_j] - 1$ and $t \in [T_j] - 1$. Moreover, replace T_{j-1} in the first equation with S_jT_j, noting $S_jT_j = D_j = D_{j-1}/S_{j-1} = T_{j-1}$.
- Replace $r \in [D/D_j] - 1$ in the second equation with $rS_{j-1} + i$ for $r \in [D/D_{j-1}] - 1$ and $i \in [S_{j-1}] - 1$.

The result, with the array for $j - 1$ on the left and the array for j on the right, is then

$$\forall r \in [D/D_{j-1}] - 1,\ \forall i \in [S_{j-1}] - 1,\ \forall s \in [S_j] - 1,\ \forall t \in [T_j] - 1 :$$
$$A_{L_j}^{((rS_j+s)T_j+t)}[i] = C_{L_j}^{(rS_{j-1}+i)}[sT_j + t] = A_{L_j}^{((rS_{j-1}+i)T_j+t)}[s]$$

This equation presents our desired mapping. Given values for r, i, s, and t, one may determine the value of any destination array $((rS_{j-1} + i)T_j + t)$ at any index (s), by the value of the appropriate source array $((rS_j + s)T_j + t)$ at the appropriate index (i).

As the formula suggests by the symmetry of i and s, this is effectively a transpose operation. Indeed, for all $r \in [D/D_{j-1}] - 1$ and all $t \in [T_j] - 1$ one can think of each of the S_j *source* arrays (on the left) with indices $(rS_j + s)T_j + t$ as a row of width S_{j-1} of an $S_j \times S_{j-1}$ matrix. On the other hand, one may think of each of the S_{j-1} *destination* arrays (on the right) with indices $(rS_{j-1} + s)T_j + t$ as a row of width S_j of an $S_{j-1} \times S_j$ matrix. Then by the symmetry of s and i these two matrices must be transposes.

7.1.3 For implementation

We discuss implementation in the context of threads on a GPU that may communicate via memory or what we call 'shuffle' operations. Threads on a GPU execute in sets call 'warps,' with set sizes a power of two and usually 32. The threads in a warp execute in lockstep, and this gives them the ability the communicate between each other using an instruction we call 'shuffle.' On a shuffle instruction, each thread in the warp provides data along with the index for some thread in the warp, the returned result being the data submitted by that thread. This way threads in a warp can shuffle data between each other in an arbitrary pattern, all done with a single instruction. All threads, on the other hand, whether or not in the same warp, may communicate using memory, but that requires three instructions rather than one. The first instruction writes to memory, the second instruction synchronizes the threads to be sure all have finished writing, and the third instruction reads back from memory. Therefore we prefer shuffle operations whenever communicating threads are in the same warp, and we resort to memory whenever communicating threads are not in the same warp.

To implement the translation using memory, threads may write their values to memory in any desired order, and then read them back according to that order and the translation formula. We will order in component-major order, first ordering components with $rS_{j-1} + i$ and within each component ordering coefficients with $sT_j + t$. Therefore the operation may be completed as follows.

1. To write to memory, each source thread first identifies its r, s, and t values. The threads then iterate through steps $i \in [S_{j-1}]$ and at each step i, source thread $(rS_j + s)T_j + t$ writes from array index i to memory address $(rS_{j-1} + i)D_j + sT_j + t$.

2. Synchronize to ensure all threads have completed writing.

3. To read from memory, each destination thread first identifies its r, i, and t values. The threads then iterate through steps $s \in [S_j] - 1$ and at each step s, destination thread $(rS_{j-1} + i)T_j + t$ reads from memory address $(rS_{j-1} + i)D_j + sT_j + t$ into array index s.

We now move on to the case that the number of threads $T[j-1]$ fits in a warp such that we may translate from $j-1$ to j using shuffle operations. While S_{j-1} and S_j may not coincide, implementation is certainly easier when they do. It may be that one is simply distinguishing between S_{j-1} and S_j because one wishes to perform $S_{j-1} + S_j$ rather than $2S$ decompositions, and this is basically our situation. In general we would like to use a single S value, but it may not divide the number the decompositions we wish to apply. Starting with D_0 we have the option to apply d decompositions for any $d \in \{0, \log(D_0) - (\log(D_0) \bmod \log(S))\}$. For any such d there is a multiplier $m := \lceil d/\log(S) \rceil$ satisfying $d \leq m \log(S) \leq \log(D_0)$ such that we *could* (if algebraically possible) apply $m \log(S)$ decompositions with no complication, but we choose to stop short with only d decompositions. While this can prevent one from applying up to $\log(S) - 1$ decompositions one could otherwise apply using variable S_j values, it turns out for our particular purposes not described in this section, such a restriction is not a problem. So henceforth we indeed focus on a singe S value, and to apply d decomposition for appropriate d we proceed as follows.

1. We may apply $\log(S)$ decompositions within threads followed by the translation of the previous subsection. We may repeat this process $\lfloor d/\log(S) \rfloor$ times, for a total of $\log(S) \lfloor d/\log(S) \rfloor$ decompositions.

2. There remain $d \bmod \log(S)$ decompositions to perform, and these too may be performed within threads, but we only apply $d \bmod \log(S)$ decompositions this last round rather than $\log(S)$ as in the previous rounds.

While we could have written the previous formulas in this section for a single S value, working with variable S_j values provided clarity (as least for the author) as well as opportunity for more flexibility in the future in case $d \in \{0, D_0 - (D_0 \bmod \log(S))\}$ becomes too restrictive. Note that T_j, D_j, and L_j values may still differ across j.

Recall from the end of the previous subsection that for every $r \in [D/D_{j-1}] - 1$ and $t \in [T_j] - 1$ we are tasked with implementing a transposition. The matrix to be transposed has size $S_j \times S_{j-1} = S \times S$ and each row of the matrix corresponds to a source array. Since we have a single S, the number of source and destination arrays coincide, so we may assume the source arrays are in fact the destination arrays too. Thus the task of transferring values from source to destination arrays becomes a task of permuting the values among the arrays. We will use shuffle operations to implement these permutations.

By examining a square matrix and recalling that arrays correspond to rows, it is clear that every array must write to and read from every other array. If we were to iterate across rows $s \in [S] - 1$ or columns $i \in [S] - 1$ and naively perform the necessary swaps, at each step a single array would need to read all its values or write all its values. This pattern is not fit for a parallel implementation. Instead we need each array at each step to read one value and write one value. While this is sufficient, it is not optimal, since it may be that an array must read values for many coefficients before it sends its own values for those coefficients, costing either temporary storage up two twice the size of the array, or complex data movement at every step. If we can have each thread at each step read and write for the same coefficient, we can avoid these costs. Arranging the communication between threads to do so seems to imply that at any step we must have *pairs* of threads swap values (rather than larger cycles of sharing). We now develop this further into a solution.

Let us identify arrays (one-to-one with threads) by $\tau(i) := (rS + i)T_j + t$ for $i \in [S] - 1$. We rewrite our array translation equation using τ as

$$\begin{gathered} \forall i \in [S] - 1,\ \forall s \in [S] - 1 \\ A_{L_j}^{(\tau(s))}[i] = A_{L_j}^{(\tau(i))}[s] \end{gathered}$$

Each array $\tau(i)$ will proceed through steps $s \in [S - 1]$, in each step to swap a coefficient with another array. The coefficient to target, however, cannot be s for all arrays as this would mean all arrays must swap with the single thread $\tau(s)$. Instead, each thread $\tau(i)$ must swap a coefficient with a distinct thread $\gamma_s(i)$ for some $\gamma_s \colon S \to S$. Thus we rather use the formula

$$\begin{gathered} \forall i \in [S] - 1,\ \forall s \in [S] - 1 \\ i' := \gamma_s(i) \\ A_{L_j}^{(\tau(i'))}[i] = A_{L_j}^{(\tau(i))}[i'] \end{gathered}$$

and it remains to find an appropriate function γ_s. We assume array $\tau(i)$ follows the formula and swaps for coefficient $\gamma_s(i)$ when swapping with array $\tau(\gamma_s(i))$. Two properties of γ_s are necessary and sufficient for our purpose.

- We need γ_s to be a permutation. This way, array $\tau(i)$ swaps with every other array $\tau(\gamma_s(i))$ exactly once, and therefore swaps for every coefficient.

- To ensure γ_s indeed induces 'swapping' between arrays, we require that $\gamma_s \circ \gamma_s$ is the identity function. This way, for array $\tau(i)$ to swap with array $i' := \gamma_s(i)$ means for array $\tau(i')$ to swap with array $\gamma_s(i') = \gamma_s(\gamma_s(i)) = i$.

In our case where S is a power of two, such a function may be had simply as $\gamma_s(i) := i \oplus s$ where $\oplus$ returns the integer had by XORing the bit decompositions of its arguments. Indeed, this function is both a permutation and a proper swapping function as $(i \oplus s) \oplus s = i$. Our final formula is

$$\forall i \in [S] - 1,\ \forall s \in [S] - 1$$
$$A_{L_j}^{(\tau(i\oplus s))}[i] = A_{L_j}^{(\tau(i))}[i \oplus s]$$

7.2 Arrangement of components for multiplication

Thus far in this section we have discussed decomposing a ring element in $\mathbb{Z}_p[X]/\Phi_n(X)$ into components. We will now assume the components are fully decomposed, in which case there are $D/\mathrm{ord}_n(p)$ of them and each is an element in a distinct finite field of extension degree $\mathrm{ord}_n(p)$. Recall that the previous section also focused on decomposition into irreducible components, but stopped short of multiplying those irrducibles with the irreducibles of another ring element. The reason is the decomposition algorithm described in this section is heuristically faster because it was designed for minimal cross-thread communication. The two decomposition algorithms encode the irreducibles differently, and therefore to focus on the faster algorithm, we will now discuss multiplication of irreducibles assuming they are encoded according to the decomposition algorithm of this section. In neither of the two notes do we discuss multiplication of irreducibles encoded according to the algorithm of the previous section, though the algorithms we will develop in the next section may serve as guides.

Our task for the remainder of this section is multiplying a ring element in fully decomposed form (decomposed by the algorithm of this section) by another ring element also in fully decomposed form. We will assume the component coefficients of the other ring element can be loaded on demand, and while they could be loaded multiple times to reduce space complexity, we will restrict to loading them only once.

Once our ring element is fully decomposed, each component may or may not be distributed among threads depending on parameters. While multiplication can be done when a component is distributed across threads, doing so efficiently seems exceedingly complex. The reason is different depending on the multiplication algorithm, the three of which we will consider being 'schoolbook' multiplication and two variants of Karatsuba multiplication. Though schoolbook multiplication minimizes space complexity, and Karatsuba multiplication minimizes time complexity, the way components are distributed among threads partially negates these benefits. One may trade space and time complexities whether multiplying within a thread or across threads, but when doing so across threads the trades are more expensive. An optimal trade-off complexity across threads comes at the cost of extra cross-thread communication. For example, our component coefficients are currently distributed among threads such that each thread could multiply for the coefficients it holds and apply finite field reduction such that the output is no larger than the inputs. To do so would not yield optimal time complexity since it would require more finite field reductions than necessary, requiring almost twice as many associated multiplications as necessary. *Not* to do so, on the other hand, would cause the polynomials multiplied within threads to expand in size such that overlapping coefficients are located on separate threads, costing both extra space and extra cross-thread communication to resolve the overlaps. Another issue is the ternary nature of Karatsuba multiplication in the context of thread count being a power of 2; either some threads already in use must delegate to other threads to have thread count a power of 3, or schoolbook multiplication must be used for one of the three recursive invocations to balance the load.

For these reasons, we will instead perform each multiplication within a single thread and accept the accompanying consequences. The primary consequence is the restriction that a component must be small enough to fit in a single thread. Furthermore, depending on the multiplication algorithm, *both* components being multiplied may need to fit in a single thread. For our purposes this restriction is acceptable. The other consequence is that with each component distributed among multiple threads, we must rearrange the components using cross-thread communication such that each component is located in a single thread. This consequence is also acceptable because should we instead multiply components when distributed across threads, at *least* the same amount of cross-thread communication and likely more would be necessary.

We now describe the process one may perform after all components have been moved into individual threads. We will not formally describe the process step-by-step, but rather now describe how the remaining items in this section fit together to constitute the process. In the next section three multiplication algorithms will be presented. All multiplication algorithms are for the purpose of multiplying two irreducible components, and all of them express their output in power basis form. In this section we will assume thread τ allocates a second array $B^{(\tau)}$ also of size S for the purpose of storing the outputs of all component multiplications. Specifically, upon allocating array $B^{(\tau)}$, one iterates in natural order through the components in the array $A^{(\tau)}$ and invokes a multiplication algorithm on each one, directing the output to array $B^{(\tau)}$. This way, upon completing all multiplications, array $B^{(\tau)}$ encodes the results with components ordered naturally and coefficients encoded by power basis. Finally, we wish to write array $B^{(\tau)}$ to memory as final output. While the next section is devoted to the multiplication algorithms, in the first subsection below we will jump ahead to the point that $B^{(\tau)}$ contains the results and is to be written to memory. We will describe how this memory operation might best be done. In the second subsection below we will cover how we may use cross-thread communication to collect components into single threads such that multiplication algorithms can be applied to them in the first place.

7.2.1 Writing results to shared memory

Upon moving components into single threads (as described in the next subsection) and invoking multiplication on each, array $B^{(\tau)}$ will contain the results and we wish to write them to memory. As mentioned, components will be encoded in $B^{(\tau)}$ in their natural order and within each component coefficients will be encoded in power basis order. Each thread is then to write array $B^{(\tau)}$ to memory. Specifically, thread $\tau \in [T] - 1$ is to write the S values of array $B^{(\tau)}$ to the τ'th segment of S memory cells. In this subsection we discuss how this memory writing operation can be done efficiently. It may seem this task of writing to memory is simple: either iterate through the S array indices and write each array value to its proper memory index, or iterate through the S memory indices and write to each memory index from its proper array index. The flaw in these simple strategies should become apparent once we clarify what we mean by ‘memory.’

Apart from cache memory, there are two primary types of GPU memory commonly called ‘global memory’ and ‘shared memory.’ Global memory is much larger and takes much longer to access. Shared memory is smaller and faster. For our circumstance, we wish to write our result to shared memory. The reason is that, while the topic is multiplying two ring elements, in practice we wish to multiply many pairs of ring elements and then sum them together. We will use shared memory to store the intermediate products before we sum them together.

Best practices for accessing shared memory differ from those for accessing global memory. In the first note when we read input from memory, we were accessing global memory, in which case we prioritize sequential threads accessing sequential memory locations. When accessing shared memory, in contrast, we prioritize each warp of X threads accessing memory indices representing the full residue set $[X]-1$ modulo X. (X is often called the ‘execution-width.’) The reason is shared memory is divided into X memory ‘banks,’ each bank only able to process one read or write operation at a time. Therefore to avoid delay we prefer read and write instructions for which each of the X threads in a warp accesses a distinct bank. To do so we must understand how banks represent memory. Bank $x \in [X]-1$ represents all memory indices with index $x \bmod X$. GPU shared memory is designed as such to optimize for the common case that sequential threads will access sequential memory locations. Unlike global memory, however, this architecture allows us to optimize memory transactions for the more general case that threads access indices with distinct residues modulo X in *any* order, noting sequentially ordered indices are one special case.

Recall how earlier in this section we used ‘memory’ (not specifying global or shared memory) when we translated between decomposition rounds $j-1$ and j by writing to and reading back from memory. In practice we would use shared memory for this. We did not concern ourselves with optimized memory access because in practice we would only translate from $j-1$ to j when $T_j \geq X$, switching to shuffle operations for $T_j < X$. On returning to that material one may observe that the T_j threads in each group write to T_j adjacent memory addresses. Therefore, whether accessing global memory or shared memory, the access pattern is appropriate provided $T_j \geq X$.

We now describe the access pattern we use for threads to write their S values to memory. We consider two separate cases for $X < S$ and $X \geq S$.

- Supposing $X < S$, we proceed in S steps, at step $s \in [S]-1$ thread $x \in [X]-1$ reads from array index $(x+s) \bmod S$ and writes to memory index $xS + (x+s) \bmod S$. To see that for any fixed s the expression yields for each x a distinct residue modulo X, first note that

  $$xS + (x+s) \bmod S = xX(S/X) + (x+s) \bmod X(S/X) = (x+s) \bmod X(S/X)$$

  With $x \in [X]-1$ the X values $\{(x+s) \bmod X(S/X)\}$ are contiguous modulo $X(S/X)$, and must therefore be contiguous modulo X.

- Supposing $X \geq S$, we divide the X threads in a warp into S groups of size X/S. We will proceed in S steps. At step $s \in [S]-1$, each thread $\delta \in [X/S]-1$ in group $\gamma \in [S]-1$ will read from array index $(\gamma+s) \bmod S$ and write to memory index $\gamma X + \delta S + (\gamma+s) \bmod S$ where we are writing the modulo operation with higher precedence than addition. (Note the symbol γ should not be confused with its role in a previous section of this section, nor should δ be confused with its role in a subsequent section.) As s spans $[S]-1$ each thread reads from all S array indices and writes to the S adjacent memory indices beginning at index $\gamma X + \delta S$. Furthermore, for each step $s \in [S] - 1$ all X threads access a distinct memory index modulo X. To see this, note the expression modulo X reduces to $\delta S + (\gamma + s) \bmod S$, and as δ and γ span their range this expression spans the residue set $[X] - 1$ modulo X.

Upon completing all multiplications and array $B^{(\tau)}$ containing the outputs, each thread in a warp may invoke this algorithm to write final outputs to memory.

7.2.2 Collecting components into threads

In this subsection we will discover the cross-thread communication pattern one may employ to rearrange data from its current state after decomposition to its desired state of every component located within a single thread. Once all decompositions have been applied, one will have last applied some $k_j = K$ decompositions within some round $j = J$. In this state our components are distributed among threads by the following equation

$$\forall r \in [D/D_J] - 1,\ \forall i \in [2^K] - 1,\ \forall s \in [S/2^K],\ \forall t \in [T_J] - 1 :$$
$$A_{L_J+K}^{(rT_J+t)}[i(S/2^K) + s] = C_{L_J+K}^{(r2^K+i)}[sT_J + t]$$

Our desired state is essentially any state in which all components are encoded within individual threads. Before discussing our desired encoding, let us define relevant notation. First note that there are 2^K components per group of T_J threads, and with each thread holding an array of size S each component must have size $T_J S/2^K$ and thus $2^K/T_J$ components will fit in each array. Let us use the symbol $u \in [2^K/T_j] - 1$ to enumerate the components within a thread, without confusing with use of the same symbol in the previous section.

Whereas the equation above is the current encoding (having components across threads), here we are concerned with notation for the desired encoding (having components within threads). Therefore to enumerate the threads in the new encoding, using $rT_J + t$ may be problematic since the variables r and t are already tied to representing the current encoding. But cross-thread communication will not cross thread group boundaries, meaning for each group $r \in [D/D_J] - 1$ of T_J threads, together encoding 2^K components, all threads in the group will exchange data among only themselves. Therefore we may reuse the variable r in the enumeration of arrays for our desired encoding, though in place of t we'll use a free variable $t' \in [T_J] - 1$. To enumerate *all* components, we combine our new enumeration of threads with our enumeration of components within a thread to get $(rT_J + t')(2^K/T_J) + u$. We can take this moment to relate the current enumeration of components to this new enumeration. Setting $r2^K + i$ to $(rT_J + t')(2^K/T_J) + u = r2^K + t'(2^K/T_J) + u$ we can substitute for i with $t'(2^K/T_J) + u$. Let us restate the current encoding using this new enumeration

$$\forall r \in [D/D_J] - 1,\ \forall s \in [S/2^K],\ \forall t, t' \in [T_J] - 1,\ \forall u \in [2^K/T_J] - 1 :$$
$$A_{L_J+K}^{(rT_J+t)}[t'(S/T_J) + u(S/2^K) + s] = C_{L_J+K}^{((rT_J+t')(2^K/T_J)+u)}[sT_J + t]$$

There are many natural ways to encode multiple components in a single array. It would be most natural and fitting for multiplication algorithms if encoding is in component-major order, and especially convenient if the coefficients are encoded in power basis order. Therefore as a naive first attempt we consider mapping all ordered threads and their ordered array entries to all ordered components and their power-basis-ordered coefficients. That is, array entries are indexed with digits (u, s, t) as $u(T_J S/2^K) + (sT_J + t)$. Going through the motions to figure out how coefficients must be exchanged between threads to achieve this encoding, one will soon discover an asymmetry in the way two threads must exchange coefficients. In particular, the coefficient a thread sends in each step is not the same as the coefficient it receives. Consequently, there are two options, doubling usage of either space or time as follows.

- Each thread could allocate another array with size S to hold newly received values for coefficients that have yet to be sent.
- Each thread could replace values sent with values received, requiring a subsequent corrective reshuffling of coefficients, costing essentially as many steps as the cross-thread communication.

To avoid these costs, we may try as a second attempt preserving component-major order but inverting coefficient order with digits (u, t, s) as $u(T_J S/2^K) + t(S/2^K) + s$. This encoding, however, leads to a similar asymmetry. We cannot, therefore, encode in a component-major order. We must instead turn to the dual encoding of the current encoding, had by swapping values t and t' to yield encoding

$$\forall r \in [D/D_J] - 1,\ \forall s \in [S/2^K] - 1,\ \forall t, t' \in [T_J] - 1,\ \forall u \in [2^K/T_J] - 1 : \\ A_{L_J+K}^{(rT_J+t')}[t(S/T_J) + u(S/2^k) + s] = C_{L_J+K}^{((rT_J+t')(2^K/T_J)+u)}[sT_J + t] \tag{19}$$

This leads to relation

$$\begin{aligned} &\forall r \in [D/D_J] - 1,\ \forall s \in [S/2^K] - 1,\ \forall t, t' \in [T_J] - 1,\ \forall u \in [2^K/T_J] - 1 : \\ &A_{L_J+K}^{(rT_J+t)}[(t'(2^K/T_J) + u)(S/2^K) + s] \\ &= C_{L_J+K}^{((rT_J+t')(2^K/T_J)+u)}[sT_J + t] \\ &= A_{L_J+K}^{(rT_J+t')}[t(S/T_J) + u(S/2^K) + s] \end{aligned}$$

where the array on the top has the current encoding and the array on the bottom has the desired encoding. We can simplify the enumerations noting s and u appear in both array expressions only as $u(S/2^K) + s$ and can thus be replaced by $v \in [S/T_J] - 1$ yielding

$$\forall r \in [D/D_J] - 1,\ \forall t, t' \in [T_J] - 1,\ \forall v \in [S/T_J] - 1 : \\ A_{L_J+K}^{(rT_J+t)}[t'(S/T_J) + v] = A_{L_J+K}^{(rT_J+t')}[t(S/T_J) + v]$$

One could implement this data rearrangement using memory by writing all values to memory and reading back accordingly. But for efficiency we prefer shuffle operations to memory operations. Note how in each group of T_J threads, every thread must exchange S/T_J values with every other thread. Once again XOR may serve our purpose as it did in the 'translation for round j' in Section 7.1.2. Iterating in steps, for each step $t' \in [T_J] - 1$ each thread $t \in [T_J] - 1$ will exchange all appropriate S/T_J values with thread $t \oplus t' \in [T_J] - 1$. Our formula becomes

$$\forall r \in [D/D_J] - 1,\ \forall t, t' \in [T_J] - 1,\ \forall v \in [S/T_J] - 1 : \\ A_{L_J+K}^{(rT_J+t)}[(t \oplus t')(S/T_J) + v] = A_{L_J+K}^{(rT_J+t\oplus t')}[t(S/T_J) + v]$$

Upon applying this formula for rearranging all components into single threads, we may apply any of the three multiplication algorithms that follow in the next section.

7.3 Multiplying components within threads

We are multiplying the $D/\mathrm{ord}_n(p)$ irreducible components of the ring element we have decomposed by the irreducible components of another ring element that we may load incrementally on demand. We will refer to the ring element we have decomposed as the 'major' ring element, and the other ring element as the 'minor' ring element, simply for sake of convenient distinction. When we begin multiplication, components and their coefficients are distributed among threads according to encoding Equation (19). Each multiplication algorithm is responsible for adapting to this input encoding. We may rewrite the encoding simplifying $rT_J + t$ to a single variable $\tau \in [T] - 1$.

$$\forall \tau \in [T] - 1,\ \forall s \in [S/2^K] - 1,\ \forall t \in [T_J] - 1,\ \forall u \in [2^K/T_J] - 1 : \tag{20}$$

$$A^{(\tau)}_{L_J+K}[t(S/T_J) + u(S/2^K) + s] = C^{(\tau(2^K/T_J)+u)}_{L_J+K}[sT_J + t] \tag{21}$$

For each τ and u there is a major component to multiply by a minor component, and we focus on one such pair. Before we begin discussion of individual algorithms, we specify in which finite field each multiplication takes place. Every multiplication takes place in a distinct extension field of $\mathbb{Z}_p[X]$ with defining modulus $X^{\mathrm{ord}_n(p)} - \zeta^{\epsilon(i)}$, where zeta is a primitive $D/\mathrm{ord}_n(p)$'th root of unity, and $i := \tau(2^K/T_J) + u$ is the component index. Henceforth we will reference $\zeta^{\epsilon(i)}$ as the value defining the relevant finite field without re-introduction.

7.3.1 Schoolbook multiplication

We begin with the schoolbook multiplication algorithm. We will proceed in $T_J S/2^K$ steps, at step $sT_J + t$ for $s \in [S/2^K] - 1$ and $t \in [T_J] - 1$ loading coefficient $sT_J + t$ of the minor component, multiplying the corresponding monomial (of power $sT_J + t$) with all coefficients of the major component, and then adding the results to an accumulator to sum the results of all steps. As for the encoding of the accumulator, we choose the power basis, that is coefficient $sT_J + t$ of component $u \in [2^K/T_J] - 1$ will reside at index $u(T_J S/2^K) + sT_J + t$. Our accumulator will be the newly allocated array $B^{(\tau)}$ of size S, and since we use it as an accumulator it is important all entries are initialized to zero. For a *single* multiplication, the time complexity will be $\mathrm{ord}_n(p)^2$ while the space complexity will be $2\mathrm{ord}_n(p) + 1$, using $\mathrm{ord}_n(p)$ space for the major component, $\mathrm{ord}_n(p)$ space within the accumulator, and an additional unit space for the currently loaded minor coefficient. Of course the time and space complexities are $2^K/T_J$ times larger accounting for all $2^K/T_J$ components multiplied, but we still find it relevant to consider the complexity for a single multiplication.

We will now present the multiplication of component $u \in [2^K/T_J] - 1$ with output to be written to accumulator $B^{(\tau)}$ starting at index $u(T_J S/2^K)$. At step $sT_J + t$ on loading coefficient $sT_J + t$ of the minor component we must multiply its corresponding monomial by all coefficients of the major component and add results to the accumulator. To do so we will shift all major monomials up by $sT_J + t$ powers, multiply the top $sT_J + t$ major monomials by $\zeta^{\epsilon(i)}$, and rotate those top $sT_J + t$ major monomials into the bottom monomials. Finally, we multiply this new arrangement of major monomials by the minor monomial and add all resulting monomials to the accumulator.

Literally rotating the major monomials and multiplying $sT_J + t$ of them by $\zeta^{\epsilon(i)}$ at each step is a task that would require both extra space and time, and is in fact unnecessary. Instead, we will keep the major monomials in place, multiply only major coefficient $T_J S/2^K - (sT_J + t)$ (non-existent for $sT_J + t = 0$) by $\zeta^{\epsilon(i)}$, multiply all major coefficients (some already multiplied by $\zeta^{\epsilon(i)}$) by the minor coefficient $sT_J + t$, and finally add results to the accumulator accordingly. Note that once a major coefficient is multiplied in-place by $\zeta^{\epsilon(i)}$ it may remain that way for all remaining steps. Moreover, post-multiplication we abuse notation and continue to use the term 'major coefficient' for reference.

First we clarify for step $sT_J + t$ which array entry to multiply by $\zeta^{\epsilon(i)}$, and second we clarify which array entry to add to which accumulator entry. We rewrite the major coefficient to be multiplied by $\zeta^{\epsilon(i)}$ piecewise as

$$T_J S/2^K - (sT_J + t) = (S/2^K - s)T_J - t = \begin{cases} t = 0: & (S/2^K - s)T_J + 0 \\ t > 0: & (S/2^K - (s+1))T_J + (T_J - t) \end{cases}$$

Recall that the u'th component in the thread is encoded in the array with coefficient sT_J+t residing at index $t(S/T_J) + u(S/2^K) + s$. Therefore, coefficient $T_J S/2^K - (sT_J + t)$ of component u must reside at index

$$\begin{cases} t = 0: & 0 \cdot (S/T_J) + u(S/2^K) + (S/2^K - s) \\ t > 0: & (T_J - t)(S/T_J) + u(S/2^K) + (S/2^K - (s+1)) \end{cases}$$
$$= \begin{cases} t = 0: & (u+1)(S/2^K) - s \\ t > 0: & (T_J - t)(S/T_J) + (u+1)(S/2^K) - (s+1) \end{cases}$$

Thus at step $sT_J + t$ one must examine t to determine the correct index expression.

Regarding multiplication with the minor coefficient and addition with the accumulator, we iterate across the major coefficients using $t'(S/T_J) + u(S/2^K) + s'$ for all $s' \in [S/2^K] - 1$ and $t' \in [T_J] - 1$, multiplying each major coefficient with the minor coefficient. As we do so, each product must be added to the accumulator at the appropriate index, which depends on both the minor coefficient and the major coefficient. Since the accumulator is encoded in the power basis, we consider the power basis indices of the major and minor coefficients multiplied, those are indices $s'T_J + t'$ for the major coefficients $sT_J + t$ for the minor coefficient. The index of their product, again in the power basis, should be $((s'T_J + t') + (sT_J + t)) \bmod (T_J S/2^K)$. Therefore the product should be added at accumulator index $u(T_J S/2^K) + \delta(s,t,s',t')$ where $\delta(s,t,s',t')$ is defined as

$$\delta(s,t,s',t') := ((sT_J + t) + (s'T_J + t')) \bmod T_J S/2^K$$

Therefore we have our formula as

$$B^{(\tau)}[u(T_J S/2^K) + \delta(s,t,s',t')] :=$$
$$B^{(\tau)}[u(T_J S/2^K) + \delta(s,t,s',t')] + A^{(\tau)}[t'(S/T_J) + u(S/2^K) + s'] \times \text{Minor}^{(\tau(2^K/T_J)+u)}[sT_J + t]$$

Note that $B^{(\tau)}$ encodes the outputs as promised, that is with components in natural order and coefficients in power basis order. To see this, note component $u \in [2^K/T_J] - 1$ is naturally encoded beginning at index $u(T_J S/2^K)$. Regarding coefficients, when we wrote the product of minor coefficient $sT_J + t$ and major coefficient $s'T_J + t'$ (in power basis) to $B^{(\tau)}$, we wrote it at coefficient index $\delta(s,t,s',t')$ which is the proper index in power basis form.

7.3.2 Minimal time Karatsuba multiplication

Karatsuba multiplication saves on time complexity relative to schoolbook multiplication, but at the expense of space complexity. By time complexity we essentially mean the number of base field multiplications that must be performed to multiply a major and minor component. Given the major component in power-basis form, we must multiply it by a minor component, which can be loaded on demand in any basis desired. We are assuming that the components to multiply are irreducible, and let us denote the component size by $\Theta := T_J S/2^K$. For sake of comparison, suppose we could fully decompose the components down to the base field (irreducibles of degree zero), at which point we'd simply perform $\log(\Theta)$ more layers of decomposition and then perform Θ multiplications in the base field (loading the minor component in fully decomposed form). Let us calculate the number of multiplications required for this approach. Each layer of decomposition requires $\Theta/2$ multiplications, so in total the number of multiplications required is $\Theta\log(\Theta)/2 + \Theta$. Karatsuba multiplication, on the other hand, requires $\Theta^{\log(3)}$ (an integer as Θ is a power of 2) multiplications for the product polynomial of degree $2(\Theta - 1)$, followed by $\Theta - 1$ multiplications for finite field reduction. For all $\Theta > 1$ Karatsuba requires more multiplications, specifically for $\Theta = 2, 4, 8, 16$ Karatsuba requires respectively $1.33, 1.5, 1.7, 2$ times more multiplications. Unfortunately, often it is the case that our ring does not fully decompose, so we must use Karatsuba multiplication and tolerate the additional time complexity, space complexity, and implementation complexity.

There are many ways to implement Karatsuba multiplication for polynomials, choosing trade-offs between space and time. Since our schoolbook algorithm prioritizes saving space at the expense of time, with Karatsuba we will instead prioritize time at the expense of space. Depending on the relative performance of the schoolbook and Karatsuba algorithms, one may choose an algorithm appropriately on the spectrum between the two. We will next present our recursive Karatsuba algorithm, which will not involve any finite field reduction to avoid redundant reduction for the sake of minimal time complexity. We will describe finite field reduction after Karatsuba multiplication, and indeed the reduction will be applied after the multiplication.

Our Karatsuba algorithm takes as inputs two arrays of length Θ holding the coefficients of two polynomials to multiply. The output of the algorithm is the product of the polynomials with $2 \cdot \Theta - 1$ coefficients, and it will *overwrite* the input to save space. The extra space required by the algorithm is $2 \cdot \Theta - 1$. Specifically, with $\log(\Theta)$ layers of recursion, the top layer will allocate Θ space, and all other layers will collectively allocate $\Theta - 1$ space. Denote the two polynomials to multiply as P and P'.

We now describe the algorithm we call 'Karatsuba' which accepts as input two polynomials represented as coefficient lists. Throughout the algorithm we will denote a polynomial via a coefficient list denoted by an array with Python slice notation indicating the relevant entries of the array. All polynomials represented as such are associated with the power basis. We will now list the steps for invoking the algorithm as Karatsuba$(P[0:\Theta], P'[0:\Theta])$. The output is the product polynomial of size $2 \cdot \Theta - 1$ and will be be encoded by overwriting the input with the lower Θ coefficients in P and the upper $\Theta - 1$ coefficients in P'. The top coefficient of P' may not equal zero and is to be ignored.

1. If $\Theta = 1$ then set $P[0] := P[0] \cdot P'[0]$ and exit the algorithm. Note that as promised the lower $\Theta = 1$ coefficients are encoded in P while the upper $\Theta - 1 = 0$ coefficients (non-existent) are encoded in P'. If $\Theta > 1$ then skip this step and perform all others.

2. Allocate an extra array denoted Q of size Θ.

3. For both P and P' we sum the bottom and top halves and store the results in Q, the two results to be multiplied as polynomials of size $\Theta/2$ in the subsequent step.

$$\begin{gathered}\forall\theta \in [\Theta/2] - 1 : \\ Q[\theta] \coloneqq P[\theta] + P[\Theta/2 + \theta] \\ Q[\Theta/2 + \theta] \coloneqq P'[\theta] + P'[\Theta/2 + \theta]\end{gathered}$$

4. The following three recursive invocations of the algorithm are independent, though they must be executed in sequence to prevent exponential growth in space complexity.

$$\begin{gathered}\text{Karatsuba}(P[0 : \Theta/2], P'[0 : \Theta/2]) \\ \text{Karatsuba}(Q[0 : \Theta/2], Q[\Theta/2 : \Theta]) \\ \text{Karatsuba}(P[\Theta/2 : \Theta], P'[\Theta/2 : \Theta])\end{gathered}$$

   Note how the first invocation takes place across the bottom halves of P and P', the second invocation across the two halves of Q, and the third invocation across the top halves of P and P'. Suppose informally that R_0, R_1, and R_2 represent the polynomials computed by the first, second, and third invocations above. Then the output polynomial is

$$R_0(X) + X^{\Theta/2}(R_1(X) - R_0(X) - R_2(X)) + X^{\Theta}R_2(X)$$

5. Let us take this step only to clarify what remains to be computed before specifying the algorithmic steps to compute it.

   - The lower half of P must hold coefficients $(0 : \Theta/2)$ of R_0. This is in fact already the case.
   - The upper half of P must hold the sum of coefficients $(\Theta/2 : \Theta - 1)$ of R_0 and coefficients $(0 : \Theta/2)$ of $R_1 - R_0 - R_2$. Since coefficient $\Theta - 1$ of R_0 is zero, we write coefficient $\Theta - 1$ of P as

$$P[\Theta - 1] \coloneqq Q[\Theta/2 - 1] - P[\Theta/2 - 1] - P[\Theta - 1]$$

     and we write the remaining $\Theta/2 - 1$ coefficients of the upper half of P as

$$\begin{gathered}\forall\theta \in [\Theta/2 - 1] - 1 : \\ P[\Theta/2 + \theta] \coloneqq P'[\theta] + (Q[\theta] - P[\theta] - P[\Theta/2 + \theta])\end{gathered}$$

   - The lower half of P' must hold the sum of coefficients $(0 : \Theta/2)$ of R_2 and coefficients $(\Theta/2 : \Theta - 1)$ of $R_1 - R_0 - R_2$. Since coefficient $\Theta - 1$ of $R_1 - R_0 - R_2$ is zero, we write coefficient $\Theta/2 - 1$ of P' as

$$P'[\Theta/2 - 1] \coloneqq P[\Theta - 1]$$

and we write the remaining $\Theta/2 - 1$ coefficients of the lower half of P' as

$$\forall \theta \in [\Theta/2 - 1] - 1 :$$
$$P'[\theta] \coloneqq P[\Theta/2 + \theta] + (Q[\Theta/2 + \theta] - P'[\theta] - P'[\Theta/2 + \theta])$$

where we are not assuming the upper half of P was set as in the previous point because we are only writing these assignments for clarification.

- The upper half of P' must hold coefficients $(\Theta/2 : \Theta - 1)$ of R_2. This is in fact already the case.

Notice how for $\theta \in [\Theta/2 - 1] - 1$ the difference $P'[\theta] - P[\Theta/2 + \theta]$ appears in the upper half of P while the difference $P[\Theta/2 + \theta] - P'[\theta]$ appears in the lower half of P'. The remaining steps will take advantage of this symmetry to compute the desired results for the upper half of P and the lower half of P'.

6. We compute the following difference to appear in the upper half of P

$$\forall \theta \in [\Theta/2] - 1 :$$
$$Q[\theta] \coloneqq Q[\theta] - P[\theta]$$

and the corresponding difference to appear in the lower half of P'

$$\forall \theta \in [\Theta/2 - 1] - 1 :$$
$$Q[\Theta/2 + \theta] \coloneqq Q[\Theta/2 + \theta] - P'[\Theta/2 + \theta]$$

7. Here we handle the outlier coefficients $P[\Theta - 1]$ and $P'[\Theta/2 - 1]$. To do so we perform the following.

$$P'[\Theta/2 - 1] \coloneqq P[\Theta - 1]$$
$$P[\Theta - 1] \coloneqq Q[\Theta/2 - 1] - P'[\Theta/2 - 1]$$

8. For $\theta \in [\Theta/2 - 1] - 1$ we will compute one of the two differences $P'[\theta] - P[\Theta/2 + \theta]$ or $P[\Theta/2 + \theta] - P'[\theta]$. Arbitrarily choosing the latter which appears in the lower half of P' we calculate

$$\forall \theta \in [\Theta/2 - 1] - 1 :$$
$$P'[\theta] \coloneqq P[\Theta/2 + \theta] - P'[\theta]$$

9. To compute the final upper half of P we write

$$\forall \theta \in [\Theta/2 - 1] - 1 :$$
$$P[\Theta/2 + \theta] \coloneqq Q[\theta] - P'[\theta]$$

To compute the final lower half of P' we write

$$\forall \theta \in [\Theta/2 - 1] - 1 :$$
$$P'[\theta] \coloneqq Q[\Theta/2 + \theta] + P'[\theta]$$

The algorithm of Karatsuba multiplication we've layed out multiplies a major and minor component without performing finite field reduction, the task to which we now turn. Continuing with the notation P and P', upon invoking the Karatsuba algorithm with $\Theta = T_J S/2^K$, the resulting polynomial has its lower Θ coefficients in P and its upper $\Theta - 1$ coefficients in P'. Finite field reduction means multiplying coefficient θ of P' for $\theta \in [\Theta - 1] - 1$ by $\zeta^{\epsilon(\boldsymbol{i})}$ and adding to coefficient θ of P.

$$\begin{gathered}\forall\theta \in [\Theta - 1] - 1 : \\ P[\theta] \coloneqq P[\theta] + \zeta^{\epsilon(\boldsymbol{i})} \cdot P'[\theta]\end{gathered}$$

Finally let us calculate the time and space complexity of applying Karatsuba followed by finite field reduction. For space complexity, the two inputs together account for $2\cdot\Theta$ space. As mentioned previously the extra space needed is $2\cdot\Theta-1$, so total space complexity is $4\cdot\Theta-1$. Note there are no temporary variables used in algorithm. Regarding time complexity, the number of multiplications is $3^{\log(\Theta)} = \Theta^{\log(3)}$ for Karatsuba due to the $\log(\Theta)$ depth recursion and each step of recursion spawning 3 instances. Then another $\Theta - 1$ multiplications are needed for finite field reduction, yielding total $\text{Mult}(\Theta) \coloneqq \Theta^{\log(3)} + \Theta - 1$. The number of additions and subtractions for Karatsuba not accounting for the 3 recursive invocations is tabulated for $\Theta > 1$ as

- Θ additions for step 3.
- $(\Theta/2) + (\Theta/2 - 1)$ subtractions for step 6.
- 1 subtraction for step 7.
- $\Theta/2 - 1$ subtractions for step 8.
- $\Theta/2 - 1$ subtractions and $\Theta/2 - 1$ additions for step 9.

Not counting additions and subtractions due to the 3 recursive invocations, that's a total of $\Theta + \Theta/2-1 = 3\cdot\Theta/2-1$ additions and $\Theta/2+1+3(\Theta/2-1) = 2(\Theta-1)$ subtractions. Then another $\Theta - 1$ additions are needed for finite field reduction. Therefore we have recursive formulas

$$\begin{aligned}\text{Add}_{\text{Karatsuba}}(\Theta) &\coloneqq 3\cdot\Theta/2 - 1 + 3\cdot\text{Add}(\Theta/2)\\ \text{Add}_{\text{Total}}(\Theta) &\coloneqq \Theta - 1 + \text{Add}_{\text{Karatsuba}}(\Theta)\\ \text{Sub}_{\text{Total}}(\Theta) &\coloneqq 2(\Theta - 1) + 3\cdot\text{Sub}_{\text{Total}}(\Theta/2)\end{aligned}$$

Rather than resolving the recursive formulas to non-recursive forms for asymptotic analysis, we write out the number of additions and subtractions for relevant powers of 2 as follows.

Θ	1	2	4	8	16	32	64	128
Mult(Θ)	1	4	12	34	96	274	792	2314
Add(Θ)	0	3	14	51	170	543	1694	5212
Sub(Θ)	0	2	12	50	195	678	2223	7050

This algorithm assumes the input is in power basis form, and the output overwrites the input P. Therefore we must load P and P' with inputs in power basis form, and with the output contained in

P we must make P an alias for the appropriate segment of output array $B^{(\tau)}$. The current encoding after arranging components into individual threads is shown in Equation (21). In that encoding, array entries are enumerated by $t(S/T_J) + u(S/2^K) + s$, whereas in the power basis encoding, entries are enumerated by $u(T_J S/2^K) + sT_J + t$. Therefore, when we are to multiply component $u \in [2^K/T_J] - 1$ we load the component into $B^{(\tau)}$ and alias with P as follows

$$P[sT_J + t] \coloneqq B^{(\tau)}[u(T_J S/2^K) + sT_J + t] \coloneqq A^{(\tau)}[t(S/T_J) + u(S/2^K) + s]$$

Doing so prepares P for multiplication with P' (to be loaded separately). Once all multiplications have taken place, $B^{(\tau)}$ contains the outputs as promised with components naturally ordered and coefficients ordered by power basis, at which point $B^{(\tau)}$ may be written to memory as final output.

7.3.3 Minimal space Karatsuba multiplication

Since we are performing this multiplication in a finite field where the output has the same size as each of the inputs, we can save space Θ at the cost of $\Theta - 2$ extra multiplications. Again, we are considering time and space complexity here for a single multiplication. By representing the input polynomials of size Θ in bit-reversed order, upon splitting the polynomials in half and multiplying two halves of size $\Theta/2$ (as the Karatsuba algorithm does 3 times), the output is also of size $\Theta/2$ rather than $\Theta - 1$. By 'bit-reversed' order, we mean the coefficient associated with power m is not located at array index m as in the power basis encoding, but rather at index m' where m' is the number corresponding to the bit-reversed binary decomposition of m. We briefly touched on this technique when we chose not to multiply components when coefficients are distributed across threads. While this technique indeed trades time for space, the trade is less expensive here applying it within threads than it would be applying it across threads.

First we present the algorithm for multiplication assuming the polynomials are already in bit-reversed form, at the end addressing how we may arrive at this form. Calling the algorithm 'DualKaratsuba' we now list the steps for invoking the algorithm as DualKaratsuba($P[0 : \Theta], P'[0 : \Theta]$, `preserve`) where `preserve` is a boolean argument indicating if P' must be preserved or whether it can be overwritten.

1. If $\Theta = 1$ then set $P[0] \coloneqq P[0] \times P'[0]$ and exit the algorithm. Note that P' is preserved regardless. If $\Theta > 1$ then skip this step and perform all others.

2. Allocate an extra array Q of size $\Theta/2$.

3. Copy the top half of P to Q, that is

$$\forall \theta \in [\Theta/2] - 1 : \\ Q[\theta] \coloneqq P[\Theta/2 + \theta]$$

4. Recursively multiply the tops of P and P', located in Q and the top half of P', respectively. We pass `true` as the boolean argument in order to preserve the top half of P'.

$$\text{DualKaratsuba}(Q[0 : \Theta/2], P'[\Theta/2 : \Theta], \texttt{true})$$

5. For both P and P' add the bottom half to the top half. Already having multiplied their two top halves, their top halves may be overwritten with these sums.

$$\begin{gathered}\forall\theta \in [\Theta/2]-1: \\ P[\Theta/2+\theta] \coloneqq P[\Theta/2+\theta] + P[\theta] \\ P'[\Theta/2+\theta] \coloneqq P'[\Theta/2+\theta] + P'[\theta]\end{gathered}$$

6. Recursively multiply the bottom halves of P and P', and also recursively multiply the sums which are located in the top halves of P and P'. After these multiplications we have no further need for P' other than to return it in preserved form if requested. Note that in these two recursive calls the two items to potentially preserve are the top and bottom halves of P'. Therefore we request preservation in these calls if and only if preservation of P' in the current call is requested.

$$\begin{gathered}\text{DualKaratsuba}(P[0:\Theta/2], P'[0:\Theta/2], \texttt{preserve}) \\ \text{DualKaratsuba}(P[\Theta/2:\Theta], P'[\Theta/2:\Theta], \texttt{preserve})\end{gathered}$$

   Note the results are stored in the bottom and top halves of P, respectively.

7. At this point the product polynomial consists of three non-overlapping parts. With respect to bit-reversed coefficients, the bottom half of P holds coefficients $(0:\Theta/2)$, the top half of P holds coefficients $(\Theta/2:\Theta)$, and Q holds coefficients $(\Theta:3\cdot\Theta/2)$. Whereas had we applied Karatsuba with the power basis the polynomial would have size $2\cdot\Theta-1$, using the bit-reversed basis the polynomial has size $3\cdot\Theta/2$.

   Next we subtract the bottom and top of the non-reduced polynomial from the middle of the non-reduced polynomial.

$$\begin{gathered}\forall\theta \in [\Theta/2]-1: \\ P[\Theta/2+\theta] \coloneqq P[\Theta/2+\theta] - Q[\theta] \\ P[\Theta/2+\theta] \coloneqq P[\Theta/2+\theta] - P[\theta]\end{gathered}$$

8. To finally construct the output polynomial in P we apply finite field reduction. Let us consider the polynomial of size $\Theta/2$ located in Q. Finite field reduction first involves multiplying this polynomial by the degree-2 monic monomial and reducing it to another polynomial in the same basis, finally adding the result to the bottom half of P. The basis here is the bit-reversed power basis in which powers are taken of the degree-2 monic monomial. But let us consider the power basis for a moment. Upon multiplying the polynomial in Q by the monomial, all coefficients shift up by 1, and the highest coefficient is multiplied by $\zeta^{\epsilon(\boldsymbol{i})}$ and rotated down into the lowest coefficient. While this transformation is simple in the power basis, we must apply it to our bit-reversed power basis instead.

   There may be more clever representations of this transformation, but we will simply map the bit-reversed power basis to the regular power basis, perform the transformation there as just described, and map back to the bit-reversed power basis. In practice, this can be done using a lookup table. First we handle the special case of the top coefficient in Q, which in either basis will be handled the same. We simply multiply it by $\zeta^{\epsilon(\boldsymbol{i})}$ and add the result to $P[0]$, that is

$$P[0] \coloneqq P[0] + Q[\Theta/2 - 1] \cdot \zeta^{\epsilon(\boldsymbol{i})}$$

The reason this doesn't change for the bit-reversed basis is because it involves the lowest index of P with all bits 0, and the highest index of Q with all bits 1, both of which are invariant under bit reversal. As for coefficient $\theta \in [\Theta/2 - 1]$ of P, we first map it to the power basis as $\rho(\theta)$ where $\rho\colon ([\Theta/2] - 1) \to ([\Theta/2] - 1)$ is the bit-reversal function. We then determine that coefficient of Q with power basis index $\rho(\theta) - 1$ should be added to $P[\theta]$, so we access that index by mapping back to the bit-reversed basis as $Q[\rho(\rho(\theta) - 1)]$. Our formula is thus

$$\begin{gathered}\forall \theta \in [\Theta/2 - 1] : \\ P[\theta] \coloneqq P[\theta] + Q[\rho(\rho(\theta) - 1)]\end{gathered}$$

9. Lastly, if `preserve` $=$ `true` then we must invert the single operation we've performed on P', which is adding its bottom half to its top half. Therefore we now subtract the bottom from the top as

$$\begin{gathered}\forall \theta \in [\Theta/2] - 1 : \\ P'[\Theta/2 + \theta] \coloneqq P'[\Theta/2 + \theta] - P'[\theta]\end{gathered}$$

Lets calculate time and space complexity (and tabulate) after we're sure it works. do it not analytically but numerically

As we did for Karatsuba multiplication, we must make P an alias for a segment of $B^{(\tau)}$ and specify how they are loaded with input. The encoding after components have been moved into individual threads is shown by Equation (21). In that encoding, array entries are enumerated by $t(S/T_J) + u(S/2^K) + s$, whereas in the bit-reversed power basis encoding, entries are enumerated by $u(T_J S/2^K) + \rho(t)(S/2^K) + \rho(s)$. Therefore, when we are to multiply component $u \in [2^K/T_J] - 1$ we load the component into $B^{(\tau)}$ and alias with P as follows

$$P[\rho(t)(S/2^K) + \rho(s)] \coloneqq B^{(\tau)}[u(T_J S/2^K) + \rho(t)(S/2^K) + \rho(s)] \coloneqq A^{(\tau)}[t(S/T_J) + u(S/2^K) + s]$$

Once all multiplications have taken place, $B^{(\tau)}$ contains outputs with components in natural order, but coefficients are now in bit-reversed power basis order. To order coefficients by power basis we apply the following rearrangement.

$$\begin{gathered}\forall \tau \in [T] - 1,\ \forall s \in [S/2^K] - 1,\ \forall t \in [T_J] - 1,\ \forall u \in [2^K/T_J] - 1 : \\ B^{(\tau)}[u(T_J S/2^K) + sT_J + t] \coloneqq B^{(\tau)}[u(T_J S/2^K) + \rho(t)(S/2^K) + \rho(s)]\end{gathered}$$

At this point $B^{(\tau)}$ may be written to memory as final output.

References

[Con] Keith Conrad. Cyclotomic extensions. `kconrad.math.uconn.edu/blurbs/galoistheory/cyclotomic.pdf`. Accessed: date-of-access.

[Mar18] Daniel A. Marcus. Number fields. `www.math.toronto.edu/~ila/2018_Book_NumberFields.pdf`, 2018. Accessed: date-of-access.

[Mil20] James S. Milne. Algebraic number theory (v3.08), 2020. Available at www.jmilne.org/math/.

[Mil22] James S. Milne. Fields and galois theory (v5.10), 2022. Available at www.jmilne.org/math/.

[Mit] Steve Mitchell. Supplementary field theory notes. `sites.math.washington.edu/~mitchell/Algg/field.pdf`. Accessed: date-of-access.

[Mor96] Patrick J. Morandi. *Field and Galois Theory*. Springer Science & Business Media, see the section on cyclotomic extensions edition, 1996.

[Ngu] Nicholas Phat Nguyen. A note on cyclotomic integers. `arxiv.org/ftp/arxiv/papers/1706/1706.05390.pdf`. Accessed: date-of-access.

[Wei] Steven H. Weintraub. Several proofs of the irreducibility of the cyclotomic polynomials. `lehigh.edu/~shw2/c-poly/several_proofs.pdf`. Accessed: date-of-access.