B-002

完全準同型暗号の高速化に向けたハードウェア利活用に関する研究調査

A Survey of the Hardware Implementations for Improving Performance of Fully Homomorphic Encryption

井上 紘太朗† 鈴木 拓也† 山名 早人†

Kotaro Inoue Takuya Suzuki Hayato Yamana

1 はじめに

近年，クラウドコンピューティングの普及に伴い，様々な計算をクラウド上へ委託して行うケースが増えてきている [1]．しかし，クラウド上へ計算委託を行う際，クラウド上に保管されているデータをどのように保護するかが課題となる．データの保護の方法には，データへの改竄耐性や，データに含まれる機密情報の漏洩を防ぐことが挙げられる．本稿では，これらの内，データに含まれる機密情報が漏洩することを防ぐことを対象とし，その中でも秘密計算の一つである準同型暗号（Homomorphic Encryption: HE）を対象とした高速化研究において，ハードウェアがどのように利活用されているかを調査する．

準同型暗号は，データを暗号化した状態で，加算や乗算を行うことができる暗号技術である．暗号化した状態での加算及び乗算をそれぞれ準同型加算及び準同型乗算と呼ぶ．準同型暗号は，その方式によって，行える演算の種類や回数が異なる．準同型加算もしくは準同型乗算のいずれかが，演算回数の制限なく行える方式を Partially HE（PHE）と呼ぶ．また，準同型加算と準同型乗算が共に実行可能だが，準同型加算もしくは準同型乗算のいずれかの実行回数に制約が存在するものを Somewhat HE（SHE）とよぶ [2]．PHE や SHE は，実行可能な演算や回数に制約が存在するため，実装できるアプリケーションや用途が限定的である．しかし，2009 年に Gentry によって発表された完全準同型暗号（Fully HE: FHE）により，PHE や SHE に存在した制限を解決し，準同型加算と準同型乗算が任意の回数行えるようになった [3]．Gentry は，Bootstrapping と呼ばれる手法を導入し，SHE から FHE を構成した．しかし，Bootstrapping の処理は，Gentry の方式で最長で 31 分 [4] と，非常に時間がかかるため，Bootstrapping を用いない [5]，あるいは適用回数を減らす工夫 [6] や，Bootstrapping そのものを高速化する [7] といった方向で研究が進められてきた．その中で，回路全体における準同型乗算回数（回路の深さ）をパラメータとして設定し，その準同型乗算回数分だけ準同型乗算が可能な方式が提案されている．これらは，Leveled FHE[1] と呼ばれ，実用化が進められている．また，Bootstrapping の他にも，完全準同型暗号の方式によっては，Relinearization や modulus switching といった完全準同型暗号における固有の演算が存在する．Relinearization については，Bootstrapping と同様に，適用回数を減らすための研究が行われている [8]．しかし，完全準同型暗号は，実用に向けて解決すべき，性能上の問題を抱えている．暗号文上での演算は，通常の演算に比べて時間・空間計算量の両面で劣っている．さらに，Bootstrapping などの完全準同型暗号特有の演算により，多くの計算時間を要している．

そこで，完全準同型暗号の演算を高速化する手法として，近年注目されているのが，GPU（Graphics Processing Unit）や FPGA（Field Programmable Gate Array），ASIC（Application Specific Integrated Circuit）といったハードウェアアクセラレータである．これらのハードウェアアクセラレータは，CPU と協調して，もしくはハードウェアアクセラレータ単体で動作し，ハードウェアアクセラレータ上での実行に適した処理を委託することにより，処理性能向上を実現している．

本稿では，完全準同型暗号の高速化を目的としたハードウェア利活用の研究動向と，現状の研究課題についてまとめる．本稿は，以下の構成を取る．3 節で調査方法についてまとめ，4 節で調査結果を報告する．その後，2 節で完全準同型暗号について概要を述べた上で，5 節で調査結果を踏まえた現状の課題点について議論を行い，最後に 6 節でまとめる．

2 完全準同型暗号

本節では，完全準同型暗号の概要と，完全準同型暗号が抱える実用上の課題について述べる．

2.1 完全準同型暗号の概要

準同型暗号は，暗号化アルゴリズムを E，復号アルゴリズムを D，演算を $\cdot$，取りうる平文の集合を $\mathcal{M}$ とした時に，式 (1) の制約下において，式 (2) のような関係 [2]，もしくは式 (3) のような関係が成り立つ暗号方式である．

$$\forall m_1, m_2, m_1 \cdot m_2 \in \mathcal{M} \tag{1}$$

$$E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2) \tag{2}$$

$$D\left(E(m_1) \cdot E(m_2)\right) = D\left(E(m_1 \cdot m_2)\right) \tag{3}$$

準同型暗号の方式は，$\cdot$ として定義されている演算や，評価可能な演算回数により，PHE，SHE，FHE に分類される．準同型暗号は，安全性を保証する基盤として，数学的な問題を採用している．数学的な問題としては，Ring-LWE(Learning With Error) や NTRU，Ring-GSW といった問題が挙げられる．本稿では，特に Ring-LWE をベースとした暗号方式について対象とする．Ring-LWE ベースの暗号方式は，式 (3) に示す関係が成り立つ暗号方式であり，FV 方式や B/FV 方式，BGV 方式といった，平文空間を整数環とする方式や，CKKS 方式と呼ばれる，式 (3) の両辺の復号結果のそれぞれに誤差が含まれることを許容することで，固定小数点での実数や複素数を扱うことができる方式が存在する．

† 早稲田大学 Waseda University

1） 文献によっては，Leveled FHE のことを SHE と呼称している場合もある．

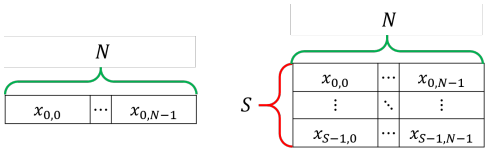

図 1　平文のデータ構造 (左) と暗号文のデータ構造 (右)

Ring-LWE ベースの暗号方式では，多項式環と呼ばれる数学理論が用いられている．ここでは，多項式環についての説明は行わず，Ring-LWE ベースの暗号方式におけるデータ構造について簡単に説明する．図 1 に Ring-LWE ベースの暗号方式におけるデータ構造の一例を示す．ただし，図 1 における $x_{i,j}$ の値は，法を Q とする整数である．Q の初期値はアプリケーション設計者によって決定され，modulus switching と呼ばれる演算を行なうことで増減する．この Q の値は 1 ワードよりも大きいことがあるため，$x_{i,j}$ は多倍長整数として扱われることがある．したがって，$x_{i,j}$ を B[Byte] とすると，平文及び暗号文の 1 つあたりのデータサイズは，それぞれ $B \times N$[Byte] と $B \times S \times N$[Byte] となる．また，暗号文においては，S は 2 以上の整数であり，平文では $S = 1$ である．準同型乗算への入力の長さをそれぞれ S_1 及び S_2 とすると，出力暗号文の長さは $S_1 + S_2 - 1$ となる．したがって，入力が 2 つの暗号文である準同型乗算では，出力暗号文の長さは増加する．暗号文の長さが大きいほど，空間計算量及び時間計算量が増加するため，Relinearization と呼ばれる演算を適用することで，長さを 1 小さくすることができる．また，準同型乗算を複数回適用すると，出力暗号文に含まれるノイズ量が指数関数的に大きくなる．ノイズが一定値を超えると正しく復号できなくなるため，BGV 方式や CKKS 方式では，modulus switching の一つである modulus reduction や rescaling と呼ばれる演算が，暗号文に含まれるノイズ量を管理するために適用される．ただし，modulus reduction や rescaling を適用すると，法 Q の値が小さくなる．法 Q の値が一定の値まで小さくなった暗号文に対して準同型乗算を適用すると正しく復号できなくなる．ここで，Bootstrapping を適用することで，ノイズ量を削減しつつ，かつ法 Q を再び大きな値とすることができ，任意の回数の準同型乗算を適用することができるようになる．

準同型演算への 2 つの入力暗号文をそれぞれ行列として扱うと，準同型加算は，行列和とすることができる．したがって，準同型加算の計算量は $O(S \times N)$ である．同様に，準同型乗算は，各行ごとの負巡回畳み込みであるため，計算量は $O(S^2 \times N^2)$ となる．ここで，各行に対して FFT（Fast Fourier Transform: 高速フーリエ変換）[2] や NTT（Number Theoretic Transform: 数論変換）を適用して変換した後に，要素ごとの乗算を行うことで，計算量を $O(S \times N \log N)$ に削減することができる．ただし，modulus switching や Relinearization といった演算の過程では，FFT や NTT で変換した後の暗号文に対しては適用できない計算が存在するため，適宜 FFT や NTT の逆変換を適用する必要がある．

また，ワードサイズを W[Byte] とすると，CRT（Chinese Remainder Theorem: 中国剰余定理）を用いて，剰余表現（Residue Number System: RNS）[9] とすることで，法 Q を $Q = \prod_{i=0}^{T-1} q_i, \log_2 q_i < W$ とすることで，多倍長整数を用いる代わりに複数の 1 ワードの値を用いて平文や暗号文を構成することができる．ただし，異なる $i \in [0, T)$ と $j \in [0, T)$ に対して，q_i と q_j は互いに素であるといった制約が必要である．また，方式や使用するアルゴリズムによっては，法 q_i にさらに他の制約がかかる．RNS を使用する場合のデータ構造を図 2 に示す．$\hat{x}_{i,j,k}, i \in [0, S), j \in [0, T), k \in [0, N)$ は，法を q_j とした値である．平文及び暗号文のデータサイズは，$W \times T \times N$[Byte] 及び $W \times S \times T \times N$[Byte] となる．したがって，$W \times T \geq B$ であるため，平文や暗号文そのものの空間計算量を削減することはできないが，並列性を高めることができる．

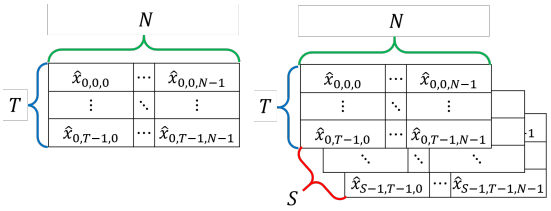

図 2　剰余表現における平文のデータ構造 (左) と暗号文のデータ構造 (右)

2.2　実用上の課題

完全準同型暗号の演算では，Bootstrapping や Relinearization の実行時間が長く，その次に準同型乗算が長い．また，完全準同型演算を構成する演算としては，FFT や NTT，及び FFT や NTT の逆変換にかかる実行時間が長い．したがって，1 節で示したように，Bootstrapping や Relinearization の実行回数を削減して高速化する方法の他に，Bootstrapping や Relinearization における演算を高速化する方法と，FFT や NTT を高速化することで，完全準同型暗号における複数の演算を高速化する方法がある．

低レベルな演算でのボトルネックとしては，多倍長整数演算と剰余演算が挙げられる．本稿で扱う Ring-LWE ベースの完全準同型暗号におけるデータ構造は 2.1 項で示したように，剰余表現ではない場合は平文や暗号文を構成する値は多倍長整数であり，剰余表現での平文や暗号文でも，$q_i > W/2$ を満たす場合では乗算を適用した際に多倍長演算が必要となる．また，剰余演算で用いられる除算命令や剰余命令は，多くのクロック数を必要とするため，遅いことが知られている．準同型暗号における演算では，剰余演算を大量に行うことから，剰余演算についても高速化の対象となる．

3　調査方法

本稿では，以下の手順に分割して研究調査を行った．

1. 調査目的を設定する．
2. 論文データベースを用いてキーワード検索を行う．
3. 検索結果から，調査目的に合致する論文を，手作業で抽出する．

2）　単に FFT といった場合，浮動小数点を用いるものを指す．

3.1 手順 1: 調査目的の設定

本稿では，完全準同型暗号の高速化を目的としたハードウェア利活用の研究動向と，現状の研究課題について調査することを目的としている．そこで，本研究の対象とする論文を，以下のように定める．

- 査読付き国際学会，あるいは査読付き論文誌で発表されている．
- 下記のいずれかを満たす．
  - 完全準同型暗号，特に Ring-LWE ベースの暗号方式の演算の一部，あるいは全てをハードウェアアクセラレータ上で実装している．
  - 完全準同型暗号の演算を実装しているわけではないが，応用先として完全準同型暗号を挙げている．

以降の手順では，上記の条件に合致する研究を対象として，調査を行う．

3.2 手順 2: 論文データベースを用いたキーワード検索

次に，論文データベースを用いて，キーワードによる検索を行った．本稿では，査読付き国際学会，あるいは査読付き論文誌で発表されている論文を対象としている．調査対象の論文を網羅的に収集するため，論文データベースとして Scopus を選定した．また，検索条件としては，以下の項目を指定した．ただし，キーワード検索の対象は，論文のタイトル，著者抄録，キーワードである．完全準同型暗号を，単に「準同型暗号」と表記している場合に対応するため，キーワードとしては“homomorphic encryption”を選定した．また検索を実施した日付は，2020 年 5 月 12 日である．

- キーワード：”homomorphic encryption” AND (”GPU” OR ”FPGA” OR ”ASIC” OR ”hardware”)
- 出版年：2014 年～2020 年
- 文献種別：Conference Paper または Article

3.3 手順 3: 調査対象論文の抽出作業

次に，手順 2 で得られた検索結果から，本稿の対象とする論文を選定した．選定は，手順 1 で定めた条件をもとに，タイトルと抄録の内容を確認する形で行った．

4 調査結果

4.1 概要

手順 1 及び手順 2 の結果，188 件の論文がヒットした．さらに，手順 3 を経て，36 件に絞り込んだ．この 36 件の論文は，下記の 3 種類に分けられる．

1. FHE の演算をハードウェア上で全て実装している，あるいは CPU や他のハードウェアアクセラレータと協調動作をした上で FHE の演算を実装している
2. FHE の演算で内部的に用いられているアルゴリズムのみを実装している

1 つ目の FHE の演算を全て実装した論文としては，例えば，準同型乗算の高速化を目的とした研究 [10–15] や，暗号化や復号の高速化を目的とした研究 [16–18] が挙げられる．

2 つ目の，FHE の演算で内部的に用いられているアルゴリズムのみを実装した論文については，FHE の演算を実装した上で包括的な測定を行っていないものが該当する．例えば，NTT をハードウェアアクセラレータ上で実装した論文 [19, 20] では，NTT に対する実行時間の計測実験を実施したが，完全準同型暗号としての実行時間は計測していない．この場合，CPU のメインメモリとハードェアアクセラレータ間のデータ転送による遅延といった影響を考慮していない．ゆえに，5 節における説明の対象からは除外した．

上記に含まれないものについては，アプリケーションのハードウェア実装に特化した研究 [21–23] が該当する．これらの論文は，準同型暗号に対する汎用的な性能改善を目的とした研究ではない．したがって，5 節における説明の対象からは除外した．

4.2 論文数の全体動向

図 3 は，各年で出版された論文数の推移を示したものである．論文数は増加傾向にあることが分かる．

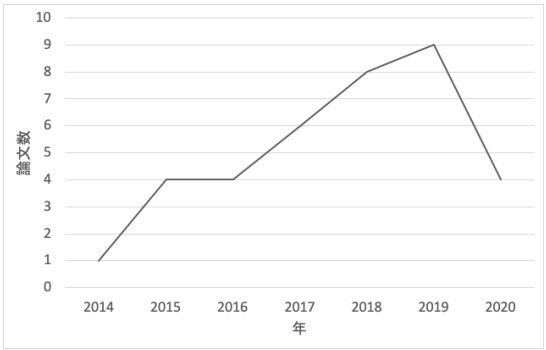

図 3 年ごとの論文数の推移

4.3 ハードウェアアクセラレータ種別ごとの研究動向

次に，ハードウェアアクセラレータ種別ごとの研究動向についてまとめる．図 4 は，各年に出版された論文を，ハードウェアアクセラレータ種別ごとに積み上げ棒グラフとして表したものである．なお，GPU と FPGA を併用するといった，異なる種別のハードウェアアクセラレータを併用する手法は存在していなかった．図 4 における 2014 年から 2019 年の論文数よると，GPU もしくは FPGA を用いる手法の論文数は増加傾向にあることが分かる．一方で，ASIC を用いる手法の論文数は毎年 0～2 本である．ASIC は FPGA と異なり，作成後に回路を変更することができないため，完全準同型暗号に適したアーキテクチャや回路構成を検討している段階であると言える．

5 実用上の課題に対する既存研究の提案手法

本節では，2 節で述べた完全準同型暗号の実用上の課題点について，既存研究の提案手法をまとめる．

表 1 に，ハードウェア上に完全準同型暗号の処理を実装している先行研究について，代表的なものをまとめた．

表 1　Ring-LWE ベースの準同型暗号に関する代表的なハードウェア実装

著者	種別	型番	多項式乗算アルゴリズム	暗号方式	実装詳細
Sadegh ら (2020) [11]	FPGA	Intel Stratix 10 GX 2800	NTT (負巡回畳込み) + CRT	CKKS	NTT，乗算，KeySwitch のそれぞれについて専用のコアを実装した．コアを組み合わせることにより，パラメータの変化に合わせてスケールする設計．438bit のセキュリティレベルで CPU 実装と比較して，準同型乗算と Relinealization の合計実行時間が 174 倍高速化を達成した．
Mert ら (2020) [16]	FPGA	Xilinx XC7VX485T	NTT (4-step)	B/FV	多項式乗算について，CPU 実装と比較して 21 倍の高速化を実現した．さらに，Iterative NTT と比較して FPGA のリソース消費量を削減できることを示した．
Turan ら (2020) [12]	FPGA	Xilinx XCVU9P (AWS F1)	NTT (Iterative) + 近似 CRT [24]	B/FV	AWS の F1 インスタンスに特化した実装した．6 枚の FPGA を使用し，著者らの以前の実装 [13] と比べて約 2 倍の高速化を達成した．
Mert ら (2019) [17]	FPGA	Xilinx XC7VX485T	NTT (Iterative)	B/FV	NTT ベースの多項式乗算器を FPGA 上へ実装した．多項式乗算について，CPU 実装と比較して 11 倍の高速化を達成した．
Lupascu ら (2019) [10]	GPU	NVIDIA K40M	NTT + CRT	BGV	分割された moduli ごとに GPU を割り当てる．4GPU で並列に実行し，CPU 実装に比べて乗算に対して 8 倍の高速化を達成した．
Sinha ら (2019) [13]	FPGA	Xilinx XCZU9EG	NTT (Iterative) + 近似 CRT [24]	B/FV	ソフトウェア実装との比較は未記載である．乗算のレイテンシは 4.458 ミリ秒と報告している．
Roy ら (2018) [14]	FPGA	Xilinx XC6VLX240T	NTT (Iterative) + CRT	B/FV	メモリ領域節約のためひねり係数は on-the-fly で計算した．B/FV 方式の CPU 実装との比較は未記載である．
Migliore ら (2018) [15]	FPGA	Intel Stratix V GX	Karatsuba	B/FV	NTT と Karatsuba のそれぞれで準同型乗算の実装を行った．N が 6144 より小さい場合，Karatsuba を用いたものは，NTT の FPGA 実装と比較した場合，準同型乗算を速く実行できるだけでなく，FPGA のリソース使用量を半分に抑えることを達成した．
Migliore ら (2017) [18]	FPGA	Intel Stratix V GX	Karatsuba	B/FV	多項式を複数に分割してバッチ処理を行うことにより，CPU 実装の NFLlib と比較して，1 回の暗号化の時間を 1.5 倍の高速化を達成した．

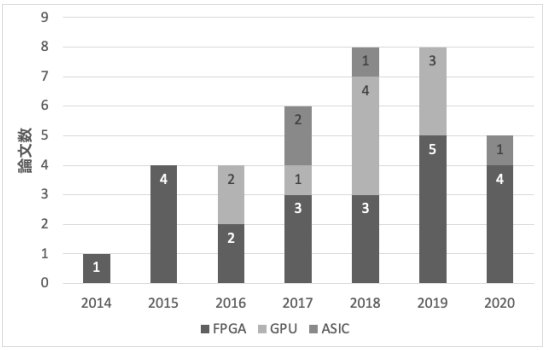

図 4　ハードウェアアクセラレータ種別ごとの研究数の推移

先行研究においては，準同型乗算においてボトルネックとなる多倍長整数演算の高速化に焦点を当てているものが多数を占めている．多倍長整数演算は，ナイーブな手法である筆算（schoolbook 法）の他に，以下のアルゴリズムが用いられている．

- CRT
- Cooley-Tukey 型 FFT
  – NTT
- Karatsuba 法

さらに，FFT は，その実装方法により，再帰型（recursive）と，反復型（iterative）の 2 種類に大別される．また，FFT による畳み込みの手法によって，巡回畳込み（cyclic convolution）と負巡回畳込み（negacyclic convolution）が存在する [25].

先行研究では，剰余演算についても高速化の試みがなされている．この剰余演算については，事前計算された値を用いた高速化が知られている．既存研究においては，Barrett 法 [26] と Montgomery 法 [27] が主に用いられていた．Barrett 法と Montgomery 法は，ともに事前計算された値を用いることによって，除算命令や剰余命令を用いることなく，高速に剰余演算を実行することができる．

5.1　GPU

GPU を用いた研究では，NVIDIA 製の GPU が用いられており，CUDA と呼ばれる GPU 上での汎用計算（GPGPU）用途のライブラリを用いて実装がされている．GPU は，安価で性能の低いコアを大量に搭載することで，並列処理性能を上げる構造となっている．そのため，いかに並列度を上げるかが，処理性能に影響する．また，大量のスレッドが同時に動作するため，GPU 上のメモリアクセス手法についても，性能に大きく影響する．

Wang ら [28] は，巨大な法上の乗算を，CRT を用いて小さい法ごとに分割し，並列に実行を行うことで，CPU 実装と比較して 273.6 倍の高速化を達成した．さらに，ストリームと呼ばれるパイプライン機構を用いて，データ転送と，処理実行の 2 つのストリームでパイプライン処理を行うことにより，データ転送時間を隠蔽している．Lupascu ら [10] は，CRT に加えて NTT を適用し，分割した法ごとに GPU を割り当てることによって，CPU 実装と比べ 8 倍の高速化を達成した．さらに，ストリームを，GPU へのデータ転送，処理の実行，GPU からのデータ受信のそれぞれに割り当てている．しかし，準同型暗号のパラメータが小さい場合，すなわち，多項式の次数が小さい場合は，GPU による高速化の恩恵を受けられないと報告している．

GPU の問題点としては，ハードウェアがネイティブで対応している最大データサイズが 8byte であるという点である．CPU 向けに実装されている準同型暗号ライブラリの多くは，多項式の係数として符号付き 64bit 整数を用いているため，乗算によるオーバーフローが発生する可能性がある．Lupascu ら [10] は，各係数どうしの乗算に Karatsuba 法を活用し，オーバーフローを検知できるように実装を行っている．

5.2　FPGA

FPGA における処理性能の観点として，レイテンシの他に，リソース消費量が挙げられる．FPGA は，機種によって LUT やレジスタの数が異なり，実装可能な回路の規模に制約が存在する．そのため，FPGA は実装においては，対象のアルゴリズムが消費するリソース量についても，最適化の対象となっている．

FPGA における多項式乗算の高速化は，パイプライン化によるスループット向上と，メモリアクセスの改善の 2 種類のアプリーチが見受けられる．Mert ら [16] は，4-step の NTT を採用し，各 step ごとをパイプラインステージとすることで，CPU 実装と比較して 21 倍の高速化を達成した．さらに，Iterative NTT を用いた場合に比べて，リソース消費量を削減することができると報告している．

FFT や NTT は複雑な処理であるため，FPGA 上のリソースを大幅に消費する．そこで，NTT の代替となる多項式乗算アルゴリズムとして，Karatsuba 法の活用が提案されている．Migliore ら [15] は，NTT と Karatsuba のそれぞれで準同型乗算のレイテンシとリソース消費量を比較し，レイテンシを 1.2 倍高速化，リソース消費量を半分に抑えることができたと報告している．

剰余演算や FFT，NTT では，各処理に共通して使用する値の事前計算を行い，あらかじめハードウェア上の BRAM に格納しておくことで，高速化を行うことが可能である [11–13, 16, 17]．一方，メモリ消費量を削減するために on-the-fly で計算するという手法 [14] も存在する．Roy ら [14] は，NTT で使用するひねり係数を事前計算するのではなく，on-the-fly で計算することで，メモリ消費量を削減する手法を提案している．

6　おわりに

本稿では，完全準同型暗号の高速化を目的とした，ハードウェアアクセラレータの利活用について研究調査を行った．本稿で示したように，ハードウェアアクセラレータを活用することによって，完全準同型暗号における処理性能を改善し，実用性を高めることができる．しかし，既存研究の多くは，単一種類のハードウェアアクセラレータを用いて実装を行っている．ハードウェアアクセラレータには，動作機構やメモリ帯域などの特性が存在し，異なる種類のハードウェアアクセラレータを組み合わせた場合の処理性能については検証の余地がある．したがって，今後は，複数種類のハードウェアアクセラレータを組み合わせた構成が，処理性能に与える影響について，検討を進めていきたいと考えている．ま

た，Bootstrapping の高速化についても検討を行い，完全準同型暗号の実用化に向けて研究を行っていく所存である．

謝辞

本研究は，JST CREST（JPMJCR1503）の支援を受けたものである．

参考文献

[1] 総務省. 情報通信白書平成 30 年版. https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/pdf/n3300000.pdf. accessed on June. 4. 2020.

[2] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, Vol. 51, No. 4, July 2018.

[3] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pp. 169–178, New York, NY, USA, January 2009. ACM.

[4] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pp. 129–148, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, p. 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[6] Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, pp. 25–43, Cham, 2016. Springer International Publishing.

[7] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pp. 34–54, Cham, 2019. Springer International Publishing.

[8] Hao Chen. Optimizing relinearization in circuits for homomorphic encryption. *CoRR*, Vol. abs/1711.06319, , 2017.

[9] P.V. Ananda Mohan. *Residue Number Systems - Theory and Applications*. Springer, 2016.

[10] C. Lupascu, M. Togan, and V. Patriciu. Acceleration techniques for fully-homomorphic encryption schemes. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, pp. 118–122, 2019.

[11] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, p. 1295–1309, New York, NY, USA, 2020. Association for Computing Machinery.

[12] F. Turan, S. S. Roy, and I. Verbauwhede. Heaws: An accelerator for homomorphic encryption on the amazon aws fpga. *IEEE Transactions on Computers*, pp. 1–1, 2020.

[13] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 387–398, 2019.

[14] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede. Hepcloud: An fpga-based multicore processor for fv somewhat homomorphic function evaluation. *IEEE Transactions on Computers*, Vol. 67, No. 11, pp. 1637–1650, 2018.

[15] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat. Hardware/software co-design of an accelerator for fv homomorphic encryption scheme using karatsuba algorithm. *IEEE Transactions on Computers*, Vol. 67, No. 3, pp. 335–347, 2018.

[16] A. C. Mert, E. Öztürk, and E. Savaş. Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 28, No. 2, pp. 353–362, 2020.

[17] A. C. Mert, E. Öztürk, and E. Savaş. Design and implementation of a fast and scalable ntt-based polynomial multiplier architecture. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pp. 253–260, 2019.

[18] Vincent Migliore, Cédric Seguin, Maria Méndez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, Guy Gogniat, and Russell Tessier. A high-speed accelerator for homomorphic encryption using the karatsuba algorithm. *ACM Trans. Embed. Comput. Syst.*, Vol. 16, No. 5s, September 2017.

[19] K. Millar, M. Łukowiak, and S. Radziszowski. Design of a flexible schönhage-strassen fft polynomial multiplier with high- level synthesis to accelerate he in the cloud. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–5, 2019.

[20] J. Ye and M. Shieh. Low-complexity vlsi design of large integer multipliers for fully homomorphic encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 26, No. 9, pp. 1727–1736, 2018.

[21] Jinjing Huang, Wei Chen, Zhixu Li, Pengpeng Zhao, Weiqing Wang, Hongzhi Yin, and Lei Zhao. Sgpm: a privacy protected approach of time-constrained graph pattern matching in cloud. *World Wide Web*, Vol. 23, No. 1, pp. 519–547, Jan 2020.

[22] Y. Wang, J. Lin, and Z. Wang. An efficient convolution core architecture for privacy-preserving deep learning. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018.

[23] D. Reis, M. T. Niemier, and X. S. Hu. A computing-in-memory engine for searching on homomorphically encrypted data. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, Vol. 5, No. 2, pp. 123–131, 2019.

[24] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the bfv homomorphic encryption scheme. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pp. 83–105, Cham, 2019. Springer International Publishing.

[25] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security*, pp. 124–139, Cham, 2016. Springer International Publishing.

[26] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*, pp. 311–323. Springer, 1986.

[27] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, Vol. 44, No. 170, pp. 519–521, 1985.

[28] W. Wang, Z. Chen, and X. Huang. Accelerating leveled fully homomorphic encryption using gpu. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2800–2803, 2014.