RNS型 近似完全準同型暗号の並列処理による高速化: 1ms以下のブートストラッピングの実現

秋元 義壮[1,†1,a)]　佐久間 淳[2,†1,b)]

概要：完全準同型暗号は, 暗号文のまま平文に対して任意の回数の加算や乗算を行うことが可能な技術である. そのため, プライバシーを保護した統計解析や機械学習への応用が期待されている. しかし, 完全準同型暗号はメモリを多く消費する, 処理に時間がかかるという欠点がある. 特に, 評価する回路の乗算の深さが深い場合にはブートストラッピングと呼ばれる処理が必要となるが, これが計算時間のボトルネックとなっている. HEAAN は, 暗号化や乗算などの準同型演算で生じる誤差を, 近似計算で生じた誤差の一部であると考えることで, 暗号文上で近似算術を行うことができるスキームである. FullRNS-HEAAN は, 値を Residue Number System (RNS) で表現することによって多倍精度での処理を避け, 64bit 整数における処理のみを用いることで, HEAAN よりも処理を高速にしたスキームであるが, このスキームではブートストラッピングが実装されていなかった. 本稿では FullRNS-HEAAN 上でブートストラッピングを実現する方法を考案し, これを実装し, 実行時間に関して HEAAN におけるブートストラッピングとの比較を行った. FullRNS-HEAAN においてブートストラッピングを実現するメリットの一つに, FullRNS-HEAAN と並列計算の親和性の高さがある. 本稿では FullRNS-HEAAN における剰余多項式環上の演算の独立性と RNS 表現上の演算の独立性に着目し, 処理を複数のスレッドに分割することで処理の高速化を行った. 結果として, $\mathbb{C}^{32768}$ をパッキングした暗号文に対するブートストラッピングの実行時間は 32 スレッド並列で 30 秒ほどになり, 要素 1 つあたりの実行時間を 0.92ms にすることができた.

キーワード：完全準同型暗号, ブートストラッピング

Speed-up of RNS type approximate homomorphic encryption by parallel processing : Bootstrapping in less than 1ms.

Yoshimasa Akimoto[1,†1,a)]　Jun Sakuma[2,†1,b)]

Abstract: Fully homomorphic encryption is a technology that cnables addition and multiplication on encrypted data without decryption.Therefore, application to privacy-preserving statistical analysis and machine learning is expected. However, full homomorphic encryption requires a lot of memory and takes time to process. In particular, when the depth of multiplication of the circuit is deep, a process called bootstrapping is required, but this process is a bottleneck of calculation time. HEAAN is a fully homomorphic encryption scheme that can perform approximate arithmetic on ciphertext by allowing some error in decryption. FullRNS-HEAAN is a variant scheme of HEAAN in which processing is performed faster than HEAAN by avoiding processing with multiple precision by expressing values using Residence Number System (RNS) and using only processing in 64 bits. However, bootstrapping was not implemented in this scheme. In this paper, we designed and implemented bootstrapping in FullRNS-HEAAN. We compared this implementation with bootstrapping in HEAAN for execution time. One of the benefits of implementing bootstrapping in FullRNS-HEAAN is its high affinity to parallel computing. In this paper, focusing on independence of bootstrapping process in FullRNS-HEAAN, we speed up the process by dividing the process into multiple threads. As a result, the execution time of bootstrapping for ciphertext packed with $\mathbb{C}^{32768}$ was about 30 seconds in 32 threads, and the execution time per element was 0.92 ms.

Keywords: Fully homomorphic encryption, Bootstrapping

1. はじめに

1.1 背景と目標

完全準同型暗号は, 秘密鍵を用いることなく, 平文に対する任意の回数の加算や乗算などの演算を暗号文上で行うことができる暗号技術であり, プライバシーを保護した統計解析や機械学習への応用が期待されている. 完全準同型暗号の暗号文には乗算が可能な回数が設定されており, その回数を超えて乗算を行うと, 乗算で増加した暗号文ノイズにより正しく復号することができなくなる. そのため, 任意の回数の乗算を行うには, 暗号文上で復号回路を準同型演算を用いて評価することによってノイズを除去するブートストラッピングという処理が必要となる. 機械学習モデルの訓練などの多数の乗算が必要となる処理へ完全準同型暗号を応用することを考えるとブートストラッピングが必要となるが, 既存の準同型暗号においては, ブートストラッピングは最も計算時間がかかる処理である. したがって, ブートストラッピングを高速化することは機械学習への応用において重要な課題である. ブートストラッピングの高速化を行う際の戦略としては,

- ブートストラッピングのアルゴリズム自体を改善する
- 複数の値を 1 つの暗号文に埋め込むことによって, SIMD 命令のように 1 回の命令で複数の値に対して同様の処理を並列に行うことで, 単位要素当たりの実行時間を減らす
- ブートストラッピングの処理で使用する加算や乗算などの基本的な演算を高速化する

などが考えられる. そこで, 本研究ではこれらの戦略に基づき, 暗号文にパッキングされている単位要素当たりのブートストラッピングの実行時間を 1ms 以下にすることを目標とする.

1.2 関連研究

完全準同型暗号は, Gentry らによる発案 [1] 以降, 様々な研究 [2,3,6,7,9,10] によってメモリ消費や計算時間が効率化されてきた.

Gentry らが考案したブートストラッピングは, 以下のような処理を行う. まず, 復号鍵 sk_A で復号が可能なメッセージ m の暗号文を $\mathsf{ct}_{\mathsf{sk}_A}(m)$ とし, もう一つ別の復号鍵を sk_B とする. 次に, sk_B で $\mathsf{sk}_A, \mathsf{ct}_{\mathsf{sk}_A}(m)$ を暗号化し, $\mathsf{ct}_{\mathsf{sk}_B}(\mathsf{sk}_A), \mathsf{ct}_{\mathsf{sk}_B}(\mathsf{ct}_{\mathsf{sk}_A}(m))$ を得る. 最後に, 得られた 2 つの暗号文を用いて, 低次元の多項式で表されるような復号回路を準同型的に評価することによって $\mathsf{ct}_{\mathsf{sk}_B}(m)$ を得る. 以上の処理により, メッセージ m は sk_B によって再暗号化されたと考えられ, 暗号文上で乗算を行うことで増加したノイズの影響をリセットできる. これにより, 任意の回数の乗算を行うことができるようになるが, 最初に考案されたブートストラッピングは 80bit セキュリティを保つようなパラメータ設定において, 1bit のメッセージの暗号文に対して 30min ほどの時間が掛かり実用的ではなかった.

以降では特筆しない限り 80bit セキュリティを保つようなパラメータ設定を考える.

BGV スキーム [3] は, 平文として整数を扱うことができ, 値のパッキングが可能な完全準同型暗号スキームである. ここで, パッキングとは複数個の値を 1 つの暗号文に埋め込む手法であり, 多数の値がパッキングされている暗号文に対して準同型演算を行えば, 要素ごとに独立に準同型演算が適用される. パッキングを用いることによって, ベクトルや行列での演算などにおいて, 複数の値に同様の処理を行うような場合には単位要素当たりに必要な計算コストを削減することができる. そのため, BGV のブートストラッピング [4] では, 単位要素当たりのブートストラッピングの実行時間で約 312ms を達成している.

また, パッキングが可能な別のスキームとして HEAAN スキーム [9] がある. HEAAN は, 平文として複素数を扱うことができ, 暗号文上での近似算術をサポートしている. HEAAN に対応したブートストラッピングとして, 暗号文を復号する際に行う剰余演算を, スケーリングされた sin 関数によって近似するという HEAAN 特有の方法が考案されている [11]. さらに, [12,13] では, そのブートストラッピングに必要となる平文の行列と暗号文のベクトルの乗算を, 高速フーリエ変換を用いることによって高速化し, 単位要素あたりのブートストラッピングの実行時間で 3.8ms を達成している. このブートストラッピングで使用されているアルゴリズムは, 現在, 単位要素当たりの実行時間で考えると最も高速なアルゴリズムであると考えられる.

また, 整数を互いに素であるような複数の法に対する剰余の集合として表現する Residue Number System (RNS) 表現を BGV や HEAAN に導入することで, スキーム自体の処理を高速化する研究がなされいる [5,14]. FullRNS-HEAAN スキーム [14] は, RNS 表現を導入することによって, HEAAN でボトルネックとなっていた多倍精度での演算を避け, 全ての処理を 64bit 整数上で行うことで高速化を行ったスキームである. ブートストラッピングの実行時間の大部分を占めている乗算やベクトルの要素のシフト操作が HEAAN [9] に比べて高速化されているため, ブートストラッピングの高速化が期待できるが, [14] で提案された FullRNS-HEAAN では, ブートストラッピングがサポートされていない.

TFHE スキーム [10] は, ブートストラッピングを高速に行うことができるスキームとして知られており, 1 つの暗号文に対するブートストラッピングの実行時間は 100ms 以下となる. しかし, TFHE は平文として 0, 1 の 2 値しか扱うことができないため, 機械学習等への応用も難しく, パッキングも実装されていない.

1.3 貢献

本稿の貢献は以下の 3 つである.

[1] 筑波大学情報学群
School of Informatics, University of Tsukuba
[2] 筑波大学システム情報系
Faculty of Engineering, Information and Systems University of Tsukuba
[†1] 現在，理化学研究所革新知能統合研究センター
Presently with RIKEN Center for Advanced Intelligence Project (AIP)
a) yoshimasa@mdl.cs.tsukuba.ac.jp
b) jun@cs.tsukuba.ac.jp

- HEAAN 上で実装されているブートストラッピング [15] を FullRNS-HEAAN 上で動作させる方法を考案した. これにより, 現在, 単位要素当たりのブートストラッピングの実行時間が最も短いアルゴリズム [11,12] を RNS 表現上で使用することが可能になったため, ブートストラッピングは多倍精度の演算を用いることなく, すべての処理を 64bit 整数上で行うことができ, [11,12] にくらべてさらなる高速化が期待できる.
- FullRNS-HEAAN で用いられている RNS 表現では, RNS 表現された値同士の演算を, それぞれの法ごとに独立して行うことができる. そのため, HEAAN と比較して, FullRNS-HEAAN は並列化と親和性が高いが, [16] では並列処理の実装がなされていなかった. FullRNS-HEAAN における並列化戦略として, RNS 表現における法ごと並列化と, 多項式演算における係数ごとの並列化が可能であるが, 処理ごとに最適な並列化戦略は異なる. そこで, 本論文では複数の並列化手法を検討し, 実験評価によって最適な並列化戦略を考察した.
- FullRNS-HEAAN 上でのブートストラッピングと 3 種類の並列化手法を実装し, 実験評価を行った. 結果として, HEAAN から FullRNS-HEAAN への変更によって, ブートストラッピングは同等の精度で約 2.2 倍の高速化が達成され, 最適な並列化手法を用いることにより, 要素 1 つあたりのブートストラッピングの実行時間を 0.92ms とすることができた.

2. 準備

2.1 記法の説明

本稿では, 全ベクトルを列ベクトルと考え $\boldsymbol{a}$ のように太字で表現する. $\langle \cdot, \cdot \rangle$ は 2 つのベクトルの内積を表す. ある実数 r に対して, $\lfloor r \rceil$ は r に最も近い整数を表し, $|r|$ は r の絶対値を表す. ある整数 q に対して, $\mathbb{Z} \cap (-q/2, q/2]$ を $\mathbb{Z}_q$ と表し, $[a]_q$ は法を q とする $\mathbb{Z}_q$ の範囲内の値に $a \in \mathbb{Z}$ を変換することを表す. $x \leftarrow D$ は分布 D から x をサンプリングすることを表し, $U(S)$ は S が有限集合である場合に S 上の一様分布を表す. 全ての要素が互いに素であるような整数の有限順序付き集合 $\mathcal{B} = \{p_0, p_1, ..., p_{k-1}\}$ を基底と呼ぶ.

2.2 近似準同型暗号

HEAAN は, 暗号化や乗算などの準同型演算で生じる誤差を, 近似計算で生じた誤差の一部であると考えることで, 暗号文上で近似算術を行うことができるスキームである.

2 のべき乗数 N に対して, $\mathcal{K} = \mathbb{Q}[X]/(X^N + 1)$ を 2N 番目の円分体とし, $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ を整数環とする. $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ をある整数 q による剰余環とする. HEAAN スキームは, 固定された基底整数 q と $Q_\ell = q^\ell (1 \leq \ell \leq L)$ という法の鎖によって構成される. ここで, ℓ はレベルと呼ばれる値で, Q_ℓ を法とするような暗号文に対して残り何回乗算を行うことができるかを表しており, L はレベルの最大値を表している.

ある多項式 $m(x)$ に対して, ct がレベル ℓ における $m(x)$ の暗号文であるとき, $\mathsf{ct} \in \mathcal{R}_{Q_\ell}^2$ であり, sk を秘密鍵とすると $[\langle \mathsf{ct}, \mathsf{sk} \rangle]_{Q_\ell} \approx m(x)$ のように復号することができる. ここで, $\approx$ は復号される多項式にあらかじめ定めた上界を持つ誤差が含まれていることを示している. ある多項式 $m_1(x), m_2(x)$ に対するレベル ℓ の暗号文が与えられたとき, 2 つの暗号文の加算と乗算の結果 $\mathsf{ct}_{add}, \mathsf{ct}_{mult}$ は $[\langle \mathsf{ct}_{add}, \mathsf{sk} \rangle]_{Q_\ell} \approx m_1(x) + m_2(x), [\langle \mathsf{ct}_{mult}, \mathsf{sk} \rangle]_{Q_\ell} \approx m_1(x) \cdot m_2(x)$ を満たす.

HEAAN では暗号化に使用するノイズを, 値を実数から整数へ近似する際やそれらの近似値を用いて計算を行った際に発生する誤差の一部として扱うことで, 以前のスキームでは評価する回路の乗算の深さの増加に伴い必要な暗号文の法の大きさが指数的に増加していたのに対し, HEAAN では増加を線形に抑えることができる. その理由は以下のとおりである. HEAAN では実数を扱うために, ある実数 $r \in \mathbb{R}$ に精度パラメータ q という大きな整数を乗算し, 整数へ丸め演算を行った $z = \lfloor q \cdot r \rceil$ を暗号化する. このとき, $q \cdot r_1, q \cdot r_2$ の暗号文の積は, $q^2 \cdot r_1 r_2$ の暗号文となる. そこで, 積の結果において精度パラメータを q に保つために, 結果に q^{-1} を乗ずる Rescale 処理を行う. 暗号文のノイズに対しても同時に q^{-1} が乗算されるため, 準同型乗算後に Rescale を行うことで, 演算の前後で平文の大きさを保つことができる. そのため, HEAAN における計算の性質上, 準同型乗算によるレベルの減少は起こらないが, Rescale 処理によりレベルが 1 だけ減少する. よって, 回路を評価するために必要な暗号文の法の大きさは回路の乗算の深さに線形に増加する.

2.3 RNS 表現における HEAAN の実現

HEAAN では, 精度パラメータという非常に大きな整数を掛け, 実数を整数に近似して暗号化してるため, 暗号文同士の演算では多倍長整数を用いて計算する必要があり, これが処理のボトルネックとなっていた. このボトルネックを解消するために, HEAAN に Residue Number System (RNS) 表現を導入することで高速化を行った FullRNS-HEAAN [14] が提案されている. RNS 表現では, 整数をある基底に対する剰余の集合として表すため, 基底の要素である法の値を 64bit 整数に収まるように設定することによって, 多倍長整数を 64bit 整数のみを用いて表すことができる. そのため, 多倍精度での演算を避け, 64bit 整数上での処理のみを用いて演算が可能となり高速化される.

HEAAN では, Rescale 処理のような暗号文の法の増減を行うために、基底を構成する法として q のみを用いており, 複数の互いに素な法を用いる必要がある RNS 表現をそのまま適用することはできなかった. そこで, FullRNS-HEAAN では, $q_i \approx q, (1 \leq i \leq L)$ で $q_i, q_j (i \neq j)$ は互いに素であるような基底 $\{q_i\}_{(1 \leq i \leq L)}$ を用いることに加えて, ある基底での RNS 表現を別の基底の RNS 表現に変換する Fast Basis Conversion を用いて暗号文の法の増減を近似的に行うことで RNS 表現を適用可能にしている.

2.3.1 RNS 表現

$\mathcal{B} = \{p_0, ..., p_{k-1}\}$ を基底とし, $P = \prod_{i=0}^{k-1} p_i$ とする. また, $[\cdot]_{\mathcal{B}}$ を $a \mapsto [a]_{\mathcal{B}} = ([a]_{p_i})_{0 \leq i < k}$ で定義されるような $\mathbb{Z}_P$ から $\prod_{i=0}^{k-1} \mathbb{Z}_{p_i}$ への写像であるとする. この時, $[a]_{\mathcal{B}}$ を

$a \in \mathbb{Z}_P$ の RNS 表現と呼ぶ. また, 多項式 $a(x) \in \mathcal{R}_P$ に対して, 係数ごとにこの表現を適用すれば多項式に対しても自然に $[\cdot]_{\mathcal{B}} : \mathcal{R}_P \to \mathcal{R}_{p_0} \times \cdots \times \mathcal{R}_{p_{k-1}}$ と拡張でき, $\mathcal{R}_P$ に対する RNS 表現が得られる.

2.3.2 Fast Basis Conversion

$\{p_0, ..., p_{k-1}, q_0, ..., q_{\ell-1}\}$ を基底とし, $\mathcal{B} = \{p_0, ..., p_{k-1}\}$ と $\mathcal{C} = \{q_0, ..., q_{\ell-1}\}$ をその部分基底とする. また, $P = \prod_{i=0}^{k-1} p_i$, $Q = \prod_{j=0}^{\ell-1} q_j$ とする. この時, ある整数 $a \in \mathbb{Z}_Q$ の基底 $\mathcal{C}$ における RNS 表現 $[a]_{\mathcal{C}}$ から基底 $\mathcal{B}$ における RNS 表現 $[a]_{\mathcal{B}}$ への変換は以下の式で表される.

$$\mathsf{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_{\mathcal{C}}) = \left(\sum_{j=0}^{\ell-1} [a^{(j)} \cdot \hat{q}_j^{-1}]_{q_j} \cdot \hat{q}_j \pmod{p_i} \right)$$

ここで, $\hat{q}_j = \prod_{j' \neq j} q_{j'} \in \mathbb{Z}$ である. この変換による出力は, $a + Q \cdot e$ を基底 $\mathcal{B}$ によって RNS 表現した $[a + Q \cdot e]_{\mathcal{B}}$ と書ける. ここで, $e \in \mathbb{Z}$ は $|a + Q \cdot e| \leq \frac{\ell}{2} \cdot Q$ を満たすような整数である. 多項式 $a(x) \in \mathcal{R}_Q$ に対して, 係数ごとにこの変換を適用すれば多項式に対しても自然に

$$\mathsf{Conv}_{\mathcal{C}\to\mathcal{B}}(\cdot) : \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j} \to \prod_{i=0}^{k-1} \mathcal{R}_{p_i}$$

と拡張できる.

2.4 FullRNS-HEAAN

以下に FullRNS-HEAAN スキームの概要を示す. より詳細な処理にいては [14] を参考にされたい. 以下では, レベル ℓ の暗号文 ct, ct' を

$$\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}(x), c_1^{(j)}(x)) \right)_{0 \leq j < \ell} \in \prod_{j=0}^{\ell} \mathcal{R}_{q_j}^2,$$

$$\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}(x), c_1'^{(j)}(x)) \right)_{0 \leq j < \ell} \in \prod_{j=0}^{\ell} \mathcal{R}_{q_j}^2$$

と表す. また, $NTT(\cdot), INTT(\cdot)$ はそれぞれ Number Theoretic Transform (NTT), Inverse NTT (INTT) を多項式に対して適用することを表し, 以下では, 暗号文の多項式には NTT が適用されているものとする.

$\mathsf{Setup}(q, L, \eta; 1^\lambda)$.

セキュリティパラメータ λ を満たすような整数 q, レベル数 L, ビット精度 η が与えられたとき, 以下の処理を行う.

- $1 \leq j \leq L$ において, $q_j/q \in (1 - 2^{-\eta}, 1 + 2^{-\eta})$ であるような基底 $\mathcal{D} = \{p_0, ..., p_{k-1}, q_0, q_1, ..., q_L\}$ を選ぶ.
- 2 のべき乗数 N を選ぶ.
- $\mathcal{R}$ 上で秘密鍵の分布 χ_{key}, 暗号化鍵の分布 χ_{enc}, 誤差の分布 χ_{err} を決める.

ここで, $0 \leq \ell \leq L$ に対して, $\mathcal{B} = \{p_0, ..., p_{k-1}\}$, $\mathcal{C}_\ell = \{q_0, ..., q_\ell\}$, $\mathcal{D}_\ell = \mathcal{B} \cup \mathcal{C}_\ell = \{p_0, ..., p_{k-1}, q_0, ..., q_\ell\}$ とし, $P = \prod_{i=0}^{k-1} p_i$, $Q_\ell = \prod_{j=0}^{\ell} q_j$ とする. $\mathcal{B}$ は Special primes と呼ばれる鍵の切り替え処理で使用される素数であり, q に近い値である必要はないが, P は小さな鍵切り替え誤差を得るのに十分な大きさである必要がある.

KeyGen.

- $s(x) \leftarrow \chi_{key}$ をサンプリングし, $\mathsf{sk} \leftarrow (1, s(x))$ を秘密鍵とする.
- $(a^{(0)}(x), ..., a^{(L)}(x)) \leftarrow U\left(\prod_{j=0}^{L} \mathcal{R}_{q_j}\right)$ と $e(x) \leftarrow \chi_{err}$ をサンプリングし, $\mathsf{pk} \leftarrow \left(\mathsf{pk}^{(j)} = (b^{(j)}(x), a^{(j)}(x)) \in \mathcal{R}_{q_j}^2\right)_{0 \leq j \leq L}$ を公開鍵とする.
- $s(x), s^2(x)$ から評価鍵 evk を生成する.
- $s(x), s(x^{5^{idx}})$ から回転鍵 pk_{rot} を生成する.

ここで, 新たに生成された鍵の多項式はすべて NTT が適用される.

$\mathsf{Encrypt}_{\mathsf{pk}}(m)$.

$m(x) \in \mathcal{R}$ に対して, $v(x) \leftarrow \chi_{enc}$ と $e_0(x), e_1(x) \leftarrow \chi_{err}$ をサンプリングし, $\mathsf{ct} = \left(\mathsf{ct}^{(j)}\right)_{0 \leq j \leq L} \in \prod_{j=0}^{L} \mathcal{R}_{q_j}^2$ を暗号文として出力する. ここで, $0 \leq j \leq L$ で $\mathsf{ct}^{(j)} \leftarrow v(x) \cdot \mathsf{pk}^{(j)} + (m(x) + e_0(x), e_1(x)) \pmod{q_j}$ としており, $v(x), m(x), e_0(x), e_1(x)$ はすべて NTT が適用されている.

$\mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{ct})$.

$\left[\left\langle \mathsf{ct}^{(0)}, \mathsf{sk} \right\rangle\right]_{q_0}$ を出力する.

$\mathsf{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}([a(x)]_{\mathcal{C}_\ell})$.

$[\tilde{a}(x)]_{\mathcal{B}} \leftarrow \mathsf{Conv}_{\mathcal{C}_\ell \to \mathcal{B}}([a(x)]_{\mathcal{C}_\ell})$ を計算し, $[\tilde{a}(x)]_{\mathcal{D}_\ell} = ([\tilde{a}(x)]_{\mathcal{B}}, [a(x)]_{\mathcal{C}_\ell})$ を出力する.

$\mathsf{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}([\tilde{b}(x)]_{\mathcal{D}_\ell})$.

$[\tilde{a}(x)]_{\mathcal{C}_\ell} \leftarrow \mathsf{Conv}_{\mathcal{B} \to \mathcal{C}_\ell}([\tilde{b}(x)]_{\mathcal{B}})$ を計算し, $0 \leq j < \ell$ において, $b^{(j)}(x) = \left(\prod_{i=0}^{k-1} p_i\right)^{-1} \cdot (\tilde{b}^{(k+j)}(x) - \tilde{a}^{(j)}(x)) \pmod{q_j}$ を求め, $[b(x)]_{\mathcal{C}_\ell}$ を出力する.

$\mathsf{Add}(\mathsf{ct}, \mathsf{ct}')$.

$0 \leq j \leq \ell$ で $\mathsf{ct}_{add}^{(j)} \leftarrow \mathsf{ct}^{(j)} + \mathsf{ct}'^{(j)} \pmod{q_j}$ を計算する.

$\mathsf{CMult}(\mathsf{ct}, \mathsf{cnst} \in \mathbb{R})$.

$cnst' = |cnst| \cdot q$ を求め,

$$\mathsf{ct}_{cmult}^{(j)} \leftarrow \begin{cases} cnst' \cdot \mathsf{ct}^{(j)} \pmod{q_j} & (cnst \geq 0) \\ q_j - cnst' \cdot \mathsf{ct}^{(j)} \pmod{q_j} & (cnst < 0) \end{cases}$$

を計算する.

$\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')$.

(1) $0 \leq j \leq \ell$ で

$$\begin{aligned} d_0^{(j)}(x) &\leftarrow c_0^{(j)}(x) c_0'^{(j)}(x), \\ d_1^{(j)}(x) &\leftarrow c_0^{(j)}(x) c_1'^{(j)}(x) + c_1^{(j)}(x) c_0'^{(j)}(x), \\ d_2^{(j)}(x) &\leftarrow c_1^{(j)}(x) c_1'^{(j)}(x) \end{aligned}$$

を計算する. $(d_0^{(j)}(x), d_1^{(j)}(x), d_2^{(j)}(x) \pmod{q_j})$

(2) $d_2^{(j)}(x) \leftarrow INTT(d_2^{(j)}(x)), 0 \leq j \leq \ell$ とする.

(3) $\mathsf{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}([d_2(x)]_{\mathcal{C}_\ell}) = [\tilde{d}_2(x)]_{\mathcal{D}_\ell}$ を計算する.

(4) $\tilde{d}_2^{(j)}(x) \leftarrow NTT(\tilde{d}_2^{(j)}(x)), 0 \leq j \leq k + \ell$ とする.

(5) $\tilde{\mathsf{ct}} = \left(\tilde{\mathsf{ct}}^{(0)} = (\tilde{c}_0^{(0)}(x), \tilde{c}_1^{(0)}(x)), ..., \tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(k+\ell)}(x), \tilde{c}_1^{(k+\ell)}(x))\right) \in \prod_{i=0}^{k-1} \mathcal{R}_{p_i}^2 \times \prod_{j=0}^{\ell} \mathcal{R}_{q_j}^2$ を計算する．ここで，$0 \le i < k, 0 \le j \le \ell$ において，$\tilde{\mathsf{ct}}^{(i)} = \tilde{d}_2^{(i)}(x) \cdot \mathsf{evk}^{(i)} \pmod{p_i}, \tilde{\mathsf{ct}}^{(k+j)} = d_2^{(j)}(x) \cdot \mathsf{evk}^{(k+j)} \pmod{q_j}$ としている．

(6) $i = 0, 1$ に対して，$\tilde{c}_i^{(j)}(x) \leftarrow INTT(\tilde{c}_i^{(j)}(x)), 0 \le j \le k + \ell$ とする．

(7) $i = 0, 1$ に対して，$\left(\hat{c}_i^{(0)}(x), ..., \hat{c}_i^{(\ell)}(x)\right) \leftarrow \mathsf{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_i^{(0)}(x), ..., \tilde{c}_i^{(k+\ell)}(x)\right)$ を計算する．

(8) $i = 0, 1$ に対して，$\hat{c}_i^{(j)}(x) \leftarrow NTT(\hat{c}_i^{(j)}(x)), 0 \le j \le \ell$ とする．

(9) $0 \le j \le \ell$ で $\mathsf{ct}_{mult}^{(j)} \leftarrow (\hat{c}_0^{(j)}(x) + d_0^{(j)}(x), \hat{c}_1^{(j)}(x) + d_1^{(j)}(x)) \pmod{q_j}$ を計算する．

$\mathsf{Rescale}(\mathsf{ct})$.

$i = 0, 1, 0 \le j < \ell$ において，

(1) $c_i^{(j)}(x) \leftarrow \left[INTT(c_i^{(j)}(x))\right]_{q_j}$ とする．

(2) $c_i^{(j)}(x) \leftarrow NTT(c_i^{(j)}(x))$ とする．

(3) $\tilde{c}_i^{(j)}(x) \leftarrow q_\ell^{-1} \cdot \left(c_i^{(j)}(x) - c_i^{(\ell)}(x)\right) \pmod{q_j}$ を計算する．

暗号文 $\tilde{\mathsf{ct}} \leftarrow \left(\tilde{\mathsf{ct}}^{(j)} = (\tilde{c}_0^{(j)}(x), \tilde{c}_1^{(j)}(x))\right)_{0 \le j < \ell - 1} \in \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j}^2$ を出力する．

$\mathsf{LeftRot}_{\mathsf{pk}_{\mathsf{rot}}}(\mathsf{ct}, \mathsf{idx} \in \mathbb{Z})$.

(1) $0 \le j \le \ell$ で $\check{\mathsf{ct}}^{(j)} = \left(\check{c}_0^{(j)}(x), \check{c}_1^{(j)}(x))\right) = \left(c_0^{(j)}(x^{5^{idx}}), c_1^{(j)}(x^{5^{idx}}))\right)$ を計算する．

(2) $\check{c}_0^{(j)}(x) \leftarrow INTT(\check{c}_0^{(j)}(x)), 0 \le j \le \ell$ とする．

(3) $\mathsf{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}([\check{c}_0^{(j)}(x)]_{\mathcal{C}_\ell}) = [\tilde{c}_0^{(j)}(x)]_{\mathcal{D}_\ell}$ を計算する．

(4) $\tilde{c}_0^{(j)}(x) \leftarrow NTT(\tilde{c}_0^{(j)}(x)), 0 \le j \le k + \ell$ とする．

(5) $\tilde{\mathsf{ct}} = \left(\tilde{\mathsf{ct}}^{(0)} = (\tilde{c}_0^{(0)}(x), \tilde{c}_1^{(0)}(x)), ..., \tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(k+\ell)}(x), \tilde{c}_1^{(k+\ell)}(x))\right) \in \prod_{i=0}^{k-1} \mathcal{R}_{p_i}^2 \times \prod_{j=0}^{\ell} \mathcal{R}_{q_j}^2$ を計算する．ここで，$0 \le i < k, 0 \le j \le \ell$ において，$\tilde{\mathsf{ct}}^{(i)} = \tilde{c}_0^{(i)}(x) \cdot \mathsf{pk}_{rot}^{(i)} \pmod{p_i}, \tilde{\mathsf{ct}}^{(k+j)} = \check{c}_0^{(j)}(x) \cdot \mathsf{pk}_{rot}^{(k+j)} \pmod{q_j}$ としている．

(6) $i = 0, 1$ に対して，$\tilde{c}_i^{(j)}(x) \leftarrow INTT(\tilde{c}_i^{(j)}(x)), 0 \le j \le k + \ell$ とする．

(7) $i = 0, 1$ に対して，$\left(\hat{c}_i^{(0)}(x), ..., \hat{c}_i^{(\ell)}(x)\right) \leftarrow \mathsf{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_i^{(0)}(x), ..., \tilde{c}_i^{(k+\ell)}(x)\right)$ を計算する．

(8) $i = 0, 1$ に対して，$\hat{c}_i^{(j)}(x) \leftarrow NTT(\hat{c}_i^{(j)}(x)), 0 \le j \le \ell$ とする．

(9) $0 \le j \le \ell$ で $\mathsf{ct}_{rot}^{(j)} \leftarrow (\hat{c}_0^{(j)}(x), \check{c}_1^{(j)}(x) + \hat{c}_1^{(j)}(x)) \pmod{q_j}$ を計算する．

3. ブートストラッピング

ブートストラッピングは，暗号文上で復号回路を評価することによって，暗号文同士の乗算によって増加した暗号文ノイズを除去する処理である．

3.1 HEAAN 上でのブートストラッピング

HEAAN において，復号処理は $[\langle \mathsf{ct}, \mathsf{sk} \rangle]_q$ のように表されるため，ブートストラッピングの処理としては，乗算によって減少したレベルを初期値に戻した後に，剰余演算を暗号文上で行うことができれば良いと考えられる．しかし，剰余演算は準同型的に行うことが出来ない．そのため，[11] では剰余演算をスケーリングされた sin 関数の評価によって近似する手法が提案されている．HEAAN におけるブートストラッピングは，以下の 5 つの手順からなる．

(1) 暗号文のレベルを上げる (ModRaise)

(2) 多項式の係数に対して演算を行うために係数をパッキングする (CoefToSlot)

(3) exp 関数を評価する (EvalExp)

(4) exp 関数の評価値の虚部から sin 関数の評価値を取り出す (ImgExt)

(5) 評価値に対して CoefToSlot と逆の処理を行い，多項式の係数へ戻す (SlotToCoef)

これらの手順において，使用する回数が多い処理は Add, CMult, Mult, Rescale, LeftRot である．特に，Mult と LeftRot は他の処理に比べて多くの計算時間を要するため，これらの処理を高速化することがブートストラッピングの計算時間を減少させることに繋がると考えられる．

3.2 FullRNS-HEAAN 上でのブートストラッピング

HEAAN のブートストラッピングで行っている手順 2 ∼ 5 の処理は，RNS 表現を容易に適用可能であるため，各ステップで使用している関数を，対応する FullRNS-HEAAN の関数に置き換えることで実装可能である．

それに対し，暗号文のレベルを $\ell (< L)$ から L に上げる ModRaise 処理は，HEAAN 上の処理とは異なり，FullRNS-HEAAN では ModUp や ModDown のように暗号文の法の変更を近似的に行う必要がある．そこで，FullRNS-HEAAN に対応した ModRaise 処理を以下のように行う．

$0 \le \ell < L$ において $\mathcal{C}_\ell = \{q_0, ..., q_\ell\}, \bar{\mathcal{C}}_\ell = \{q_{\ell+1}, ..., q_L\}$ とし，$\mathcal{C}_L = \{q_0, ..., q_L\}$ とする．このとき，$\mathsf{ModRaise}_{\mathcal{C}_\ell \to \mathcal{C}_L}$ は以下のように行う．

$\mathsf{ModRaise}_{\mathcal{C}_\ell \to \mathcal{C}_L}([a(x)]_{\mathcal{C}_\ell})$.

$[\tilde{a}(x)]_{\bar{\mathcal{C}}_\ell} \leftarrow \mathsf{Conv}_{\mathcal{C}_\ell \to \bar{\mathcal{C}}_\ell}([a(x)]_{\mathcal{C}_\ell})$ を計算し，$[\tilde{a}(x)]_{\mathcal{C}_L} = ([a(x)]_{\mathcal{C}_\ell}, [\tilde{a}(x)]_{\bar{\mathcal{C}}_\ell})$ を出力する．

この処理は，Mult や LeftRot で用いられている ModUp と同様の処理を利用している．

4. FullRNS-HEAAN の並列化

本章では，HEAAN と FullRNS-HEAAN に共通して見られる剰余多項式環上での演算における係数ごとの処理の独立性に着目した **Coefficient-wise 並列**と，FullRNS-HEAAN に見られる RNS 表現上の演算における法ごとの処理の独立性に着目した **Modulus-wise 並列**を考える．

4.1 Coefficient-wise 並列

HEAAN および FullRNS-HEAAN では, 剰余多項式環上での乗算を高速に行うために, 多項式を Number Theoretic Transform (NTT) によって別の多項式に変換する. NTT 適用前の多項式の加算と乗算は, それぞれ NTT 適用後の多項式において係数ごとに独立して和と積を計算することに対応している. そのため, 剰余多項式環上での演算では, 多項式の係数ごとの処理を並列化する Coefficient-wise 並列を考えることができる. Coefficient-wise 並列は, 値が固定されている多項式の次元数 N 個分の処理を並列化するため, 暗号文のレベルに依らず処理は全スロットに均等に分割され, 安定した高速化が期待できる. しかし, NTT やその逆変換 INTT (Inverse NTT) の処理では, 多項式の係数ごとに処理は独立していない。そのため、Mult, LeftRot の手順 2,4,6,8 や Rescale の手順 1, 2 における NTT と INTT の処理はシングルスレッドで実行しなければならず、計算時間のボトルネックになると考えられる.

4.2 Modulus-wise 並列

FullRNS-HEAAN では RNS 表現を用いており, RNS 表現された値の演算においては, それぞれの法における処理は独立している. そのため, 法ごとに処理を並列化する Modulus-wise 並列を考えることができる. Modulus-wise 並列では, NTT や INTT を含むほとんどの処理において並列化を行うことができる. しかし, 基底を構成する法の数は Rescale を行うごとに変化し, 法の数によっては Modulus-wise 並列は全スレッドに対して均等に処理が分割されず, 高速化されにくい場合があると考えられる. また, LeftRot の手順 1 の処理などは, メモリアクセスや並列化によるオーバーヘッドのために, Modulus-wise 並列よりも Coefficient-wise 並列が適していると考えられ, 処理によって最適な並列化手法を選択する必要があると考えられる.

4.3 Coef&Mod-wise 並列

FullRNS-HEAAN では, HEAAN と同様に剰余多項式環上で演算を行うため, Modulus-wise 並列に加えて Coefficient-wise 並列も同時に考えることができる. そこで, 2 つの並列化手法を組み合わせた **Coef&Mod-wise 並列**を考える. Coef&Mod-wise 並列では, 各処理において最適な並列手法を選択することができるため, NTT と INTT には Modulus-wise 並列を適用し、LeftRot の手順 1 などの Coefficient-wise 並列が適している処理には Coefficient-wise 並列を適用することができる。また, Coefficient-wise 並列と Modulus-wise 並列の両方が適用可能である場合, 係数と法に関する 2 重ループを 1 重ループ化することによって, Modulus-wise 並列における法によって処理を均等に分割することができないという問題を解消することができ, 同時に, 並列化のオーバーヘッドを削減することができると考えられる.

5. 実装結果

本章では, FullRNS-HEAAN 上でのブートストラッピングの実装結果と, 処理の並列化による高速化の結果を示す. 実装は FullRNS-HEAAN ライブラリ [16] を利用し, ブートストラッピングは [15] を参考に実装した. 実験環境は, CPU : Intel(R) Xeon(R) E5-4627 v3 @ 2.60GHz, プロセッサ数 : 40, g++ : 6.3.0, OpenMP ; 4.0 である. 以下では, パラメータを

- $N = 2^{16}$: 多項式の次元数
- $q_0 \approx 2^{61}$: レベル 0 の法に対応する法
- $q_\ell \approx 2^{55} (\ell = 1, ..., L)$: レベル $1 \sim L$ に対応する法
- $\sigma = 3.2$: 誤差多項式の係数をサンプルする際の離散ガウシアン分布の標準偏差
- $h = 64$: 秘密鍵を符号付き 2 値多項式の集合からランダムに選択する際のハミング重み
- $\lambda = 80bit$: セキュリティパラメータ
- $r = 8$: sin 関数を評価する際の繰り返し数
- $radix = 8$: FFT における基数

として実験を行っている. 上記のパラメータは Albrecht らの推定 [8] を参考に, 少なくとも 80bit のセキュリティレベルを達成するように設定されている. 暗号文には $N/2 = 32768$ 個の値をパッキングしている. 同様のパラメータで実験するために, HEAAN では最大の暗号文の法 Q_L を $2^{61+55 \cdot L}$ とした. また, 実行時間は Add,CMult,Mult,Rescale,LeftRot においては 100 回, Bootstrapping では 10 回の平均を取った.

5.1 FullRNS-HEAAN 上でのブートストラッピング

スキームの変更による違いを確認するために, ブートストラッピングの HEAAN における実装と FullRNS-HEAAN における実装を実行時間と精度で比較する. そのため, それぞれの実装で Add, CMult, Mult, Rescale, LeftRot, Bootstrapping の実行時間を計測し, ブートストラッピングの前後における誤差を計測した. 暗号文の初期レベル L は 32 とし, 1 スレッドで実行している. 結果は表 1 のようになった.

FullRNS-HEAAN では, Add, CMult, Rescale 操作において処理は遅くなっている. これは, Add, CMult は HEAAN においても高速に動作しており, 多倍精度での演算を避けたことによる 1 回の処理の高速化よりも, 処理回数が増加したことによる影響のほうが大きくなったためであると考えられる. また, Rescale は HEAAN での処理と比べて NTT などの処理が増えたためであると考えられる. 対して, ブートストラッピングの処理の大部分を占めていた Mult, LeftRot はそれぞれ 2.76, 1.28 倍高速化され, 結果的にブートストラッピングは約 2.19 倍高速になった. また, ブートストラッピング後の暗号文において, 両方のスキームで精度は 11bit, レベルは 7 となった.

5.2 FullRNS-HEAAN の並列処理による高速化

4 章で考えた Coefficient-wise 並列, Modulus-wise 並列, Coef&Mod-wise 並列を用いて並列化した場合の実行時間の変化を調べるために以下の 2 つの実験を行った。

5.2.1 スレッド数を変化させた場合の実行時間

スレッド数を変化させた場合の各並列手法における実行時間の変化を調べるために、スレッド数を $1, 2, 4, 8, 16, 32$

表 1　HEAAN と FullRNS-HEAAN の実験結果の比較 (L = 32, #Threads = 1)
Table 1　Comparison of experimental results of HEAAN and FullRNS-HEAAN. (L = 32, #Threads = 1)

Scheme	Add(ms)	CMult(ms)	Mult(ms)	Rescale(ms)	LeftRot(ms)	Bootstrapping(ms)
HEAAN	34.743	31.385	8956.36	10.563	3854.39	494082
FullRNS-HEAAN	57.59	97.25	3246.3	460.39	3018.38	224006

と変化させ, それぞれのスレッド数における Add, CMult, Mult, Rescale, LeftRot の実行時間を計測した. 暗号文のレベル L は 16 としている。結果は図 1 のようになった. 縦軸は実行時間を示し, 横軸はスレッド数を示す.

いずれの並列化手法も, スレッド数の増加に伴い実行時間は減少していくことがわかる. Modulus-wise 並列と Coef&Mod-wise 並列は, 4 章で考察した通り, NTT の処理が必要となる Mult, Rescale, LeftRot においては, Coefficient-wise 並列よりも高速化されていることがわかるが, スレッド数の変化による差はほとんど見られない. また, NTT の処理を行わない Add, CMult においても Coefficient-wise 並列が他の並列手法に比べて遅くなっているのは, 実装上でスレッドへの処理の分割を基底の各法ごとに行っており, 並列化のオーバーヘッドが大きいためであると考えられる.

5.2.2　基底を構成する法の数を変化させた場合の実行時間

次に, 基底を構成する法の数を変化させた場合の各並列手法における実行時間の変化を調べるために、法の数に対応している暗号文のレベル L を 1 ∼ 32 と変化させ, それぞれのレベルにおける Add, Mult, CMult, Rescale, LeftRot の実行時間を計測した. スレッド数は 16 としている。結果は図 2 のようになった. 縦軸は実行時間を示し, 横軸はレベル数を示す.

Coefficient-wise 並列では, レベルの増加に伴い線形に実行時間が増加している. Modulus-wise 並列では, 4 章で考察した通り, レベルがスレッド数に比べて小さい時やスレッド数を超えたあたりから急に実行時間が増加している. Coef&Mod-wise 並列では, Modulus-wise 並列に見られた急な実行時間の変化が抑えられており, ほとんどのレベルにおいて実行時間が最も少なくなっている. また、Modulus-wise 並列と Coef&Mod-wise 並列を比較すると、Modulus-wise 並列においても処理が均等に分割されていると考えられるレベル数が 32 の場合に、Add, CMult, Mult, Rescale の処理ではほとんど差が見られないが、LeftRot では Coef&Mod-wise 並列の方が約 140ms ほど高速になっており、Coef&Mod-wise 並列において各処理で最適な並列化手法を選択したことによる効果が確認できる。

以上より, Coef&Mod-wise 並列が最も高速で, 実行時間のレベルへの依存が少ないため, 最適な並列化手法であると考えられる.

5.3　ブートストラッピングの高速化

HEAAN 上のブートストラッピングを 1 スレッドおよび 32 スレッドで実行した場合の実行時間と, 今回実装した FullRNS-HEAAN 上のブートストラッピングを 1 スレッドおよび 32 スレッドで Coefficient-wise, Modulus-wise, Coef&Mod-wise 並列で実行した場合の実行時間を計測し比較を行った. 暗号文の初期レベル L を 32 としている. 結果は表 2 のようになった. ここで, 表 2 における Amortized Time は, 要素 1 つあたりの実行時間を表している.

HEAAN で 1 スレッドの実行時間と FullRNS-HEAAN で 1 スレッドの実行時間を比較すると, FullRNS-HEAAN で実装したことによって約 2.2 倍の高速化を達成していることがわかる. また, Coef&Mod-wise 並列を用いることによって, 1 スレッドの場合と比較して約 7.46 倍の高速化を達成していることがわかる. 結果として, 要素 1 つあたりのブートストラッピングの実行時間は約 0.92ms となり, 1ms 以下にするという目標を達成することができた.

6.　おわりに

本稿では, FullRNS-HEAAN に対応したブートストラッピングを考案し, 実装を行い, 実験によって HEAAN 上のブートストラッピングとの実行時間と精度に関する比較を行った. その結果, FullRNS-HEAAN 上でブートストラッピングを実装することによって約 2.2 倍高速化され, 精度は変わらないことが確認できた. さらに, FullRNS-HEAAN において Coefficient-wise 並列, Modulus-wise 並列, Coef&Mod-wise 並列の 3 つの並列化手法を比較検討し, 処理の高速化を行った. その結果, Coef&Mod-wise 並列を用いてブートストラッピングを行った場合に, 要素 1 つあたりのブートストラッピングの実行時間を約 0.92ms とすることができ, 目標であった 1ms 以下を達成した.

参考文献

[1] C. Gentry. : *Fully homomorphic encryption using ideal lattices*, In In Proc. STOC, pages 169–178, 2009.

[2] M. v. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. : *Fully homomorphic encryption over the integers*, In Advances in Cryptology–EUROCRYPT 2010, pages 24–43. Springer, 2010.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan.: *(Leveled) fully homomorphic encryption without bootstrapping*, In Shafi Goldwasser, editor, ITCS2012, pages 309–325. ACM, January 2012.

[4] S. Halevi and V. Shoup. : *Bootstrapping for HElib*, In Advances in Cryptology–EUROCRYPT 2015, pages 641–670. Springer, 2015.

[5] S. Halevi, Y. Polyakov, and V. Shoup. : *An improved RNS variant of the BFV homomorphic encryption scheme*, Cryptology ePrint Archive, Report 2018/117, 2018. https://eprint.iacr.org/2018/117.

[6] Junfeng Fan and Frederik Vercauteren. : *Somewhat practical fully homomorphic encryption*,

*1 HEAAN ライブラリ [15] で使用されている多倍長整数ライブラリ NTL 中の並列化

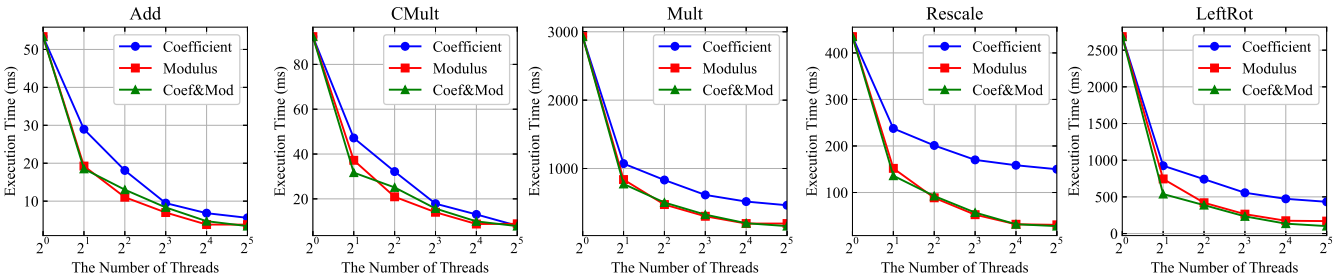

図 1　スレッド数を変化させた場合の各並列手法における各処理の実行時間 (L = 16)

Fig. 1　Execution time of each process in each parallel method when the number of threads is changed. (L = 16)

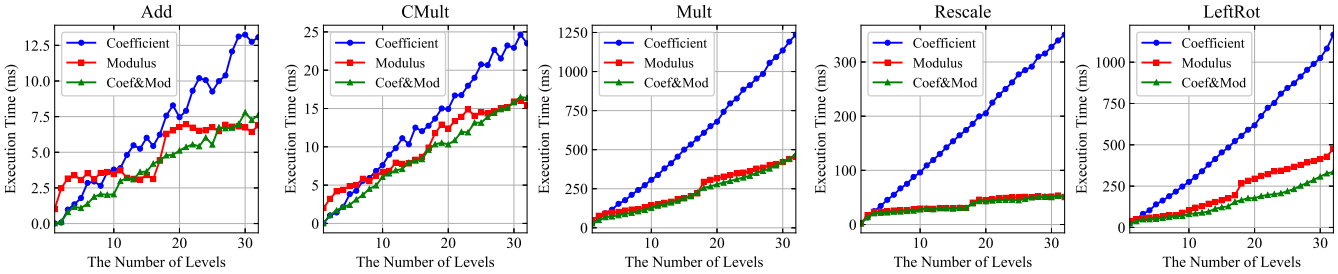

図 2　レベル数を変化させた場合の各並列手法における各処理の実行時間 (#Threads = 16)

Fig. 2　Execution time of each process in each parallel method when the number of levels is changed. (#Threads = 16)

表 2　HEAAN と FullRNS-HEAAN 上でのブートストラッピングの実行時間の比較 (L = 32)

Table 2　Comparison of bootstrapping execution time on HEAAN and FullRNS-HEAAN. (L = 32)

Scheme	Parallel	#Threads	Bootstrapping(ms)	Amortized Time(ms)
HEAAN	None	1	490445	14.97
	NTL*1	32	132281	4.04
FullRNS-HEAAN	None	1	224006	6.84
	Coefficient-wise	32	131247	4.01
	Modulus-wise	32	38098	1.16
	Coef&Mod-wise	32	30034	0.92

Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.

[7] L. Ducas and D. Micciancio. : *FHEW: Bootstrapping homomorphic encryption in less than a second*, In Advances in Cryptology–EUROCRYPT 2015, pages 617–640. Springer, 2015.

[8] M. R. Albrecht, R. Player, and S. Scott. : *On the concrete hardness of learning with errors*, Journal of Mathematical Cryptology, 9(3):169–203, 2015.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song. : *Homomorphic encryption for arithmetic of approximate numbers*, In Advances in Cryptology–ASIACRYPT 2017, pages 409–437. Springer, 2017.

[10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabach'ene. : *TFHE: Fast Fully Homomorphic Encryption over the Torus*, IACR Cryptology ePrint Archive, 2018:421.

[11] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. : *Bootstrapping for approximate homomorphic encryption*, In Advanced in Cryptology–EUROCRYPT 2018, pages 360–384. Springer, 2018.

[12] J. H. Cheon, K. Han, and M. Hhan.: *Faster homomorphic discrete fourier transforms and improved fhe bootstrapping*, Cryptology ePrint Archive, Report 2018/1073, 2018. https://eprint.iacr.org/2018/1073, To appear IEEE Access.

[13] H. Chen, I. Chillotti, and Y. Song. : it Improved bootstrapping for approximate homomorphic encryption, Cryptology ePrint Archive, Report 2018/1043, 2018. https://eprint.iacr.org/2018/1043, to appear in EUROCRYPT 2019.

[14] J. H. Cheon, K. Han, A. Kim, M. Kim, Y. Song.: *A full RNS variant of approximate homomorphic encryption*, In SAC 2018, pp. 347–368, 2018.

[15] Kyoohyung, Han. : *HEAAN*, https://github.com/HanKyoohyung/ImprovedLT

[16] Kyoohyung, Han. : *FullRNS-HEAAN*, https://github.com/HanKyoohyung/FullRNS-HEAAN