SCIS 2026 2026 Symposium on
Cryptography and Information Security
Hakodate, Japan, Jan. 26 – 30, 2026
The Institute of Electronics,
Information and Communication Engineers

準同型暗号 CKKS 方式における GPU を用いた複数暗号文演算の高速化
Acceleration of Multiple Ciphertext Operations Using GPU in the CKKS Homomorphic Encryption Scheme

鷹尾 直幸 * 服部 大地 † 若杉 飛鳥 † 小寺 雄太 *
Naoyuki Takao Daichi Hattori Asuka Wakasugi Yuta Kodera
野上 保之 *
Yasuyuki Nogami

Abstract— 準同型暗号は，暗号化されたデータに対して直接演算を行える暗号方式であり，データの機密性保持と利活用の両立を可能にする技術として注目されている．様々な方式が提案されている中で，CKKS 方式は実数を扱えることから，機械学習や統計解析などの分野での応用が期待されている．一方で，準同型暗号の欠点として計算コストが高いという問題があり，実用的な応用範囲は大きく制限されている．そのため，GPU などの並列計算基盤を活用した高速化手法の研究が進められてきた．しかし，Yang らによる Phantom-fhe ライブラリなどの GPU を用いた CKKS の既存実装は，単一暗号文の処理に焦点を当てており，複数暗号文を対象とした高速化が十分に検討されていない．そこで本稿では，Phantom-fhe ライブラリを基盤に，複数暗号文の演算における CPU と GPU の同期の待機時間の削減とデータコピーの最適化を図る方針を提案する．また，暗号文数を変化させた条件下で，提案方針の適用の有無による実行時間を比較評価し，その有効性を確認する．

キーワード 準同型暗号, GPU

1 序論

近年，機械学習の発展やクラウドコンピューティングの普及に伴い，データの利活用が活発になっている．具体的には，AI モデルの外部委託学習や IoT デバイスからのデータ収集も盛んに行われており，その結果，機微情報を含むデータが第三者に渡る機会も増加している．多用なデータの流通に伴い，その機密性を確保しつつ安全にデータを利用するための仕組みが不可欠となっている．準同型暗号は，この課題を解決する手法の一つとして注目されている暗号方式である．この方式は，暗号文に対して直接演算処理を行うことができ，その過程において復号を必要としない．よって，機械学習を含む各種演算を平文空間から暗号文空間へ移行することで，データの機密性保持と安全な利用の両立が可能となる．

AI モデルの学習アルゴリズムでは，行列演算が多く用いられ，行列の加算，アダマール積，行列積などが挙げられる．本稿で扱う準同型暗号方式では，単一の暗号文に複数のデータをまとめて格納することができる．しかし，大量のデータを平文とする場合には，複数個の暗号文が生成される．また，複数個の暗号文に対して演算を行う場合，単一の暗号文演算を繰り返し行う必要がある．一般に，準同型暗号では 1 回あたりの演算が平文に比べて高コストであり，そのような演算を多数回用いる行列演算では計算時間が大きく増加する．大規模行列を暗号化する場合，複数個の暗号文に分割して暗号化する必要がある．このとき，行列演算を行うために，複数暗号文に対して多数回の演算を実行する必要があり，計算時間が膨大になる．GPU を用いた並列処理による演算の高速化は，この問題の有効な解決策の 1 つである．本稿では，複数暗号文に対する 1 回あたりの演算を GPU を用いて高速化する．

1.1 先行研究

準同型暗号に対して，GPU を用いて高速化を行なった研究は多く知られている．Wang ら [12] は，Gentry[6]

* 岡山大学大学院 環境生命自然科学研究科, 〒 700-8530 岡山県岡山市北区津島中 3-1-1. Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, 3-1-1 Tsushima-Naka, Kita-ku, Okayama-shi, Okayama 700-8530, Japan.
† EAGLYS 株式会社, 〒 151-0051 東京都渋谷区千駄ヶ谷 5 丁目 27-3 やまとビル 7F. EAGLYS Inc., 7F Yamato building, 5-27-3 Sendagaya, Shibuya-ku, Tokyo, 151-0051, Japan.

の完全準同型暗号方式に対して，初めて GPU 実装を行った．また，Jung ら [7, 8] は，CKKS 方式に対して，初めて GPU 実装を行った．2023 年に Yang ら [13] によって複数の準同型暗号方式に対して GPU を用いた高速化手法が提案された．この手法は，Phantom-fhe というライブラリとして実装されており，GPLv3 ライセンスで公開されている．さらに，Özcan ら [10] によって，HEonGPU という FHE に関する GPU 実装ライブラリが公開されている．そして，Choi ら [4] によって，Phantom-fhe および HEonGPU よりも高速な GPU 実装設計が与えられ，Cheddar という OSS ライブラリが公開されている．

1.2 目的と構成

準同型暗号には，BGV[1]，BFV[5]，CKKS[3] などの様々な方式が提案されている．その中で，CKKS 方式は実数や複素数を平文として扱うことができ，他の方式と比べて多様なデータを処理できるため，本研究では CKKS 方式を対象とする．

GPU を用いた準同型暗号の既存の高速化手法は，単一暗号文同士の演算に焦点を当てており，複数暗号文を対象とした高速化については十分に検討されていない．

本稿では，GPU 上での複数の CKKS 暗号文演算の高速化手法について提案する．Phantom-fhe ライブラリを基盤として，複数暗号文に対する演算の実装を行い，その性能評価を通じて提案手法の有効性を検証する．

本稿は次のように構成される．第 1 章では，準同型暗号の GPU 実装について紹介した．第 2 章では，CKKS 方式と GPU について概説する．第 3 章では，Phantom-fhe ライブラリの概要を紹介する．第 4 章では，複数暗号文の演算に対する高速化手法について述べる．第 5 章では，実験結果を示し，得られた結果について考察する．最後に第 6 章で，結論を述べる．

1.3 貢献

本稿では，準同型暗号 CKKS 方式に対して，GPU を用いた複数暗号文演算の高速化について，Phantom-fhe ライブラリを基盤として行う．まず，複数暗号文演算の標準実装を，Phantom-fhe を基盤とした単一暗号文演算の処理フローを逐次的に行うものと定める．そして，標準実装に対して高速化を行う方法として，CPU と GPU の同期の待機時間の削減と GPU メモリ上でのデータコピーの最適化の 2 つの手法を提案する．標準実装と提案手法の実行時間を測定し，比較することで提案手法の有効性を示す．結果として，暗号文数 $d = \{10, 100, 1000\}$ の場合において，2 つの提案手法は，標準実装に比べていずれも高速であり，2 つの手法を組み合わせることでさらなる高速化が可能であることを確認した．また，2 つ目の手法によって，GPU メモリ上でのデータコピーを削減していることを確認した．以上より，CPU と GPU の同期の待機時間の削減および GPU メモリ上でのデータコピーの最適化は，今回の実験範囲において，高速化に寄与すると言える．

2 準備

2.1 CKKS

本節では，準同型暗号 CKKS 方式の概要を説明する．本稿では，分析対象データを cleartext と呼ぶ．また，暗号化対象データを plaintext と呼び，cleartext から plaintext への変換アルゴリズムを encode，その逆変換を decode と定める．

本稿では, Slot 方式による encode を対象とする．Slot 方式とは，cleartext に対して，逆フーリエ変換アルゴリズムを適用することにより，多項式として表される plaintext を生成する encode 方式である．plaintext 空間は，整数係数多項式環に対する円分多項式による剰余環によって定められ，本稿では，その多項式の長さを N とする．

秘密鍵を sk，公開鍵を pk とする．plaintext p に対して暗号化すると，暗号文

$$c = (a, b) = \mathrm{Encrypt}_{\mathrm{pk}}(p)$$

が得られ，これは長さ N の整数係数多項式の組で表される．このとき，c に対して sk による復号を行うと，$\mathrm{Decrypt}_{\mathrm{sk}}(c) \approx p$ が成り立つ．

以下では，p_1, p_2 を plaintext として，

$$c_1 = (a_1, b_1) = \mathrm{Encrypt}_{\mathrm{pk}}(p_1)$$
$$c_2 = (a_2, b_2) = \mathrm{Encrypt}_{\mathrm{pk}}(p_2)$$

と定める．暗号文に対する加算演算は，$c_1 + c_2 = (a_1 + a_2, b_1 + b_2)$ によって定められる．

次に，暗号文の乗算演算について説明する．CKKS 方式において，暗号文の乗算には評価鍵が必要である．暗号文 c_1, c_2 に対して，多項式乗算により，3 つの多項式からなる暗号文

$$c = (d_0, d_1, d_2) = (a_1 a_2, a_1 b_2 + a_2 b_1, b_1 b_2)$$

が得られる．c は，c_1, c_2 と要素数が異なるため，評価鍵を用いて暗号文の長さを 2 へ戻す必要がある．この操作を再線型化と呼ぶ．さらに，CKKS 方式では，再線型化後に rescale と呼ばれる処理を適用する．再線型化および rescale の詳細については，[3] を参照されたい．

2.2 GPU

GPU は多数の演算ユニットを備えた, 高い並列計算能力を持つプロセッサである．近年は機械学習やブロック

チェーンなど，計算集約型の分野で広く利用されている [11]．GPU は複数の Streaming Multiprocessor（SM）によって構成され，各 SM は多数の演算コアに加えて共有メモリを備えている．

thread とは，GPU における最小の実行単位のことである．一方，GPU 上での処理実行時には，thread は warp と呼ばれる，32 個の thread からなるグループにまとめられて実行される [9]．warp 内の thread は，同一の命令を並列に実行する SIMT（Single Instruction, Multiple Threads）方式で動作する．各 SM には複数の warp スケジューラが備えられており，複数の warp を高速に切り替えながら発行することで演算ユニットの稼働率を高め，高いスループットを実現する．

thread block とは，複数の thread をまとめたものであり，grid とは，複数の thread block をまとめたものである．GPU では，thread，thread block，grid といった階層構造によって並列処理が組織化される．thread block に属する thread は通常同一の SM に割り当てられ，共有メモリなどのローカルな資源を利用しながら協調して演算を実行する．さらに，GPU 上での処理実行時には，grid を構成する thread block が順次 SM にスケジューリングされる．このような thread，thread block，grid の階層構造を用いることで，GPU は膨大な数の thread を体系的に管理し，同種の計算を大規模データに対して効率的かつ高スループットに適用することが可能となる．

CPU と GPU はどちらも汎用プロセッサであるが，設計思想は大きく異なる [2]．ここで，汎用プロセッサとは，特定用途に限定されず，プログラムを変更することで多用な種類の計算を実行できるプロセッサのことである．CPU は少数の高性能演算コアと大容量キャッシュを備え，分岐処理や逐次的な処理に適している．一方，GPU は制御回路やキャッシュを最小限に抑え，多数の演算コアに割り当てる構造を採用している．そのため，GPU は複雑な制御を必要としない同一パターンの計算を，多数のデータに対して同時に実行する処理において CPU と比較して高い性能を発揮する．特に，GPU のアーキテクチャは，行列演算や多項式演算といった大規模並列処理において有効である．

GPU プログラミングでは，CPU をホスト，GPU をデバイスとして，二者間で処理が行われる．カーネル関数とは，デバイス上で実行される関数である．カーネル関数は，thread block 数や thread 数といった実行構成を指定して呼び出され，デバイスで並列に実行される．GPU プログラミングにおける典型的な実行フローを以下に示す．なお，H はホスト，D はデバイスを表す．

1. デバイスメモリ確保呼び出し
2. データコピー呼び出し（H to D）
3. カーネル関数呼び出し
4. データコピー呼び出し（D to H）
5. ホストとデバイスの同期
6. デバイスメモリ解放呼び出し

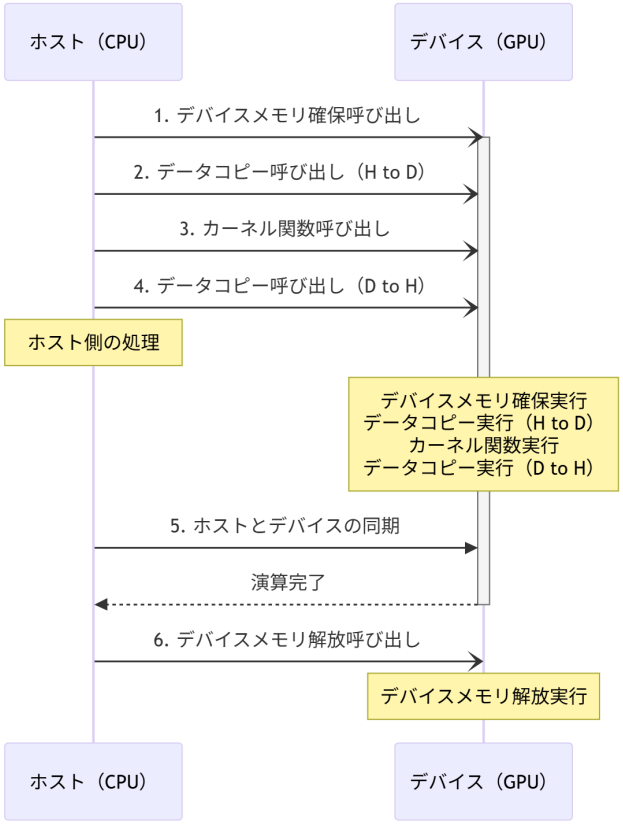

図 1: GPU プログラミングにおけるホストとデバイスの処理フロー

基本的にはステップ 1 から 4 において，ホストとデバイスは非同期となる．そのため，ステップ 4 の後，ホストはデバイスのデータコピー実行やカーネル関数実行の完了を待つことなく，ホスト側の処理を行うことができる．一方，ホストがデバイスの処理結果を参照する場合は，ステップ 5 のように同期を行い，デバイスでの処理の完了まで待機する必要がある．

以上の手順をシーケンス図として**図** 1 に示す．なお，カーネル関数の呼び出し後，ホストはデバイスの演算と独立して他の処理を実行できる．

3 Phantom-fhe

3.1 Phantom-fhe について

Phantom-fhe [13] は，GPU を用いて準同型暗号演算を高速化するために Yang らによって開発されたライブ

ラリである．Phantom-fhe の特徴的な設計として，暗号文データをデバイスメモリ上に保持し続けることで，ホストとデバイス間のデータコピーを最小化する点が挙げられる．

通常 GPU プログラミングでは，**図** 1 に示すように，各カーネル関数呼び出しに対して，ステップ 1 からステップ 6 までの一連の処理を行う．一方 Phantom-fhe では，演算内容に合わせて，いくつかの処理フローを削減している．具体的には，encode ではステップ 1-5 が実行され，ホストメモリの cleartext を plaintext へ変換し，plaintext をデバイスメモリ上に配置する．その後，加算や乗算などを含む通常の暗号文演算ではステップ 3 のカーネル関数呼び出しのみを行う．decode ではステップ 3-6 が実行され，デバイスメモリ上の plaintext を cleartext へ変換し，cleartext をホストメモリに配置する．encode および decode では，データコピー呼び出しの後に同期処理を行う．この設計により，ステップ 2, 4, 5 が削減されるため，GPU の演算性能をカーネル関数の処理に集中させることができる．

3.2 Phantom-fhe における単一暗号文演算

本稿では，CKKS 方式における暗号文同士の加算と乗算に焦点を当て，使用者は単一暗号文演算のみを行うことを想定する．Phantom-fhe では，暗号文演算を通常非同期で実行するが，本稿では，Phantom-fhe における単一暗号文演算の処理を，カーネル関数呼び出し・カーネル関数実行・ホストとデバイスの同期によって構成されるものと定める．なお，暗号文乗算では，データコピー呼び出し（D to D）およびデータコピー実行（D to D）を必要とする．乗算の処理フローの詳細は以下のように与えられる．

1. **データコピー呼び出し（D to D）**

   乗算の際に必要になるステップで，ホストからデバイスへの命令発行を示し，非同期に行われる．そのため，ホストはデータコピー呼び出し後直ちに次の処理を進めることができる．

2. **データコピー実行（D to D）**

   乗算の際に必要になるステップで，デバイスメモリ上で暗号文のデータコピーを実行する．CKKS 方式における暗号文は通常 2 つの多項式で表されるが，暗号文同士の乗算を行うと，結果の暗号文は 3 つの多項式で表される．Phantom-fhe では，暗号文のデータはデバイスメモリ上に保持されるため，乗算の結果を格納するために新たなデバイスメモリ領域を確保し，元の暗号文データをコピーする必要がある．

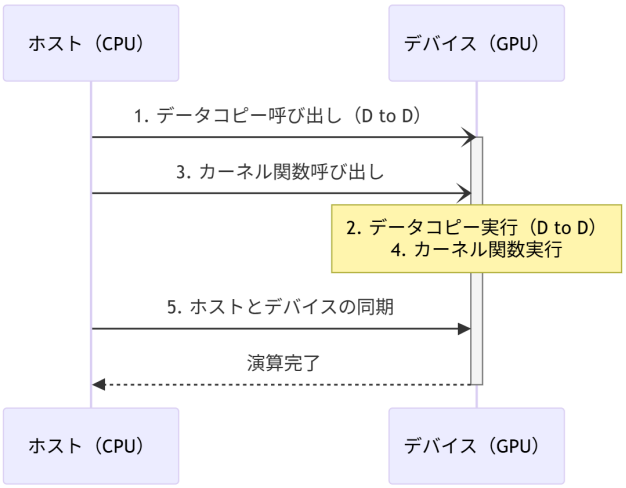

図 2: Phantom-fhe における単一暗号文演算の処理フロー

3. **カーネル関数呼び出し**

   ホストからデバイスへの命令発行を示し，非同期に行われる．そのため，ホストはカーネル関数呼び出し後直ちに次の処理を進めることができる．

4. **カーネル関数実行**

   デバイス上で実行される暗号文演算の実体である．Phantom-fhe では，暗号文同士の加算および乗算に対応するカーネル関数が実装されており，これらは GPU の並列計算能力を活用して効率的に演算を実行する．

5. **ホストとデバイスの同期**

   ホストがデバイスでの処理完了を待機するために使用される．

また，Phantom-fhe における単一暗号文演算の処理フローのシーケンス図を**図** 2 に示す．

次に，Phantom-fhe における単一暗号文の演算 1 回あたりの，各ステップの実行時間を測定した．実験環境および使用した CKKS 方式の暗号文パラメータは，**表** 1 および**表** 2 に示す．なお，秘密鍵の分布は ternary であり，パラメータは 128 bit security を満たす．加算については `phantom::add_inplace` 関数，乗算は `phantom::multiply_inplace` 関数を使用した．測定には NVIDIA が提供している Nsight Systems を用い，初回実行時のキャッシュの影響を排除するため，事前にウォームアップ実行を 5 回行った．その後，各演算を 10 回ずつ実行し，ステップごとの実行時間の平均値を求めた．単一暗号文の加算および乗算を実行した場合の，各ステップにかかる実行時間および実行回数を，**表** 3 に示

表 1: 実験環境

項目	詳細
OS	Ubuntu 22.04.4 LTS
CPU	Intel Core i9-11900K
GPU	NVIDIA GeForce RTX 3090
RAM	64GB
CUDA	12.4
Nsight Systems	2025.1.1

表 2: CKKS 方式の暗号文パラメータ

パラメータ	値
polynomial modulus degree N	8192
coefficients modulus	{60, 40, 60}
scale	2^{40}
noise standard deviation	3.19

す．なお，表中の「—」は該当するステップが実行されなかったことを示す．

表 3 において，加算のステップ 3 およびステップ 4 がそれぞれ 2 回ずつ行われているのは，CKKS 方式における暗号文は通常 2 つの多項式で構成され，Phantom-fhe の加算実装では，それぞれに対してカーネル関数を呼び出し・実行しているためである．一方，乗算では，複数の多項式演算を 1 回のカーネル関数呼び出し・実行で処理しているため，ステップ 3 およびステップ 4 はそれぞれ 1 回ずつとなっている．また，ステップ 1 およびステップ 2 は乗算にのみ必要なステップであり，加算では実行されていない．

なお，**表 3** に示している各ステップと Nsight Systems 上での表記の対応は，**表 4** のようになっている．

4 Phantom-fhe を用いた複数暗号文演算

本章では，Phantom-fhe における複数暗号文演算について，複数の処理方法を説明する．まずは標準実装を示し，続いて，標準実装からいくつかのステップを最適化した 2 種類の方法を提案する．また，それぞれの最適化方法に対して，標準実装との差異を示し，その実装方法を示す．

4.1 標準実装

本稿では，複数暗号文演算の標準実装の処理フローとは，単一暗号文の処理フローを逐次的に暗号文の数だけ繰り返すものと定める．暗号文の数が d 個の場合の標準実装の処理フローを**図 3** に示す．

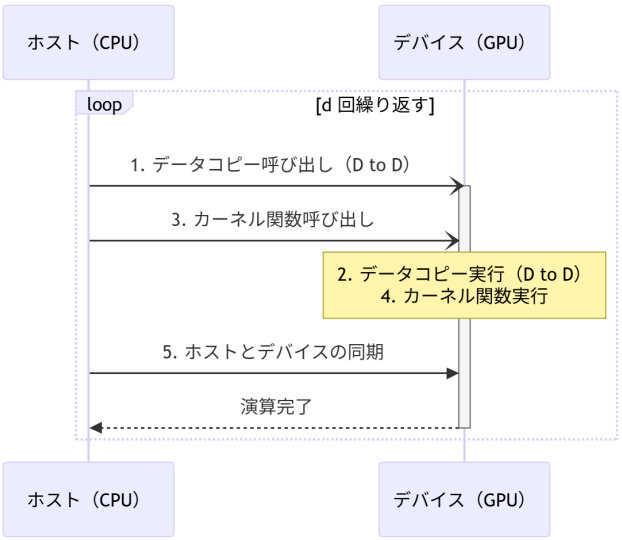

図 3: 複数暗号文演算の標準実装の処理フロー

4.2 方法 1: ホストとデバイスの同期の待機時間の削減

標準実装では，**図 3** に示したように，各暗号文の演算ごとにホストとデバイスの同期を行う．そのため，ある暗号文に対する演算処理が全て終了してはじめて，その次の暗号文に対する演算処理が行われる．

これに対して，方法 1 では，各暗号文の演算処理フロー内でホストとデバイスの同期を行うのではなく，最後の暗号文のカーネル関数呼び出し後の一度だけ同期を行う．そうすることで，ある暗号文に対する処理が終了する前に，その次の暗号文に対する演算処理を行うことができ，ホストとデバイスの同期の待機時間を削減できる．d 個の暗号文に適用した場合の方法 1 の処理フローを**図 4** に示す．

方法 1 の実装は，ホストとデバイスの同期を最後の一度のみ残し，残りの同期を削除することで達成される．

4.3 方法 2: デバイスメモリ上でのデータコピーの最適化

標準実装では，**図 3** に示したように，データコピー呼び出しとデータコピー実行が必ず発生する．**表 3** より，データコピー呼び出しおよびデータコピー実行に必要な実行時間は，カーネル関数呼び出しおよびカーネル関数実行と比較して無視できない．

本稿では，暗号文のデータコピー処理が，これらの実行時間の多くを占めていると仮定する．暗号文のデータコピーを乗算処理フローから削除することで，**表 3** 中のステップ 1 および 2 の実行時間を削減できる．

CKKS 暗号文は通常 2 つの多項式で表されるが，暗号文乗算を行うと，暗号文は一時的に 3 つの多項式で表される．そのため，標準実装では，3 つの多項式を格納

表 3: Phantom-fhe における単一暗号文演算の各ステップにかかる実行時間および呼び出し回数

	加算		乗算	
ステップ	実行時間 [μs]	回数	実行時間 [μs]	回数
1. データコピー呼び出し（D to D）	—	0	2.5	1
2. データコピー実行（D to D）	—	0	1.5	1
3. カーネル関数呼び出し	1.7	2	2.0	1
4. カーネル関数実行	1.6	2	2.2	1
5. ホストとデバイスの同期	4.5	1	5.3	1

表 4: Phantom-fhe における単一暗号文演算の各ステップと Nsight Systems 上での表記の対応

ステップ	Nsight Systems 上での表記
1. データコピー呼び出し（D to D）	`cudaMemcpyAsync_ptsz`
2. データコピー実行（D to D）	`Memcpy D to D`
3, 4. カーネル関数呼び出し・実行（加算）	`add_rns_poly`
3, 4. カーネル関数呼び出し・実行（乗算）	`tensor_prod_2x2_poly`
5. ホストとデバイスの同期	`cudaDeviceSynchronize`

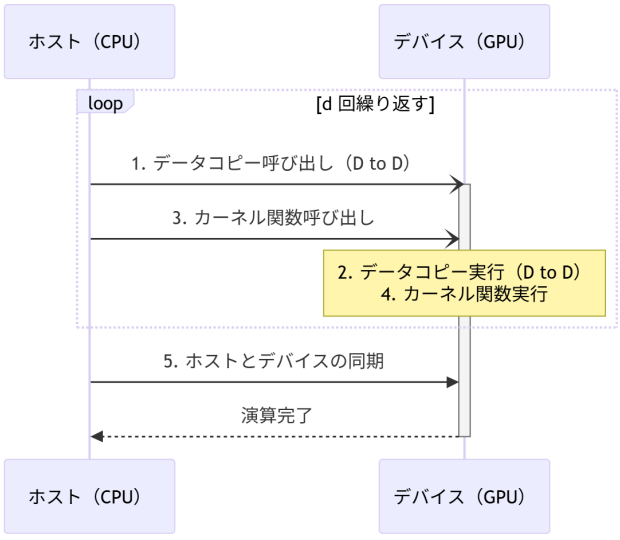

図 4: 方法 1 を適用した場合の複数暗号文演算の処理フロー

するためのデバイスメモリ領域を確保し，それに対して入力の暗号文データをコピーしている．方法 2 では，デバイスメモリ上でのデータコピーを最適化し，標準実装から入力の暗号文データのコピー処理を削除する．方法 2 の実装は，乗算フロー実行前に，デバイスメモリ上にデータを持たない暗号文を用意し，その暗号文を処理フローに与える．処理フローでは，データコピー呼び出しの前に，与えられた暗号文に対して 3 つの多項式を格納するためのデバイスメモリ領域を確保し，カーネル関数でその領域に乗算結果を代入することで達成される．

5 高速化の検証

本章では，前章で述べた最適化方法の効果を評価するために，各方法を適用した場合の実行時間を測定し比較する．実験環境および使用した CKKS 方式の暗号文パラメータについては，**表 1** および**表 2** を参照されたい．

d 個の暗号文を持つリストに対して，次の 4 つの場合，標準実装，方法 1，方法 2，そして方法 1 と方法 2 を組み合わせた場合において，それぞれを適用した複数個の暗号文加算および乗算を実行する．実行時間とは，複数暗号文演算の処理の開始から終了までの時間と定める．$d = \{10, 100, 1000\}$ として，各場合の試行を 1000 回測定し，その実行時間の平均値を算出する．

表 5 は d 個の暗号文加算の実行時間の平均値を示し，**表 6** は乗算の実行時間の平均値を表す．なお，方法 2 は乗算に対して適用するため，加算は標準実装と方法 1 の 2 つの場合のみを測定する．

また，方法 2 によって，各暗号文に対する乗算のデー

表 5: 複数暗号文の加算における実行時間

暗号文の数	10	100	1000
標準実装 [μs]	78.48	817.9	8069.4
方法 1[μs]	45.73	500.9	4973.7

表 6: 複数暗号文の乗算における実行時間

暗号文の数	10	100	1000
標準実装 [μs]	103.4	1004.0	10141.5
方法 1[μs]	61.67	592.9	5873.5
方法 2[μs]	70.34	699.0	6877.4
方法 1+方法 2[μs]	37.63	338.8	3297.1

タコピー呼び出しとデータコピー実行の実行時間の削減を達成できるかを確認する．**表 3** と同様の実験を方法 2 に対して行い，**表 7** は各ステップにおける実行時間を示す．また，標準実装における，各暗号文の各ステップにかかる実行時間および実行回数は，**表 3** と同様であり，**表 7** に再度示す．

表 5 および**表 6** より，方法 1 を適用した加算や乗算は標準実装に比べて高速であることを確認した．よって，方法 1 によるホストとデバイスの同期の待機時間の削減は，今回の実験範囲において高速化を達成したと言える．

さらに，**表 6** より，方法 2 を適用した乗算は，標準実装の乗算に比べて高速であることを確認した．方法 2 により，データコピー呼び出しの実行時間を削減し，特に，データコピー実行の実行回数を 0 にしていることを確認した．よって，方法 2 によるデバイスメモリ上でのデータコピーの最適化も，今回の実験範囲において高速化を達成したと言える．

最後に，複数暗号文における乗算に対して，方法 1 と方法 2 を同時に適用した場合が最も高速であることを確認した．

6 まとめ

本稿では，準同型暗号 CKKS 方式における GPU を用いた複数暗号文演算の高速化を Phantom-fhe ライブラリを基盤に行った．複数暗号文演算の標準実装を，Phantom-fhe を基盤とした単一暗号文演算の処理フローを逐次的に行うものと定め，それに対して高速化を行う方法を 2 つ提案した．方法 1 ではホストとデバイスの同期の待機時間の削減を行い，方法 2 ではデバイスメモリ上でのデータコピーの最適化を行った．

本稿では，提案手法を実装し，暗号文数 $d = \{10, 100, 1000\}$ の場合における実行時間を測定した．その結果，標準実装に比べていずれの手法も高速であり，方法 1 と方法 2 を組み合わせることでさらなる高速化が可能であることを確認した．また，方法 2 の処理フローにおける各ステップの実行時間を測定し，標準実装と比較した．結果として，方法 2 により，データコピー呼び出しの実行時間を削減し，データコピー実行の実行回数を 0 にしていることを確認した．以上より，ホストとデバイスの同期の待機時間の削減およびデバイスメモリ上でのデータコピーの最適化は，今回の実験範囲において，高速化に寄与すると言える．

今後の展望として，カーネル関数呼び出しの回数削減も有効な手法である可能性がある．**表 3** に示したように，カーネル関数呼び出しに必要な実行時間は，カーネル関数実行の実行時間と比較して無視できない．今後はこの課題を解決するために，一度のカーネル関数呼び出しで，複数の暗号文演算を実行する手法を考察したい．また，一度のカーネル関数呼び出しで，いくつかの暗号文演算を並列に実行することを検討している．これを実現する手法として，1 つの暗号文の処理に使用する thread 数を削減することが挙げられる．本手法を実装することで，いくつかの暗号文演算を並列に実行し，カーネル関数呼び出し回数の削減を図ることができると考えている．

参考文献

[1] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014), pp. 1–36.

[2] Robert Chab, Fei Li, and Sanjeev Setia. “Algorithmic Techniques for GPU Scheduling: A Comprehensive Survey”. In: *Algorithms* 18.7 (2025), p. 385.

[3] Jung Hee Cheon et al. “Homomorphic encryption for arithmetic of approximate numbers”. In: *International conference on the theory and application of cryptology and information security*. Springer. 2017, pp. 409–437.

[4] Wonseok Choi, Jongmin Kim, and Jung Ho Ahn. “Cheddar: A Swift Fully Homomorphic Encryption Library Designed for GPU Architectures”. In: *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 2026, pp. 35–49.

[5] Junfeng Fan and Frederik Vercauteren. “Somewhat practical fully homomorphic encryption”. In: *Cryptology ePrint Archive* (2012).

表 7: 乗算の各ステップの実行時間および呼び出し回数

	標準実装		方法 2	
ステップ	実行時間 [μs]	回数	実行時間 [μs]	回数
1. データコピー呼び出し（D to D）	2.5	1	0.055	1
2. データコピー実行（D to D）	1.5	1	—	0
3. カーネル関数呼び出し	2.0	1	1.8	1
4. カーネル関数実行	2.2	1	2.2	1
5. ホストとデバイスの同期	5.3	1	5.5	1

[6] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.

[7] Wonkyung Jung et al. “Accelerating fully homomorphic encryption through architecture-centric analysis and optimization”. In: *IEEE Access* 9 (2021), pp. 98772–98789.

[8] Wonkyung Jung et al. “Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 114–148.

[9] NVIDIA. *CUDA C++ Programming Guide*. Online. Version 13.0. 2025. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/ (visited on 12/02/2025).

[10] Ali Şah Özcan and Erkay Savaş. “Heongpu: a gpu-based fully homomorphic encryption library 1.0”. In: *Cryptology ePrint Archive* (2024).

[11] Cristina Silvano et al. “A survey on deep learning hardware accelerators for heterogeneous hpc platforms”. In: *ACM Computing Surveys* 57.11 (2025), pp. 1–39.

[12] Wei Wang et al. “Accelerating fully homomorphic encryption using GPU”. In: *2012 IEEE conference on high performance extreme computing*. IEEE. 2012, pp. 1–5.

[13] Hao Yang et al. “Phantom: A CUDA-Accelerated Word-Wise Homomorphic Encryption Library”. In: *IEEE Trans. Dependable Secur. Comput.* 21.5 (2024), pp. 4895–4906. DOI: 10.1109/TDSC.2024.3363900. URL: https://doi.org/10.1109/TDSC.2024.3363900.