一般の基数の数論変換アルゴリズムの比較

名古屋工業大学大学院　工学研究科　工学専攻　情報工学系プログラム　情報数理分野
兼高泰河 (Taiga KANETAKA) *

概要

数論変換とは素数 q に対して $\mathbb{Z}/q\mathbb{Z}$ 上で定義される高速フーリエ変換である．整数係数多項式の乗算を最も高速に計算することができる手法として知られていて，近年では耐量子計算機暗号の高速化などに用いられる．特に多項式を再帰的に 2 分割することによって変換結果を得るアルゴリズムを 2 基数の数論変換という．本講演では整数 $r \geq 1$ に対して基数が 2^r の数論変換を実装し，乗算の回数や実行時間を比較し，基数 2 の数論変換が最も高速であるということを示す．

1　導入

近年，計算機の発展やセキュリティ意識の向上によって，従来よりも複雑な暗号方式が求められている．そのなかでも代表的な暗号方式として，格子暗号がある．格子暗号の中には，現在用いられている暗号方式よりも安全と考えられている耐量子計算機暗号 [4][5][8] や，暗号化したまま演算を可能にする準同型暗号等 [1][3] があり，盛んに研究が行われている．しかし，暗号を計算するために多項式同士の乗算を多用するため，実行時間が大きくなるという欠点を抱えている．多項式同士の乗算を高速に行うために高速フーリエ変換 [2] や数論変換 [7] というアルゴリズムが用いられる．

高速フーリエ変換とは，$\mathbb{C}$ 上の多項式を再帰的に分割することで，多項式同士の乗算を計算しやすい形に変形するアルゴリズムである．$(n-1)$ 次多項式同士の積を素朴に計算する場合，係数同士の乗算は $\mathcal{O}(n^2)$ 回発生するが，2 分割の高速フーリエ変換を用いることで，$\mathcal{O}(n \log n)$ 回まで減らすことができる．特に，高速フーリエ変換の計算に必要な乗算の回数は $\frac{n}{2} \log_2 n$ 回である．さらに，4 分割の高速フーリエ変換 [9] を用いることで，乗算回数は $\frac{3n}{8} \log_2 n$ まで減少する．しかし，計算機で実装した場合は丸め誤差が発生する．

数論変換 [7] とは，$\mathbb{Z}/q\mathbb{Z}$ 上の多項式を高速フーリエ変換と同様の操作で変形するアルゴリズムである．こちらは小数を扱わないので誤差が生じることなく計算できる．また，2 分割では係数同士の乗算の回数を $\frac{n}{2} \log_2 n$ 回まで減らすことができる．しかし，4 分割の数論変換 [6] は虚数単位が使えない関係で，$\frac{5n}{4} \log_4 n = \frac{5n}{8} \log_2 n$ 回の乗算を必要とする．高速フーリエ変換と同じ回数にはならないが，疑似的な複素数を導入し並列化することで，理論的には $(\frac{3}{8} n \log_2 n + \frac{n}{2})$ 回程度の乗算と同じ程度の時間で計算することができる．

本研究では整数 $r \geq 1$ に対し，2^r 分割の数論変換の乗算回数を計算し，実際に多項式を計算する際に最適な分割数を求める．

* E-mail:clz14036@nitech.jp

2　高速フーリエ変換

2.1　離散フーリエ変換

定義 1 (離散フーリエ変換)**.** f を $\mathbb{C}$ 上の $(n-1)$ 次多項式，ω を $\mathbb{C}$ の原始 n 乗根とする．このとき，

$$\hat{f}(t) = \sum_{i=0}^{n-1} f(\omega^i) t^i$$

で定義される $\mathbb{C}$ 上の $(n-1)$ 次多項式を f の離散フーリエ変換と呼ぶ.

定義 2 (離散フーリエ逆変換)**.** $\hat{f}$ を $\mathbb{C}$ 上の $(n-1)$ 次多項式とする．このとき，

$$f(x) = \frac{1}{n} \sum_{i=0}^{n-1} \hat{f}(\omega^{-i}) x^i$$

で定義される $\mathbb{C}$ 上の $(n-1)$ 次多項式を $\hat{f}$ の離散フーリエ逆変換と呼ぶ.

注 3. 離散フーリエ逆変換は原始 n 乗根 ω^{-1} を用いた離散フーリエ変換を n で割ったものと考えることができる，

本章の離散フーリエ (逆) 変換で用いられる n, ω は特に断りのない限り固定されているものとする.

定理 4. $\mathbb{C}$ 上の $(n-1)$ 次多項式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$ の離散フーリエ変換を $\hat{f}$ とする．このとき，$\hat{f}$ の離散フーリエ逆変換は f となる.

証明は省略する.

2.2　多項式の乗算

多項式の乗算を求めるのに必要な係数同士の積の回数について述べる．n, m を正の整数，K を体として，K 上の多項式 $f(x) = \sum_{i=0}^{n} a_i x^i$ と $g(x) = \sum_{j=0}^{m} b_j x^j$ の積を求める．このとき，各係数同士の積を一つずつ計算すると，

$$f(x) \cdot g(x) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j}$$

となり，K 上の乗算の回数は $\mathcal{O}(nm)$ となる.

次に，多項式の乗算と離散フーリエ変換の関係について述べる．K 上の多項式 $f(x) = \sum_{i=0}^{n} a_i x^i$ に対して，$V(f) \coloneqq (a_0, a_1, \ldots, a_n)$，$\mathbf{a} = (a_0, a_1, \ldots, a_n) \in K^{n+1}$ に対して, $F(\mathbf{a})(x) \coloneqq \sum_{i=0}^{n} a_i x^i$ と定義する.

定理 5. $f(x) = \sum_{i=0}^{n-1} a_i x^i$, $g(x) = \sum_{j=0}^{n-1} b_j x^j$ を $\mathbb{C}$ 上の $(n-1)$ 次多項式とし，f の離散フーリエ変換を $\hat{f}$，g の離散フーリエ変換を $\hat{g}$ とする．このとき，$F(V(\hat{f}) \otimes V(\hat{g}))(x)$ は $f(x) \cdot g(x)$ を $(x^n - 1)$ で割った余りの離散フーリエ変換と等しい.

証明. $V(\hat{f}) \otimes V(\hat{g})$ の i 番目の成分を計算する．$V(\hat{f})$ の i 番目の成分を $\hat{f}_i$，$V(\hat{g})$ の i 番目の成分を $\hat{g}_i$ とする．

$$\begin{aligned}
\hat{f}_i \cdot \hat{g}_i &= \sum_{j=0}^{n-1} f_j \omega^{ij} \sum_{k=0}^{n-1} g_k \omega^{ik} \\
&= \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \omega^{i(j+k)} f_j g_k \\
&= \sum_{l=0}^{n-1} \omega^{il} \sum_{m=0}^{l} f_m g_{l-m} + \sum_{l=n}^{2n-2} \omega^{il} \sum_{m=l-n+1}^{n-1} f_m g_{l-m} \\
&= \sum_{l=0}^{n-1} \omega^{il} \sum_{m=0}^{l} f_m g_{l-m} + \sum_{l=0}^{n-2} \omega^{il} \sum_{m=l+1}^{n-1} f_m g_{l+n-m} \\
&= \sum_{l=0}^{n-1} \omega^{il} \left(\sum_{m=0}^{l} f_m g_{l-m} + \sum_{m=l+1}^{n-1} f_m g_{l+n-m} \right) \qquad (1)
\end{aligned}$$

次に，$f(x) \cdot g(x)$ を $(x^n - 1)$ で割った余りを計算する．

$$\begin{aligned}
f(x) \cdot g(x) &= \sum_{l=0}^{n-1} \sum_{m=0}^{l} f_m g_{l-m} x^l + \sum_{l=n}^{2n-2} \sum_{m=l-n+1}^{n-1} f_m g_{l-m} x^l \\
&= \sum_{l=0}^{n-1} \sum_{m=0}^{l} f_m g_{l-m} x^l + \sum_{l=0}^{n-2} \sum_{m=l+1}^{n-1} f_m g_{l+n-m} x^{l+n}
\end{aligned}$$

$(x^n - 1)$ で割って整理すると以下のようになる．

$$\sum_{l=0}^{n-1} x^l \left(\sum_{m=0}^{l} f_m g_{l-m} + \sum_{m=l+1}^{n-1} f_m g_{l+n-m} \right) \bmod (x^n - 1) \qquad (2)$$

(1)，(2) より，$F(V(\hat{f}) \otimes V(\hat{g}))(x)$ は $f(x) \cdot g(x)$ を $(x^n - 1)$ で割った余りの離散フーリエ変換と等しい． □

原始 n 乗根 ω の累乗が事前に計算されていると仮定する．成分ごとの乗算は $\mathcal{O}(n)$ だが，離散フーリエ変換を素朴に計算すると，乗算が $\mathcal{O}(n^2)$ 回発生する．次節以降で解説する高速フーリエ変換を用いることで離散フーリエ変換を $\mathcal{O}(n \log n)$ 回の乗算で計算することができ，多項式の乗算全体に必要な $\mathbb{C}$ 上の乗算も $\mathcal{O}(n \log n)$ 回となる．

2.3 基数 2 の高速フーリエ変換

離散フーリエ変換の計算を高速化する手法として高速フーリエ変換 [2] が知られている．

n を 2 の累乗，ω を $\mathbb{C}$ の原始 n 乗根とする．$\mathbb{C}$ 上の $(n-1)$ 次多項式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$ に対す

る離散フーリエ変換後の j 次の項の係数を $\hat{a}_j$ とする．$\hat{a}_j$ を 2 分割する．

$$\begin{aligned}\hat{a}_j &= \sum_{i=0}^{n-1} a_i \omega^{ij} \\ &= \sum_{i=0}^{n/2-1} a_{2i}\omega^{2ij} + \sum_{i=0}^{n/2-1} a_{2i+1}\omega^{(2i+1)j} \\ &= \underbrace{\sum_{i=0}^{n/2-1} a_{2i}(\omega^2)^{ij}}_{(n/2-1)\text{ 次多項式の離散フーリエ変換}} + \omega^j \underbrace{\sum_{i=0}^{n/2-1} a_{2i+1}(\omega^2)^{ij}}_{(n/2-1)\text{ 次多項式の離散フーリエ変換}}\end{aligned}$$

ω^2 は原始 $(n/2)$ 乗根なので，前半は a_{2i} $(0 \le i \le n/2-1)$ を i 次の係数とした $(n/2-1)$ 次多項式の離散フーリエ変換の j 次の係数，後半は a_{2i+1} $(0 \le i \le n/2-1)$ を i 次の係数とした $(n/2-1)$ 次多項式の離散フーリエ変換の j 次の係数に ω^j をかけたものとなっている．

$\omega^{n/2} = -1$ なので，$j = 0, 1, \ldots, n/2-1$ に対して，

$$\begin{cases} \hat{a}_j = \displaystyle\sum_{i=0}^{n/2-1} a_{2i}(\omega^2)^{ij} + \omega^j \sum_{i=0}^{n/2-1} a_{2i+1}(\omega^2)^{ij} \\ \hat{a}_{j+n/2} = \displaystyle\sum_{i=0}^{n/2-1} a_{2i}(\omega^2)^{ij} - \omega^j \sum_{i=0}^{n/2-1} a_{2i+1}(\omega^2)^{ij} \end{cases} \tag{3}$$

となる．

以上のことから，$(n-1)$ 次多項式の離散フーリエ変換を求めるためには，$(n/2-1)$ 次多項式の離散フーリエ変換を 2 つと $n/2$ 回の乗算を計算すればよい．0 次多項式の離散フーリエ変換は恒等写像となるので，ω の累乗が事前に計算されていると仮定すると，全体の乗算回数は $\frac{n}{2}\log_2 n$ 回の乗算で離散フーリエ変換を計算できる．つまり $\mathcal{O}(n \log n)$ 回の乗算で済むため，素朴な多項式の積や離散フーリエ変換よりも高速に計算できることがわかる．

定義 6 (基数 r の高速フーリエ変換)**.** 多項式を r 分割する操作を再帰的に繰り返して離散フーリエ変換を求める手法を基数 r の高速フーリエ変換と呼ぶ.

本節で述べたのは基数が 2 の高速フーリエ変換である．

2.4 基数 4 の高速フーリエ変換

基数 4 の高速フーリエ変換 [9] について解説する．

n を 4 の累乗，$\omega = e^{\frac{2\pi i}{n}}$ とする．$\mathbb{C}$ 上の $(n-1)$ 次多項式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$ に対する離散フーリエ変換後の j 次の項の係数を $\hat{a}_j$ とする．$\hat{a}_j$ を 4 分割する．

$$
\begin{aligned}
\hat{a}_j &= \sum_{i=0}^{n-1} a_i \omega^{ij} \\
&= \sum_{i=0}^{n/4-1} a_{4i}\omega^{4ij} + \sum_{i=0}^{n/4-1} a_{4i+1}\omega^{(4i+1)j} + \sum_{i=0}^{n/4-1} a_{4i+2}\omega^{(4i+2)j} + \sum_{i=0}^{n/4-1} a_{4i+3}\omega^{(4i+3)j} \\
&= \underbrace{\sum_{i=0}^{n/4-1} a_{4i}(\omega^4)^{ij}}_{(n/4-1)\text{ 次多項式の離散フーリエ変換}} + \omega^j \underbrace{\sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij}}_{(n/4-1)\text{ 次多項式の離散フーリエ変換}} \\
&\quad + \omega^{2j} \underbrace{\sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij}}_{(n/4-1)\text{ 次多項式の離散フーリエ変換}} + \omega^{3j} \underbrace{\sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}}_{(n/4-1)\text{ 次多項式の離散フーリエ変換}}
\end{aligned} \tag{4}
$$

ω^4 は原始 $(n/4)$ 乗根なので，4 つの $(n/4-1)$ 次多項式の離散フーリエ変換に分かれている．$\omega = e^{\frac{2\pi i}{n}}$ なので，$\omega^{n/4} = i, \omega^{n/2} = -1, \omega^{3n/4} = -i$ である．したがって，$j = 0, 1, \ldots, n/4-1$ に対して，

$$
\begin{cases}
\hat{a}_j = \displaystyle\sum_{i=0}^{n/4-1} a_{4i}(\omega^4)^{ij} + \omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij} \\
\qquad\qquad + \omega^{2j} \displaystyle\sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij} + \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij} \\
\hat{a}_{j+n/4} = \displaystyle\sum_{i=0}^{n/4-1} a_{4i}(\omega^4)^{ij} + i\omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij} \\
\qquad\qquad - \omega^{2j} \displaystyle\sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij} - i\omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij} \\
\hat{a}_{j+n/2} = \displaystyle\sum_{i=0}^{n/4-1} a_{4i}(\omega^4)^{ij} - \omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij} \\
\qquad\qquad + \omega^{2j} \displaystyle\sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij} - \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij} \\
\hat{a}_{j+3n/4} = \displaystyle\sum_{i=0}^{n/4-1} a_{4i}(\omega^4)^{ij} - i\omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij} \\
\qquad\qquad - \omega^{2j} \displaystyle\sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij} + i\omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}
\end{cases} \tag{5}
$$

となる．計算機を用いる場合，虚数単位 i は複素数の実部と虚部の入れ替えと符号反転のみで表現で

きるので，乗算は必要ない．したがって，$\hat{a}_j, \hat{a}_{j+n/4}, \hat{a}_{j+n/2}, \hat{a}_{j+3n/4}$ の計算で発生する乗算は

$$\omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij}, \quad \omega^{2j} \sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij}, \quad \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}$$

の 3 回である．

以上のことから，$(n-1)$ 次多項式の離散フーリエ変換を求めるためには，$(n/4-1)$ 次多項式の離散フーリエ変換を 4 回と $3n/4$ 回の乗算を計算すればよい．0 次多項式の離散フーリエ変換は恒等写像となるので，全体の乗算回数は $\frac{3}{4}n\log_4 n = \frac{3}{8}n\log_2 n$ 回となり，基数 2 の高速フーリエ変換から減少していることがわかる．

3 数論変換

本章では q を素数，ω を $\mathbb{Z}/q\mathbb{Z}$ の原始 n 乗根とする．

3.1 基数 2 の数論変換

定義 7 (数論変換)**.** f を $\mathbb{Z}/q\mathbb{Z}$ 上の $(n-1)$ 次多項式とする．このとき，

$$\hat{f}(t) = \sum_{i=0}^{n-1} f(\omega^i)t^i$$

で定義される $\mathbb{Z}/q\mathbb{Z}$ 上の $(n-1)$ 次多項式を f の数論変換と呼ぶ.

数論逆変換も離散フーリエ逆変換と同様に定義でき，多項式の乗算も同じ乗算回数で実現できる．また，数論変換も高速フーリエ変換と同様のアルゴリズムで高速化することができる．

n を 2 の累乗とする．$\mathbb{Z}/q\mathbb{Z}$ 上の $(n-1)$ 次多項式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$ に対する数論変換後の j 次の項の係数を $\hat{a}_j$ とする．$\hat{a}_j$ を離散フーリエ変換と同様に 2 分割すると，$\omega^{n/2} = -1$ なので，$j = 0, 1, \ldots, n/2-1$ に対して，(3) となる．したがって，基数 2 の高速フーリエ変換と同様，$\mathbb{Z}/q\mathbb{Z}$ 上の乗算が $\frac{n}{2}\log_2 n$ 回発生する．

定義 8 (基数 r の数論変換)**.** 多項式を r 分割する操作を再帰的に繰り返して数論変換を求める手法を基数 r の数論変換と呼ぶ.

本節で述べたのは基数 2 の数論変換である．

3.2 基数が 4 の数論変換

基数 4 の数論変換 [6] について解説する．

n を 4 の累乗とする．$\mathbb{C}$ 上の $(n-1)$ 次多項式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$ に対する数論変換後の j 次の項の係数を $\hat{a}_j$ とする．$\hat{a}_j$ を 4 分割すると，数論変換と同様 (4) となり，ω^4 は原始 $(n/4)$ 乗根なので，4 つの $(n/4-1)$ 次多項式の数論変換に分かれている．$\omega^{n/2} = -1$ なので，$j = 0, 1, \ldots, n/4-1$ に対して，(5) の i を $\omega^{n/4}$ に置き換えた式を得る．基数 4 の高速フーリエ変換とは異なり，虚数単位

を用いた乗算の省略ができない．したがって，$\hat{a}_j, \hat{a}_{j+n/4}, \hat{a}_{j+n/2}, \hat{a}_{j+3n/4}$ の計算で発生する乗算は，高速フーリエ変換と同じ

$$\omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij}, \quad \omega^{2j} \sum_{i=0}^{n/4-1} a_{4i+2}(\omega^4)^{ij}, \quad \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}$$

の 3 回と，

$$\omega^{n/4} \cdot \omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij}, \quad \omega^{n/4} \cdot \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}$$

の 2 回を合わせた 5 回である．

以上のことから，$(n-1)$ 次多項式の数論変換を求めるためには，$(n/4-1)$ 次多項式の数論変換を 4 回と $5n/4$ 回の $\mathbb{Z}/q\mathbb{Z}$ 上の乗算を計算すればよい．全体の乗算回数は $\frac{5}{4}n\log_4 n = \frac{5}{8}n\log_2 n$ 回の乗算となり，基数 2 の数論変換と比較すると増加していることがわかる．

次に，複素数のように，$\omega^{n/4}$ の乗算を省略できる形で値を保持する方法を考える．複素数は実部と虚部で分けて値を保持しているため，虚数単位との乗算は実部と虚部の入れ替えと符号の反転のみで対応することができる．そこで，数論変換の計算過程では，以下で定義する疑似複素数形式で値を保持しておき，最後に $\mathbb{Z}/q\mathbb{Z}$ に変換するという操作を考える．

定義 9 (疑似複素数)**.** $a \in \mathbb{Z}/q\mathbb{Z}$ に対して，

$$b + c \cdot \omega^{n/4} = a$$

をみたす $(b, c) \in (\mathbb{Z}/q\mathbb{Z})^2$ を a の疑似複素数と呼ぶ．

$\omega^{n/4}$ のスカラー倍に関しては複素数における i 倍のように扱うことができるため，

$$\omega^{n/4}(b, c) = (-c, b)$$

と計算でき，乗算を省略することができる．これにより，

$$\omega^{n/4} \cdot \omega^j \sum_{i=0}^{n/4-1} a_{4i+1}(\omega^4)^{ij}, \quad \omega^{n/4} \cdot \omega^{3j} \sum_{i=0}^{n/4-1} a_{4i+3}(\omega^4)^{ij}$$

で発生した 2 回の $\omega^{n/4}$ 倍を乗算なしで計算できる．

以上を踏まえると，疑似複素数を用いた基数 4 の数論変換で発生するスカラー倍は，基数 4 の高速フーリエ変換の乗算回数と等しく，$\frac{3}{8}n\log_2 n$ 回となる．しかし，スカラー倍演算 1 回で $\mathbb{Z}/q\mathbb{Z}$ 上の乗算が 2 回発生するため，$\mathbb{Z}/q\mathbb{Z}$ 上の乗算は $\frac{3}{4}n\log_2 n$ 回となり，基数 $2, 4$ の数論変換よりも演算回数が大きくなるということになる．

また，疑似複素数の数論変換は係数が疑似複素数のままなので，$\mathbb{Z}/q\mathbb{Z}$ に変換するためには係数ごとに 1 回の $\mathbb{Z}/q\mathbb{Z}$ 上の乗算をする必要がある．したがって，全体では $(\frac{3}{4}n\log_2 n + n)$ 回の乗算を必要とする．

しかし，疑似複素数を用いた数論変換は並列性に優れていると考えられる．疑似複素数の演算は成分ごとに計算することになるため，同時に計算しても干渉することがない．また，スカラー倍は同じ

値を各成分に掛けるため，それぞれ異なる値を掛けるよりもメモリ効率が良いと考えられる．また，最後の $\mathbb{Z}/q\mathbb{Z}$ に戻す過程で発生する乗算も，掛ける値が $\omega^{n/4}$ と一定で，各乗算に依存性がないので，スカラー倍と同様に並列性に優れていると考えることができる．したがって，2 成分のスカラー倍を並列に処理できる環境では，理想的には $(\frac{3}{8}n\log_2 n + \frac{n}{2})$ 回の乗算と同等の時間で計算できるということになる．

4　主結果

前章の疑似複素数を用いた基数が 4 の数論変換と同様の手法で，整数 $r \geq 1$ に対して，基数が 2^r の数論変換を実装する．

定義 10 (拡張した疑似複素数)．$a \in \mathbb{Z}/q\mathbb{Z}$ に対して，

$$b_0 + \sum_{i=1}^{2^{r-1}-1} b_i \cdot (\omega^{n/2^r})^i = a$$

をみたす $(b_0, b_1, \ldots, b_{2^{r-1}-1}) \in (\mathbb{Z}/q\mathbb{Z})^{(2^{r-1}-1)}$ を a の基数 2^r 用に拡張された疑似複素数と呼ぶ．

基数 2^r 用に拡張された疑似複素数を値として保持する基数が 2^r の数論変換を考える．．n を 2^r の累乗とする．

$$\begin{aligned}
\hat{a}_j &= \sum_{i=0}^{n-1} a_i \omega^{ij} \\
&= \sum_{k=0}^{2^r-1} \left(\sum_{i=0}^{n/2^r-1} a_{2^r i+k} \omega^{(2^r i+k)j} \right) \\
&= \underbrace{\sum_{i=0}^{n/2^r-1} a_{2^r i} (\omega^{2^r})^{ij}}_{(n/2^r-1) \text{ 次多項式の数論変換}} + \sum_{k=1}^{2^r-1} \left(\omega^{kj} \underbrace{\sum_{i=0}^{n/2^r-1} a_{2^r i+1} (\omega^{2^r})^{ij}}_{(n/2^r-1) \text{ 次多項式の数論変換}} \right)
\end{aligned}$$

この式で乗算が発生するのは後半の ω^{kj} の乗算部分である．基数 4 の数論変換と同様に，$j = 0, 1, \ldots, n/2^r - 1$ に対して $\hat{a}_j, \hat{a}_{j+n/2^r}, \ldots, \hat{a}_{j+(2^r-1)n/2^r}$ を考えると，

$$\omega^j \sum_{i=0}^{n/2^r-1} a_{2^r i+1} (\omega^{2^r})^{ij}, \quad \omega^{2j} \sum_{i=0}^{n/2^r-1} a_{2^r i+2} (\omega^{2^r})^{ij}, \ldots, \quad \omega^{(2^r-1)j} \sum_{i=0}^{n/2^r-1} a_{2^r i+2^r-1} (\omega^{2^r})^{ij}$$

の $(2^r - 1)$ 回の乗算と，

$$\omega^{n/2^r} \cdot \omega^j \sum_{i=0}^{n/2^r-1} a_{2^r i+1} (\omega^{2^r})^{ij}$$

のような，$\omega^{n/2^r}$ の累乗の乗算が複数回必要となるが，後者の乗算は拡張された複素数によって，成分のシフトと符号の反転のみで実現できる．乗算回数を数えると，数論変換での疑似複素数のスカ

ラー倍が $\frac{(2^r-1)n}{2^r}\log_{2^r} n = \frac{(2^r-1)n}{r\cdot 2^r}\log_2 n$ 回ということになる．したがって，$\mathbb{Z}/q\mathbb{Z}$ 上の乗算の回数は $\frac{2^{r-1}(2^r-1)n}{r\cdot 2^r}\log_2 n$ 回である．また，基数 2^r 用に拡張した疑似複素数を $\mathbb{Z}/q\mathbb{Z}$ に戻す際の $\mathbb{Z}/q\mathbb{Z}$ 上の乗算の回数は $n(2^{r-1}-1)$ 回となり，全体での乗算回数は $\frac{2^{r-1}(2^r-1)n}{r\cdot 2^r}\log_2 n + n(2^{r-1}-1)$ である．これは r に関して単調に増加するため，基数が 2 の数論変換が乗算の回数において最適であるとわかる．ただし，スカラー倍の乗算を全て並列に処理できる環境では $\frac{1}{2^{r-1}}$ 倍の時間で数論変換を求めることができる．このとき，n と r の値次第では乗算にかかる時間を減らすことができる．

参考文献

[1] Chillotti, Ilaria; Gama, Nicolas; Georgieva, Mariya; Izabachène, Malika. TFHE: fast fully homomorphic encryption over the torus. J. Cryptology 33 (2020), no. 1, 34–91.

[2] Cooley, James W.; Tukey, John W. An algorithm for the machine calculation of complex Fourier series. Math. Comp. 19 (1965), 297–301.

[3] CRYPTREC 暗号技術調査ワーキンググループ（高機能暗号），CRYPTREC 暗号技術ガイドライン（高機能暗号），2023 年 3 月．`https://www.cryptrec.go.jp/report/cryptrec-gl-2005-2022.pdf`

[4] CRYPTREC 暗号技術調査ワーキンググループ（耐量子計算機暗号），CRYPTREC 暗号技術ガイドライン（耐量子計算機暗号），2023 年 3 月．`https://www.cryptrec.go.jp/report/cryptrec-gl-2004-2022.pdf`

[5] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. NIST PQC, October 2020. `https://falcon-sign.info/falcon.pdf`.

[6] S. Kudo, Y. Nogami, S. Huda and Y. Kodera. Optimizing CRYSTALS-Dilithium in Rust: Radix-4 NTT and Assembly-level Comparison with Official C Implementation. 2024 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan), Taichung, Taiwan, 2024, 395-396, September 2024.

[7] Zhichuang Liang and Yunlei Zhao. Number Theoretic Transform and Its Applications in Lattice-based Cryptosystems: A Survey, arXiv:2211.13546, November 2022.

[8] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203, August 2024.

[9] Xin Xiao, Erdal Oruklu and Jafar Saniie. Fast memory addressing scheme for radix-4 FFT implementation. 2009 IEEE International Conference on Electro/Information Technology, 437-440 August 2009.