完全準同型暗号を用いたゲノム秘匿情報検索アプリケーションにおける Persistent Memory および低レイテンシ SSD の比較調査

廣江　彩乃†　　小口　正人†

† お茶の水女子大学大学院人間文化創成科学研究科　〒 112–8610 東京都文京区大塚 2-1-1
E-mail: †{ayano-h,oguchi}@ogl.is.ocha.ac.jp

あらまし　暗号文のままデータを加算乗算できる完全準同型暗号は，秘匿データを扱う際やクラウドなどの外部演算資源を使用する際に，データの盗聴対策として大変有用である．その一方，演算量が多く，大量のメインメモリを必要とするため実行時間が非常にかかる問題点がある．そこで本研究では，近年注目が集まっている Storage Class Memory(SCM) の活用を考え，メインメモリと同様に扱える不揮発メモリ (Persistent Memory) と低レイテンシ SSD を用いて性能評価を行う．本研究では，完全準同型暗号を用いたゲノム秘匿情報検索アプリケーションに対して，不揮発メモリと低レイテンシ SSD の違いを考慮しつつ，実験結果を考察する．

キーワード　セキュリティ，不揮発メモリ，SCM，性能評価

1　はじめに

近年個人情報を含むビッグデータを扱う処理をクラウドコンピュータに委託する取り組みが増えている．企業などの様々な組織は顧客から個人情報や技術情報などの機密情報を保持しており，解析技術の発展に伴ってこれらのデータを利用することでさらに有用なデータを得ることができる．そのためには，膨大なデータから有益な情報や特徴的なパターンを抽出することができるような処理能力が高い計算機での演算が必要となる．そこで各企業が保有する膨大なデータをクラウドやデータセンタなどの大型の計算機とストレージを保持する機関に委託し，その機器に対して利用者が問い合わせをすることで統計処理を行っていくことが考えられる．一方で，クラウドコンピュータ上でこれらのデータから解析結果を得る際には処理依頼データの送信から演算途中，そして演算結果を送信するまでの過程においてデータ漏洩リスクがある．例えばゲノムデータなどの生体情報や医薬品の開発における技術情報は利用価値が高く盗聴対象となりやすい．このように秘匿情報を扱う際には強固なセキュリティ管理が必要不可欠となる．そこで有用と考えられているのが，データを秘匿したまま処理を行うことができる完全準同型暗号 [1] である．完全準同型暗号とは暗号文同士の加法と乗算が成立する暗号手法であり，2 章で説明する．

クライアント・サーバ構成の秘匿データを扱うシステムに使用する暗号方式には従来の共通鍵暗号などによる暗号化手法も考えられる．しかし暗号化したデータ同士の複雑な演算を行うためにはクライアントの秘密鍵をサーバ側に渡す必要がある．この際にはサーバ側でその秘密鍵を使って暗号化されて送られてきたデータを復号して平文のデータに戻して処理を行う必要があり，この際の盗聴リスクがある．

以上の理由から盗聴のリスクが低いのが完全準同型暗号であるが、この暗号化手法にはコンピュータリソースへの負荷が大きすぎるという問題がある．アルゴリズムの高速化は進められているものの，依然として演算量が多いためである．それに加えて完全準同型暗号を用いて暗号化するとデータの大きさは元の大きさより更に膨大になる [2]．その結果高価なメインメモリが大量に必要になったり，演算中にメインメモリが不足して swap 処理が発生し，処理速度がメインメモリに遥かに劣る HDD などのストレージ領域を用いる必要が出てきたりする．クラウドコンピュータを使用することを考慮すると特に，用いるコンピュータリソースによって費用がかなり左右される．そのため上で述べたように完全準同型暗号などを用いて秘匿する必要があるビッグデータを使うアプリケーションをメインメモリ上のみで実行することは費用面で現実的ではなく，実行に際して分散するストレージ領域を用いることが想定される．そこで本研究では，近年開発が進む不揮発メモリや高性能な SSD などを活用することを考える．異なる特徴を持つ記憶装置を秘匿検索処理の際に用いて，その際の実行時間の比較・評価を行う．そして完全準同型暗号の実用化に向けた高性能記憶装置の有効利用について考察する．

2　完全準同型暗号

2.1　特　　徴

式 (1) が示すように暗号文同士での加算が成立する性質を加法準同型性，また式 (2) のように暗号文同士での乗算が成立する性質を乗法準同型性という．

加法準同型性・乗法準同型性

$$\mathrm{Encrypt}(m) \oplus \mathrm{Encrypt}(n) = \mathrm{Encrypt}(m + n) \quad (1)$$

$$\mathrm{Encrypt}(m) \otimes \mathrm{Encrypt}(n) = \mathrm{Encrypt}(m \times n) \quad (2)$$

完全準同型暗号 FHE はこの両方の性質を持ち合わせた暗号化手法である．FHE を用いることで，平文上で行うのと同様に

暗号文同士での加法演算・乗法演算を行うことが出来る．完全準同型暗号は公開鍵暗号方式の機能を持つため，秘密鍵を用いることなく暗号文同士の演算から平文同士の演算を暗号化した値を導くことが可能となる．そのため，ゲノム秘匿検索に完全準同型暗号を適用することで，秘密鍵を渡すことなくサーバがデータ同士の処理を行えると期待できる [3].

2.2 暗号手法の課題

完全準同型暗号の概念自体は，公開鍵暗号が考案された当初の 1978 年に Rivest ら [4] によって提唱され，2009 年に Gentry [1] が実現手法を提案した．提案当時は計算量の大きさから実用性が乏しかったが，その後も様々な研究によって高速化や改良が進められ，簡単な計算であれば十分な性能レベルを示している [2].

各暗号文には，暗号の解読困難性を高めるため，ランダムなノイズが付加されている．完全準同型暗号処理の課題として，計算量が大きいことに加えて，暗号文に含まれるノイズが演算の度に増加し，閾値を越えると復号することが不可能になるということが挙げられる．特に乗算を行った際のノイズの増加が著しいため，暗号文に対する乗算操作の演算回数を限定した制限付き準同型暗号， Somewhat Homomorphic Encryption (SHE, SwHE) が考案され，処理速度の改善に成功している．しかし制限付き準同型暗号では，演算回数が制限されるため，実装できる機能が限られて汎用性に欠ける. [5]

また，bootstrap と呼ばれるノイズをリセットする手法の導入を行うことで，演算回数が限られてしまうことは解決される．bootstrap 処理とは，暗号文のノイズを初期値に近い量に減少させ，暗号化した状態での計算を理論上何度でも可能にする手法である．しかし，実際にはこの処理も計算量が大きく，依然として計算量の大きさの問題は残る. [6]

2.3 完全準同型暗号ライブラリ

暗号スキームの研究が進むに連れて，多くのオープンソースの暗号ライブラリが幅広い用途に向けてオンラインで公開されるようになった．例えば HElib [7] は，初期に公開されたよく知られているライブラリの一つである．IBM の研究者らによって C++で実装され，ノイズをリセットする bootstrap をサポートしている．本研究でも HElib を用いて暗号化アプリケーションを実行する．

他にも，PALISADE [8] や SEAL [9] などがよく知られており，この二つもどちらも C++で実装されている．これらは HElib と違って bootstrap 処理が無かったり，HElib は外部のライブラリに依存しているのに対して，PALISADE や SEAL は外部ライブラリに依存していないという点が異なる点である．

3 ストレージデバイス

3.1 ストレージデバイスの現状

近年開発が進む記憶装置の一つとしてストレージクラスメモリや高性能 SSD がある．これらメモリの階層は図 1 が示す通りである [10].

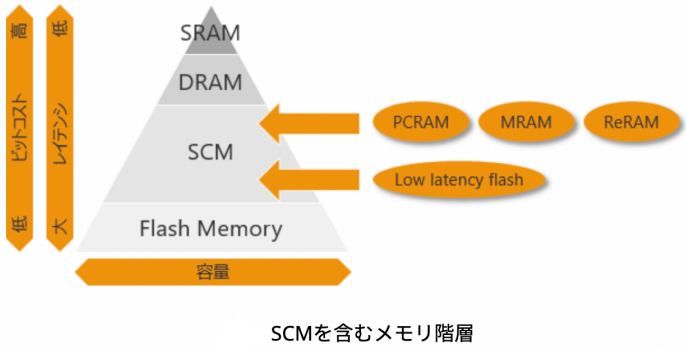

図 1 メモリ階層

高性能なメモリの研究開発が進む背景が以下だ．
5G の伸長などにあわせて，生成されるデータが爆発的に増えていき，生成されたデータは機械学習や AI といった様々なアプリケーションで CPU が処理するためにデータベース化されストレージに保存されている。現在のコンピューティングシステムでは，データを処理するためにデータが保存されているストレージ空間から CPU がデータを処理するためのメモリ空間へのデータの転送やデータの入れ替えが発生する．しかし，メモリ空間のデバイスである DRAM とストレージ空間のデバイスである フラッシュメモリを用いた SSD の間に大きな性能ギャップがあり，データの処理効率が良いとは言えない．そのため、データの入れ替えを少なくするために，主記憶容量の拡張と，データの転送を高速にするためのストレージ容量の低レイテンシ化が求められている．その DRAM とフラッシュメモリの間の性能・容量ギャップを埋めるために、SCM(ストレージクラスメモリ) と呼ばれる階層が考案され、様々なデバイス候補を各社が開発している．

3.2 ストレージクラスメモリ/低遅延 SSD

ストレージクラスメモリとは，メインメモリとストレージとの間の性能・コストの差を埋めるメモリの総称である．その特徴は，DRAM などのメモリよりも大容量で，SSD などのストレージよりも読み書き速度が速い不揮発性のメモリである，という点である．メインメモリのように処理が高速で、ビット単価が DRAM よりも安い不揮発性のメモリを作ろう，ということで考案された．メインメモリとストレージの長所を盛り込んだ形だ．最初にこの言葉が使われたのは，IBM が発表した論文 [11] である．その後，東芝メモリを前身にもつキオクシアや Intel がストレージクラスメモリの開発を進めている．ストレージクラスメモリは階層の名前であり，近年開発が進む低遅延 SSD や PCRAM(相変化メモリ) や MRAM(磁気抵抗変化メモリ) などの次世代不揮発性メモリを含む様々な記憶装置がこれにあたる．
低遅延 SSD は，低遅延フラッシュメモリとも呼ばれ，メモリに比べるとアクセスに時間がかかるフラッシュメモリを高速化したものである．低遅延 SSD は，ストレージクラスメモリに比べるとかなり製品化が進んでいる．キオクシアや Intel,

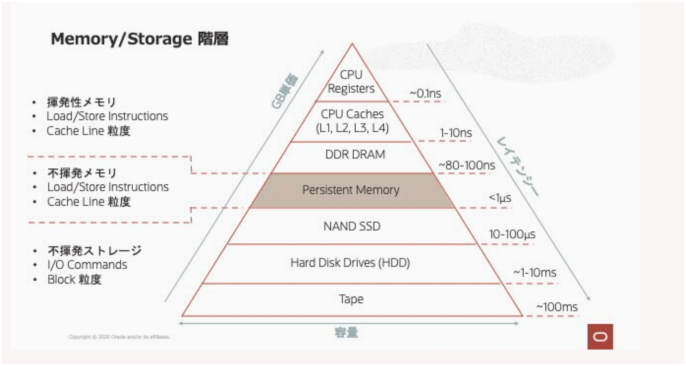

図 2　メモリ階層における不揮発メモリの位置付け

Samsung などの各大手メーカが低遅延フラッシュとして，多層の 3D NAND Flash メモリ・チップを発表している．3D NAND チップとは，従来の 2DNAND チップがすでに限界まで高密度になっていることから開発された．3D NAND チップは，メモリセルを並べたレイヤーを積み上げることによって，構成されている [12]．本研究では，まずは高性能 SSD を用いて評価実験を行う．今回実験で用いる SSD についてはそれぞれの特徴は，実験環境の章にまとめる．

3.3　不揮発メモリ

ここで不揮発メモリ について紹介する [13]．不揮発メモリとは，永続メモリや Persistent Memory とも呼ばれる．その名の通り不揮発性であるために情報を保持しておくことができるメモリということである．従来，システムメモリとは揮発性でありバイトアドレス指定可能なものであった．一方で HDD や SSD といったストレージについては，不揮発性でデータを保持しておくことが可能であり，こちらはブロック指向であった．不揮発性で，バイトアドレス指定が可能という特徴を持つ不揮発メモリの登場によって，そのメモリとストレージの境界が曖昧になった．図 2 にメモリ階層における不揮発メモリの位置付けを示す．ここで気になるのが不揮発メモリの速度であるが，DIMM(Dual Inline Memory Module) の規格で従来の揮発性 DRAM と一緒に動作させるのに十分な速度を出すことができる．また，不揮発メモリに搭載されているのは 3D XPoint メモリである．対して，従来のストレージ製品で最もよく使われているのは NAND Flash メモリなどフラッシュベースの不揮発性メモリである．そしてそれらは，NAND 論理ゲートにちなんで NAND Flash，もしくは単にフラッシュと呼ばれる．これは主にメモリカードや USB フラッシュドライブ，また汎用ストレージ用の SSD で用いられる．不揮発メモリにおいて，3D XPoint はメモリ DIMM としても Intel Optane SSD としてブランド化された SSD 形式でも利用することができる．見た目はメインメモリと相違なく，実際にメインメモリと並べてバスに挿すことが可能である．

4　先 行 研 究

先行研究 [14] によるゲノム秘匿検索アプリケーションを，本研究のメモリ性能評価に用いるため，本章ではその概要を述べる．[15]

4.1　問 題 設 定

ゲノム秘匿検索を適用するモデルについて述べる．サーバにデータベースを設置し，ゲノム配列データをサンプルごとに並べる．このゲノムデータはゲノム配列全体を用いるのではなく，個体差が現れやすい特定の位置の塩基を取り出した SNP（一塩基多型）を並べた SNP 配列を用いている．ゲノムデータは A, G, C, T の 4 種の塩基配列から構成されるため，ゲノム秘匿検索は 4 種に限定された文字列検索とみなす．サンプルそれぞれの長いゲノム配列データを行ごとに並べて二次元配列状のデータベースとすることで，各サンプルについての文字列検索を行うことが出来る．ゲノム秘匿検索システムはサーバ・クライアント形式で実装される．サーバはクライアントから一致判定を行いたいクエリ文字列を FHE により暗号化したものとゲノム配列上の検索開始点（以下 ポジション）を受け取ると，データベース上のデータと FHE 演算によるマッチング判定を行い，その結果を返す．この検索を，サーバとクライアントの双方のデータが秘匿された状態で，可能な限り高速に行うことが先行研究の目的である．

4.2　手 法 概 要

石巻ら（2016）は従来の FHE を用いたゲノム秘匿検索手法に bootstrapping を導入し，その演算の最適化を行うことで，計算量の削減を行った [14]．石巻らのシステムはサーバとクライアントが 1:1 で問い合わせを行うもので，サーバはクライアントから検索したい文字列を暗号化処理したものとその文字列を検索したい配列上のポジションを受け取り，秘匿検索を行ってマッチしたか否かの結果を返す．
また，先行研究 [14] ではゲノムデータの秘匿検索を高速に行うために，ゲノム配列のデータベースを離散データ構造である Positional-Burrows Wheeler Transform (PBWT) [16] の形に変換している．これはゲノムデータに対して列ごとのソートを行ったもので，クエリとデータベースに含まれるサンプルとのマッチング判定の計算量を大幅に削減することが出来る．また，クライアントがサーバにダミーを含めた複数の検索開始ポジションを伝えることで，実際に利用するポジションを秘匿することが出来る等，秘匿性向上のための工夫も為されている．

4.3　ゲノム秘匿検索アプリケーション

具体的なアプリケーションの流れを以下に示す．ゲノム秘匿検索アプリケーションの目的は，ゲノムデータベースに対して問い合わせを行い，クエリとデータベース間でのマッチの有無について検索結果を得ることである [17]．クエリを生成するのをクライアント側，ゲノムデータベースを保持し，それとクエリとのマッチ判定を行って結果を送信するのをサーバ側として，クライアント・サーバ型のシステムとする．図 3 にアプリケーションの大まかな流れを示す．

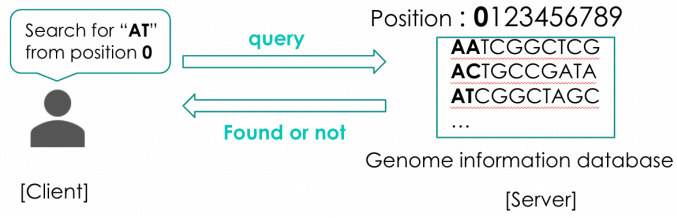

図 3　アプリケーションの流れ

細かいアプリケーションの処理の流れは以下である．

（1） クライアントは検索したい文字列とその文字列の検索開始位置の情報をクエリとして生成する．
そのクエリ全文を暗号化し，公開鍵などと共にサーバに送信する．

（2） サーバは FHE を用いた秘匿検索演算を行い，保持するデータベースとの一致判定を行う．
bootstrap 処理によるノイズのリセットを適宜行う．

（3） サーバはマッチしたか否かの結果をクライアントへ送信する．

（4） クライアントは自身の秘密鍵によって復号を行い，結果を得る．

データベース内でゲノム配列データをサンプルごとに並べることによって，各サンプルを特定のポジションから検索することが可能となる．この仕組みは，PBWT の手法を用いることで実現される．また，クライアントはダミーを含む複数のポジションを指定することで，実際に用いるポジションを秘匿することが出来る．このように，ゲノム検索を高速かつできるだけ秘匿に行う工夫が施されたアプリケーションである．実装は C++で行われている．また，完全準同型暗号計算には GitHub 上で公開されている HElib [7] を，分散時における各マシンの制御のための MPI(Message Passing Interface) を利用するライブラリとしては，Open MPI [18] が用いられている．

5 実　　験

5.1 検 証 方 法

本研究では上で述べた完全準同型暗号を用いたゲノム秘匿検索アプリケーションを異なる記憶装置を用いて実行し，実行時間を計測する．そして，実行条件の違いによる分析を行う．今回注目するのは swap 処理である．プログラム実行に必要なメインメモリの量が容量を超える際には，swap 処理を行なってストレージに領域を確保する必要がある．この場合において，メインメモリに比べると処理に時間がかかるストレージへのアクセスが大きなデータを扱う暗号化アプリケーションの実行時間にどの程度影響するかという点に着目する．本実験では使用コンピュータリソースが限られるためにメインメモリが不足するクラウド環境を想定し，使用可能なメインメモリを制限して，メインメモリの外の swap 領域へのアクセスを発生させる．そして，その swap 領域に永続メモリや高性能 SSD などの記憶装置を指定して，アクセス速度の差を検証していく．

5.2 実 験 条 件

5.2.1 アプリケーション使用可能メモリ

本研究ではクラウドコンピュータのインスタンスを利用する際に，限られたメモリ容量の中でビッグデータマイニングを行うとメモリが不足して swap 処理によるストレージ領域の利用が発生することを想定している．一般的なインスタンスではせいぜい 128GB 程度のメモリが搭載され，数テラバイトのデータが用いられる．本実験で用いるデータサイズは 108KB であるが，FHE で暗号化するとそのサイズは一万倍程度に膨れ上がるため 1 GB 程度になる．実験環境の章で示す通り本実験では使用可能メモリを 1GB に制限している．これらの 108KB のデータサイズ 1GB の使用可能メモリは実験の都合上，実際の実行条件を縮小して行っているものであり，実用上に起きうる現象を再現している．

5.2.2 アプリケーション実行パラメータ

実験では，4.3 節に示したゲノム秘匿検索アプリケーションの実行の際に指定するパラメータを変化させて実験を行なった．パラメータは，ゲノムデータ (クエリ) の長さと検索の開始位置のバリエーションの 2 種類であるが，今回の実験では，メモリへの負荷に影響を及ぼすことができる，クエリの長さを変更させて実行した．表 1 に，パラメータとして用いた 3 段階の数値を示す．

5.2.3 使用記憶装置

本実験では，完全準同型暗号を使用した際にメインメモリの容量が不足し swap 処理が発生することを想定し，その swap 領域に異なる記憶装置を用いることによってその結果を考察する．ここでは今回用いた記憶装置について説明する．
まず不揮発メモリについてである．使用した不揮発メモリの性能については表 2 に示す．

次に，使用した 4 種類の SSD について表 3 で説明する．

5.3 実 験 環 境

表 4 に示すスペックを持つサーバ上でゲノム秘匿検索アプリ

表 1　アプリケーション実行時に使用したパラメータ

クエリ長	3
	4
	5

表 2　不揮発メモリ性能

製品名	Intel Optane 200 Series [21]
容量	128GB *2 = 256GB
インタフェース	DDR-T

表 3　使用 SSD 性能

	Intel OPTANE SSD 905P [22]	Samsung 980 PRO [23]	Kioxia EXCERIA PLUS SSD [24]	INTEL SC2KB019T8 [25]
capacity	118GB	500GB	1TB	1.92TB
memory type	3D XPoint	NAND Flash	NAND Flash	NAND Flash
interface	NVMe	NVMe	NVMe	SATA

表 4　サーバ性能

CPU	Intel®Xeon®Silver 4313 16 Cores
DRAM	DDR4-2667 16GB*12=192GB 3200MHz
PCIe Gen	4.0
OS	Ubuntu 18.04.6 LTS
Docker version	20.10.7

ケーションの実行を行う．表 3 の 4 種類の SSD を swap 先として指定して実行する他に，実行に十分なメモリを割り当てることで swap 処理を発生させない条件 (1) を加えて，表 5 に示す 6 種類の異なる条件下で計測していく．なお，アプリケーションは Docker コンテナ内で実行しており，コンテナが使用できるメインメモリの量を制限することによってアプリケーション実行時に swap 処理を引き起こしている．

表 5　実 行 条 件

	メインメモリ	不揮発メモリ [21]	Intel Optane SSD [22]	Samsung SSD [23]	Kioxia SSD [24]	Intel SSD [25]
条件 (1)	○	-	-	-	-	-
条件 (2)	1GB(※)	○	-	-	-	-
条件 (3)	1GB(※)	-	○	-	-	-
条件 (4)	1GB(※)	-	-	○	-	-
条件 (5)	1GB(※)	-	-	-	○	-
条件 (6)	1GB(※)	-	-	-	-	○

(※) docker コマンドによってコンテナの使用可能メモリを 1GB に制限

		クエリ長4	クエリ長5	クエリ長6
条件1	メモリ制限なし	27分34秒	33分37秒	40分16秒
条件2	不揮発メモリ	27分53秒	35分18秒	42分41秒
条件3	Intel Optane 905P	27分46秒	35分16秒	42分47秒
条件4	Samsung 980 PRO	29分33秒	37分46秒	45分18秒
条件5	KIOXIA-EXCERIA PLUS	32分16秒	41分31秒	50分31秒
条件6	Intel SATA	36分53秒	47分23秒	57分15秒

表 6　実 行 時 間

5.4　実験結果と考察

上で述べた環境・条件で実験を行い実行時間を計測した．各条件ごとにかかった実行時間の結果を表 6 に示す．そして，いかにこの結果を 3 つに分けて考察する．

5.4.1　不揮発メモリについて

不揮発メモリを用いた条件 2 と，条件 1, 条件 3 の 3 つの結果を比較する．これらの結果をグラフにしたのが図 4 である．条件 1 ではメインメモリのみを実行に用いており，条件 2 と条件 3 ではメインメモリ 1GB と swap 領域にそれぞれ不揮発メモリと SSD を用いている．この条件 3 で用いた SSD に搭載されているメモリは 3D XPoint メモリであり，条件 1 の不揮発メモリに搭載されているものと同じである．まず図 4 を見てみると，条件 1 は 3 つのパラメータを用いたときいずれにおいても実行時間が最も短い結果であり，メインメモリの処理速度の速さが表れている．次に，条件 2 と 3 について注目してみると，クエリ長がどの長さのときにも実行時間にはほとんど差が出ていない．一般的に，DIMM などのメインメモリのデータ入出力に使われるバスは SSD が使う PCI バスよりも桁違いに高速である．それにも関わらず，今回の実験においては実行時間に差が出ない結果となった．この原因は少なくとも二点考えられる．まずは，アプリケーションの性質として，それほどリクエスト数が多くなかったため，性能差が反映されなかった結果であると考えられる．二点目は，CPU の性能として DDIO(Data Direct I/O Technology) [27] が備わっており，これによって PCIe バスに接続された SSD が L3 キャッシュとの間で直接読み書きの操作を行うことができ，ボトルネックが軽減されたことに起因すると考えられる．

5.4.2　搭載メモリについて

異なる種類のメモリを搭載している SSD を用いた，条件 3, 条件 4 の 2 つの結果を比較する．これらの結果をグラフにしたのが図 5 である．ここで用いた SSD には，条件 3 のものは 3D XPoint メモリが，条件 4 のものは NAND Flash メモリが搭載されている．前者の方がレイテンシが短いという特徴がある．今回の実験の結果において，図 5 に示す通り，3D XPoint メモリ

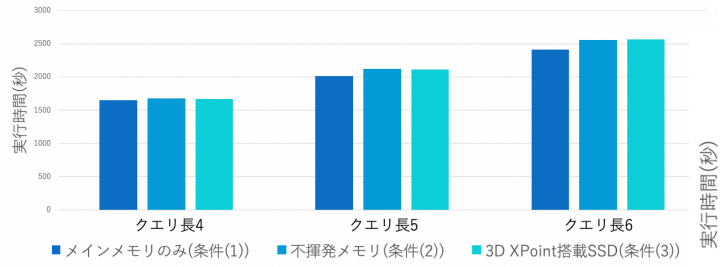

図 4　条件 1-3 の実行時間比較

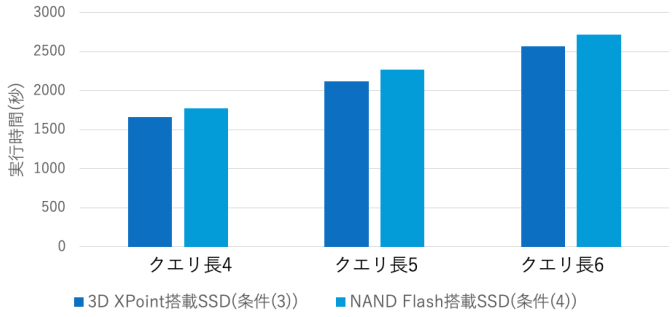

図 5　条件 3,4 の実行時間比較

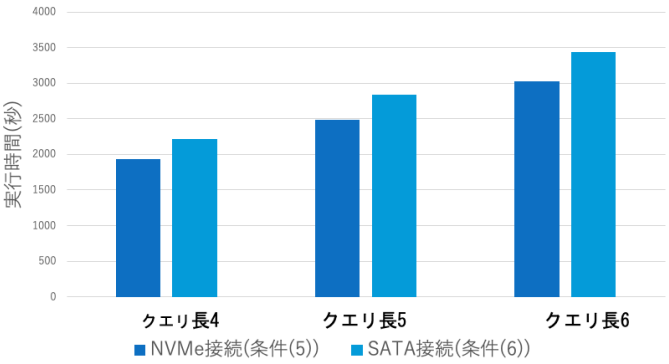

図 6　条件 5,6 の実行時間比較

を搭載した SSD の方がどのパラメータを使用した条件下においても実行時間が短くなっており，表 6 を見てもおよそ 2 分から 2 分半実行時間に差が出ている．また，パラメータのクエリ長が長くなるにつれて実行時間の差の開き方も変わっている．このことから，処理内容が増えてメインメモリが多く使われるようになるほど，swap 処理が増えて swap 領域の SSD が使われ，SSD のレイテンシ性能の差が表れることがわかる．

5.4.3　接続方式について

異なる接続方式の SSD を用いた条件 5 と条件 6 の 2 つの結果を比較する．これらの結果をグラフにしたのが図 6 である．条件 5 で用いられた SSD は NVMe 接続である一方で，条件 6 で用いられたものは SATA 接続である． なお，この二つの SSD に搭載されているのはどちらも NAND Flash であり，同じ種類のメモリが用いられている．接続方式による速度については，最新のマザーボードには最大スループット 600MB/秒の SATA III が使用されている一方で、NVMe ドライブの速度は最大 3,500MB/秒である [28]．表 6 や図 6 を見ると，接続方式の異なる SSD を使用した 2 つのケースにおいて 4 分半から 6 分半の差が出ている．よって，NVMe 接続が今回用いた完全準同型暗号アプリケーションの実行時間短縮に有効であることがわかった．

6　まとめと今後の課題

本実験から 3 つのことがわかった．まず不揮発メモリについて，メインメモリと同じバスに挿して使えるという処理速度に長けた製品であるが，今回扱った完全準同型暗号アプリケーション実行においては NVMe 接続の同じメモリを搭載した SSD とほとんど実行時間に差が出ない結果となった．次に，SSD に搭載されているメモリの種類について，低レイテンシな特徴を持つ 3D XPoint メモリと一般的に SSD によく用いられている NAND Flash メモリについて，本実験においてはかなり実行時間に差が見られた．よって，実行時間の長さが課題である完全準同型暗号を用いた処理においてレイテンシの性能が有用であることが示された．最後に SSD の接続方式について，NVMe 接続と SATA 接続の SSD を用いて実行時間を比較すると，NVMe 接続の高速な読み書き性能が発揮されて実行時間が短くなったことがわかった．以上のことから，完全準同型暗号を用いたアプリケーション実行においてメインメモリ以外の高性能な記憶装置の有効な性質がわかった．今後はこれら以外にも，完全準同型暗号実用化に向けて，記憶装置の有用な性質を調査していく．

謝　　辞

本研究の一部は、キオクシア株式会社と JST CREST JPMJCR22M2 の支援をうけて実施したものです。

文　　献

[1] Craig Gentry, et al.，”Fully homomorphic encryption using ideal lattices.” In STOC, Vol. 9, pp. 169–178, 2009
[2] Tibouchi Mehdi, 整数上完全準同型暗号の研究 (特集クラウドビジネスを支えるセキュリティ基盤技術) NTT 技術ジャーナル, Vol. 26, No. 3, pp. 71–75, 2014
[3] 安田雅哉，完全準同形暗号の応用 (小特集 完全準同形暗号の研究動向). 電子情報通信学会誌, Vol. 99, No. 12, pp. 1167–1175, 2016
[4] R. L. Rivest et al.，“ On data banks and privacy homomorphisms, ” Foundations of secure computation, vol. 4, no. 11, 1978, pp. 169 – 180
[5] NRI，クラウドの普及を支える新たな暗号技術への期待 デジタルイノベーション 2, itf_201710_6.pdf
[6] 佐藤宏樹, 馬屋原昂, 石巻優, 今林広樹, 山名早人. 完全準同型暗号のデータマイニングへの利用に関する研究動向. 第 15 回情報科学技術フォーラム F-002, 2016
[7] Shoup V. and Halevi S., http://shaih.github.io/HElib/ index.html

[8] PALISADE, https://palisade-crypto.org/software-library/
[9] Microsoft SEAL, https://github.com/Microsoft/SEAL
[10] KIOXIA，技術トピックス- DRAM 代替を目指した XL-FLASH™ によるデータベース性能比較実証 https://about.kioxia.com/ja-jp/rd/cutting-edge-researches/technology-topics/topics-20.html
[11] IBM Journal of Research and Development , ”Storage-class memory: The next storage system technology”, Volume: 52 , Issue: 4.5 , July 2008
[12] Intel，3D NAND が必要な理由 https://www.intel.co.jp/content/www/jp/ja/technology-provider/products-and-solutions/storage/why-3d-nand.html
[13] Oracle, https://blogs.oracle.com/oracle4engineer/post/persistent-memory-primer-jp
[14] Y. Ishimaki et al.,“Privacy-preserving string search for genome sequences with FHE bootstrapping optimization.”2016 IEEE International Conference on Big Data (Big Data). IEEE. 2016, pp. 3989–3991.
[15] 山田　優輝,『完全準同型暗号を用いた暗号化データベースにおける秘匿検索の分散処理による高速化』 http://www.is.ocha.ac.jp/ oguchi_lab/Publications/paper2017/deim2018_yuki.pdf
[16] R. Durbin, ”Efficient haplotype matching and storage using the Positional Burrows-Wheeler Transform (PBWT),”Bioin- formatics, vol. 30, no. 9, pp. 1266-1272, 2014
[17] K. Shimizu, K. Nuida, and G. Ratsch,“Efficient privacy-preserving string search and an application in genomics.”Bioinformatics 32.11 (2016), pp. 1652– 1661.
[18] Open MPI, https://www.open-mpi.org/
[19] David Wai-Lok Cheung, Vincent TY Ng, and Benjamin W Tam. Maintenance of discovered knowledge: A case in multilevel association rules. In KDD, Vol. 96, pp. 307–310, 1996.
[20] Micron, https://investors.micron.com/static-files/8ce1925c-f488-47c1-be33-15ba6049b747
[21] Intel,https://www.intel.co.jp/content/www/jp/ja/products/sku/203879/intel-optane-persistent-memory-200-series-128gb-pmem-module/specifications.html
[22] Intel， https://www.intel.co.jp/content/www/jp/ja/products/memory-storage/solid-state-drives/consumer-ssds/optane-ssd-8-series/optane-ssd-800p-series/800p-118gb-m-2-80mm-3d-xpoint.html
[23] SAMSUNG, https://www.samsung.com/semiconductor/minisite/jp/ssd/consumer/980pro/
[24] KIOXIA, https://personal.kioxia.com/ja-jp/ssd/exceria-plus-nvme-ssd.html
[25] Intel, https://www.mouser.jp/datasheet/2/612/dc_d3_s4510_s4610_series_brief-1534874.pdf
[26] Intel, https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology optane-technology/balancing-bandwidth-and-latency-article-brief.html
[27] Intel，https://www.isus.jp/products/vtune/vtune-cookbook-effective-utilization-of-data-direct-io-technology/
[28] Kingston, https://www.kingston.com/jp/blog/pc-performance/two-types-m2-vs-ssd