情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.6　No.1　75–83 (May 2016)

コンシューマ・システム論文

ニアデータ処理向けFPGAアクセラレータ

岡田 光弘[1,a)]　野村 鎮平[1]　鈴木 彬史[1]　藤本 和久[1]

受付日 2015年10月1日, 採録日 2016年2月23日

概要：近年，SSD の普及にともない HDD に比べて記憶デバイスからの読み出しスループットが 2 桁以上向上した．一方で，CPU のコア数・周波数向上によるパフォーマンス向上が鈍化してきており，アプリケーションの一部処理を FPGA にオフロードすることで，高性能化したいという要望が高まってきている．本研究では，購入可能な SSD と FPGA ボードを用いて，高性能なニアデータ処理向け FPGA アクセラレータを実現することを目的とし，FPGA が SSD からリードしたデータを処理する方式を提案する．さらに，本方式を評価する評価システムを構築し，SSD のリード性能と同等の高性能な FPGA アクセラレータが実現できることを確認した．

キーワード：ニアデータ処理，FPGA，SSD，NVMe 通信

FPGA Accelerator for Near-data Processing

Mitsuhiro Okada[1,a)]　Shimpei Nomura[1]　Akifumi Suzuki[1]　Kazuhisa Fujimoto[1]

Received: October 1, 2015, Accepted: February 23, 2016

Abstract: SSD's read throughput has been improved over two orders of magnitude compared with HDD's. On the other hand, the pace of CPU performance growth has slowed down. Therefore, an FPGA accelerator has been attracting attention to offload a part of processing in CPUs. In this paper, we propose a new method in which an FPGA can read data directly from an SSD using a standard SSD and FPGA board. We have developed a testbed equipped with the proposed method and confirmed its effectiveness.

Keywords: near-data processing, FPGA, SSD, NVMe communication

1. はじめに

近年，SSD（Solid State Drive）の普及にともない HDD（Hard Disk Drive）に比べて記憶デバイスからの読み出しスループットが 2 桁以上向上した．一方で，CPU のコア数・周波数向上によるパフォーマンス向上が鈍化してきており，アプリケーションの一部処理を FPGA にオフロードすることで，高性能化したいという要望が高まってきている [1]．

このような背景から，大量の蓄積データを読み出して解析するビックデータ解析の分野では，記憶素子の近くで処理するニアデータ処理に関する研究がさかんに行われている [2], [3]．

本研究では，コンシューマ製品（購入可能な SSD と FPGA ボード）を用いて，高性能なニアデータ処理向け FPGA アクセラレータを実現することを目的とし，FPGA が SSD からリードしたデータを処理する方式を提案する．また，本 FPGA アクセラレータを画像マイニングシステムへ適用することを想定し，マイニング処理の前処理に用いられるガウシアンフィルタを FPGA にオフロードして，FPGA アクセラレータの性能評価を行う．

2. 本研究の狙い

本章では，本研究の狙いについて述べる．**図 1** は，FPGA アクセラレータのデータフローについて，3 つの方式を示している．以下，画像処理を FPGA にオフロードする前

[1] 株式会社日立製作所研究開発グループ情報通信イノベーションセンタ
Center for Technology Innovation–Information and Telecommunications, Research & Development Group, Hitachi, Ltd., Yokohama, Kanagawa 244–0817, Japan
[a)] mitsuhiro.okada.uf@hitachi.com

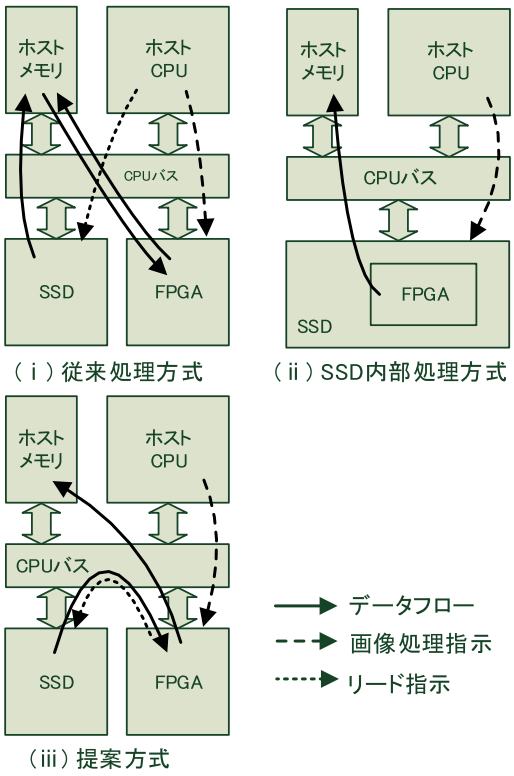

図 1　FPGA アクセラレータのデータフロー
Fig. 1　Data flows of an FPGA accelerator.

提で説明する．

（ｉ）従来処理方式は，文献 [4], [5] で提案されている一般的な FPGA アクセラレータのデータフローである．これらの FPGA アクセラレータは，ホストメモリに格納されているデータの処理を対象としている．そのため，ホスト CPU は，SSD に対してリード指示を発行し，リードしたデータをホストメモリにいったん格納してから，FPGA に対して画像処理指示を発行する必要がある．この方式では，ホスト CPU はリード指示と画像処理指示の最低 2 回の指示が必要となる．さらに，SSD からリードしたデータがホストメモリを経由するため，ホストメモリの帯域を大幅に使用してしまうという欠点がある．

それに対する改善策として，文献 [2], [3] では SSD 内部にオフロード処理用の FPGA を内蔵する方法を提案している（(ii) SSD 内部処理方式）．この方式では，ホスト CPU が SSD に画像処理指示を発行すると，SSD 内部の FPGA で画像処理を行い，ホストメモリに画像処理結果のデータを返すことができる．そのため，ホストメモリの帯域を余分に使用しないという利点がある．しかしながら，購入可能な SSD にはこのような機能がないため，専用の SSD を自製する必要がある．

そこで，本研究では FPGA が SSD からリードしたデータを処理する方式を提案する（(iii) 提案方式）．この方式では，ホスト CPU が FPGA に画像処理指示を発行した後，FPGA が SSD に対してリード指示を出す．そのため，

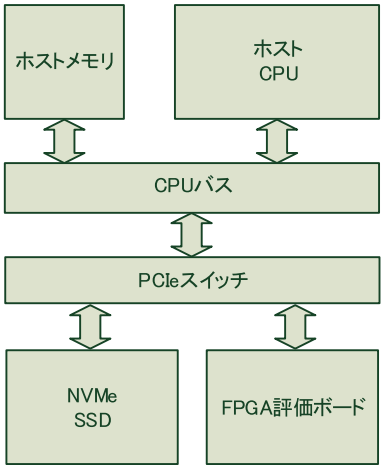

図 2　評価システムの構成
Fig. 2　Evaluation system.

ホスト CPU は，FPGA の処理が完了するまで，何も処理をする必要がなく，ホスト CPU の指示回数は (ii) SSD 内部処理方式と同等となる．また，ホストメモリにデータを転送する途中に FPGA で処理するため，SSD 内部で処理するよりレイテンシは増えるが，FPGA 内の処理がボトルネックにならないように設計することで，SSD 内部で処理する場合と同等のスループットが期待できる．さらに，SSD と FPGA ボードは完全に独立しているので，専用の SSD である必要がなく，コンシューマ製品の組合せで実現可能な構成である．

3.　評価システムの構成

図 2 に本研究で試作する評価システムの構成を示す．ホスト CPU とホストメモリが接続された CPU バスに PCI Express（PCIe）[*1] スイッチを接続し，その先に SSD と FPGA ボードを接続する構成を考案した．PCIe スイッチを使用した理由は，SSD からリードしたデータを PCIe スイッチ内でルーティングして FPGA に入力できるため，CPU バスへの負荷も図 1 (ii) SSD 内部処理方式と同等にできると考えたからである．

SSD については，様々ある通信プロトコル [6] の中で最も高性能である Non-Volatile Memory Express（NVMe）プロトコル [7] に対応した PCIe 接続の SSD を使用する．以降，単に SSD と記載したものは NVMe プロトコル対応の PCIe 接続の SSD を表す．

FPGA ボードについては，PCIe 接続の FPGA 評価ボードを使用し，通信プロトコルは SSD と同一の NVMe プロトコルに統一した．

次章で NVMe プロトコルについて説明する．

4.　NVMe プロトコルの概要

NVMe プロトコルは，2011 年に Version1.0 としてリリー

[*1] PCI Express（PCIe）は，PCI-SIG の登録商標です．

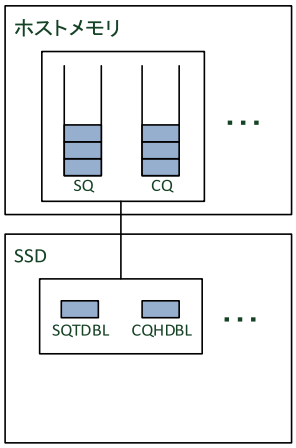

図 3　NVMe 通信のインタフェース
Fig. 3　NVMe communication interface.

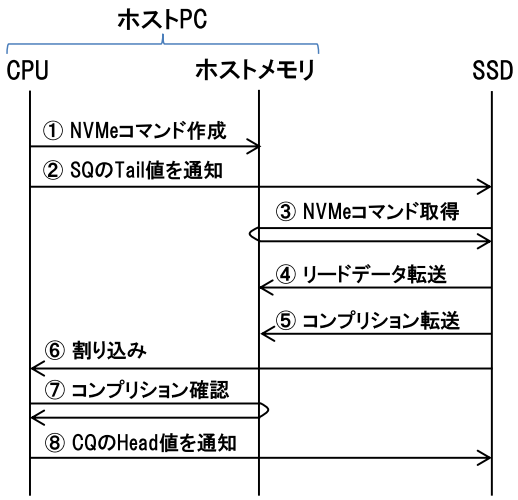

図 4　リードコマンドの処理フロー
Fig. 4　Processing flow of a read command.

スされた低処理負荷の通信プロトコルの規格である．

図 3 に NVMe 通信のインタフェースを示す．NVMe プロトコルでは，CPU がアクセスするホストメモリ上にコマンド用のキュー（I/O Submission Queue（SQ））とコンプリション用のキュー（I/O Completion Queue（CQ））を設ける．さらに，SSD の内部に，SQ の Tail 値を通知するレジスタ（Submission Queue Tail Doorbell（SQTDBL））と CQ の Head 値を通知するレジスタ（Completion Queue Head Doorbell（CQHDBL））を設ける．

このインタフェースは，NVMe ドライバの起動時に SSD と通信して自動で生成される．また，このインタフェースは，複数生成することが可能で，通常は CPU コアごとに生成する．これにより，CPU コア間のロックを排除することができるので，低負荷での通信が可能となっている．

次に，NVMe プロトコルの具体的な通信例をリードコマンドの処理フローを用いて説明する（**図 4**）．まず，ホスト CPU で動作する NVMe ドライバは，ホストメモリ上の SQ に NVMe コマンドを作成する（①）．次に，SQTDBL を用いて SSD に SQ の Tail 値を通知する（②）．SSD は，SQTDBL の Tail 値の更新を検知し，NVMe コマンドを取得する（③）．NVMe コマンドの中には，リードコマンドを示すオペコード，リードしたいデータの格納位置を示す転送元アドレスとそのサイズ，リードデータの転送先アドレス等が入っている．

次に，SSD はフラッシュメモリからデータを読み出して，ホストメモリにリードデータ転送する（④）．リードデータの転送が完了した後，SSD はホストメモリの CQ にコマンド完了を知らせるコンプリションを転送し（⑤），ホスト CPU に割込みを発行する（⑥）．割込みを受けた CPU は，コンプリションを確認して，コマンド終了を認知する（⑦）．最後にホスト CPU は，CQHDBL を用いて SSD に CQ の Head 値を通知する（⑧）．

以上が NVMe プロトコルの一連の動作となる．

5.　関連研究

5.1　PCIe ファブリックを用いたデバイス間通信の研究

PCIe ファブリックを使った演算デバイス間の通信についてはいくつかの手法が研究されている．たとえば，文献 [8], [9] では FPGA に通信インタフェースを実装して，GPU と FPGA 間の通信を行っている．また，文献 [10] の GPU 向けの開発環境（CUDA）では，GPU と GPU のダイレクト通信を可能にする専用の API を提供している．

しかしながら，これらの文献では，演算デバイス間で通信しており，演算デバイスと記憶デバイス間の通信にそのまま利用することはできない．

5.2　FPGA と汎用記憶デバイスの連携処理に関する研究

FPGA と汎用記憶デバイスが連携して処理する技術として，文献 [11] がある．文献 [11] では，FPGA と汎用 HDD を一体にした処理装置を開発し，FPGA が HDD から直接データを読み出して FPGA 内でデータベース処理の一部処理を実行する技術が提案されている．

しかしながら，この文献では FPGA と汎用 HDD を一体にした専用の処理装置を開発しており，コンシューマ製品を組み合わせるというコンセプトではない．また，この技術は HDD を対象としており，インタフェースが異なる高性能な SSD にそのまま利用することはできない．

5.3　高速画像処理回路の研究

画像処理回路の研究としては，連続で入力される画像を対象として，リアルタイムで処理する手法が研究されている．文献 [12] では，ノイズ除去回路を 96.5 MHz で実装しており，4 Mpixel/second の処理速度を実現している．

しかしながら，この論文では，連続で入力される画像を 96.5 MHz の動作周波数で，1 画素ずつ処理しているため，GB/s オーダのスループットで画像処理することはで

きない．

6. 解決すべき課題

本研究で解決すべき課題は，3 つある．

1 つ目の課題は，FPGA と SSD の通信技術の確立である．ホスト CPU が FPGA への指示のみで，オフロード処理を完了するためには，FPGA が SSD にリードコマンドを発行する必要がある．しかし，NVMe インタフェース（SQ/CQ と SQTDBL/CQHDBL）は，NVMe ドライバのみが管理している情報のため，FPGA は NVMe インタフェースにアクセスできないという課題がある．

2 つ目の課題は，FPGA の処理フローの確立である．現状 FPGA が SSD にリードコマンドを発行して，リードしたデータを処理する研究は行われていないため，新たな処理フローを提案する必要がある．

3 つ目の課題は，FPGA の実装である．図 1 (ii) SSD 内部処理方式と同等の処理性能を実現するためには，SSD のリード性能と同一の処理能力が求められる．本研究で使用する SSD の最大リード性能は 2.6 GB/s であるため，2.6 GB/s 以上の処理性能でフィルタ処理が可能な FPGA を設計する必要がある．

7. 提案手法

7.1 FPGA と SSD の通信技術

FPGA が SSD にリードコマンドを発行するためには，FPGA がアクセスできる NVMe インタフェースを用意する必要がある．そこで，**図 5** のように SSD と FPGA に新たに FPGA がアクセスする専用の NVMe インタフェース（SSD に SQTDBL_S，CQHDBL_S，FPGA に SQ_S，CQ_S）を作成することを提案する．

この FPGA 専用の NVMe インタフェースの生成は，FPGA ドライバと NVMe ドライバに実装した．

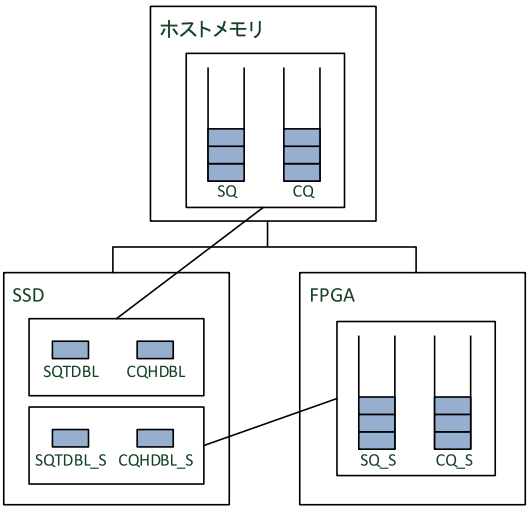

図 5　FPGA 専用 NVMe インタフェース
Fig. 5　NVMe interface for an FPGA.

図 6 に FPGA 専用 NVMe インタフェースの作成フローを示す．初めに，FPGA ドライバは，FPGA 内の SQ_S のアドレスと CQ_S のアドレス，それぞれのキューの深さを NVMe ドライバに通知する (a)．NVMe ドライバは，NVMe の管理コマンドを用いて，SSD に SQTDBL_S と CQHDBL_S を作成してアクセス可能な状態にする (b)．その後，NVMe ドライバは，SQTDBL_S と CQHDBL_S のアドレスを FPGA ドライバに通知する (c)．最後に，FPGA ドライバは，SQTDBL_S と CQHDBL_S のアドレスを FPGA に通知する (d)．

以上，FPGA ドライバと NVMe ドライバが連携する処理を各々のドライバに追加することで，FPGA は SSD にリードコマンドを発行可能になる．

7.2 FPGA の処理フロー

次に，提案する FPGA の処理フローを述べる．**図 7** は，図 1 (iii) 提案方式の処理フローを具体化したものである．ここで，ホスト CPU と FPGA 間の通信は，FPGA ドライバ起動時に作成する NVMe インタフェース（SQ_F/CQ_F

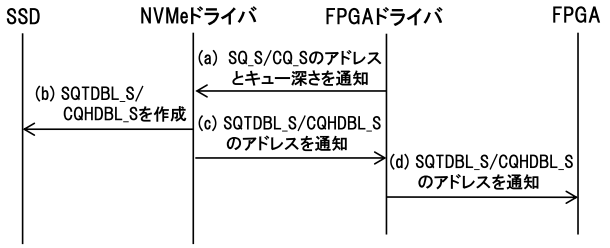

図 6　FPGA 専用 NVMe インタフェース作成フロー
Fig. 6　Creation flow of NVMe interface for an FPGA.

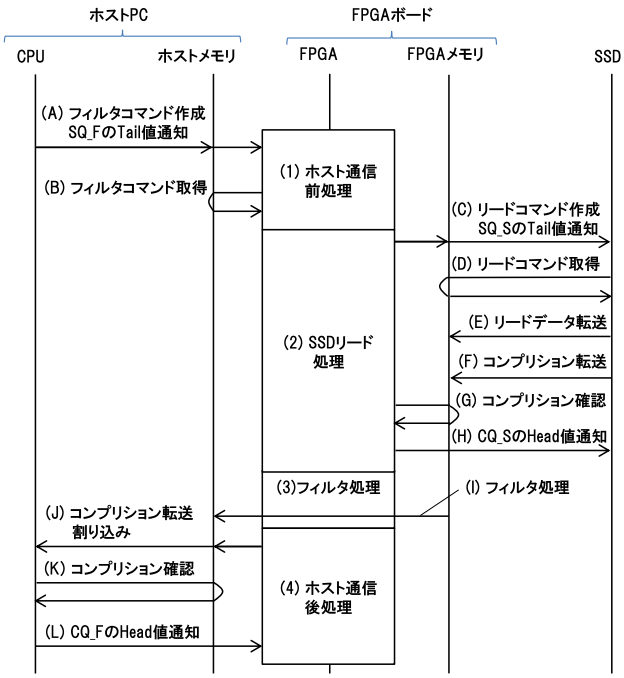

図 7　FPGA の処理フロー
Fig. 7　Processing flow of an FPGA.

と SQTDBL_F/CQHDBL_F）を用いて行う．以下，図 7 の (1)～(4) の各処理について詳述する．

(1)　ホスト通信前処理

ホスト通信前処理は，ホスト CPU からフィルタコマンドを受け付ける処理である．ホスト CPU は，ホストメモリ上の SQ_F にフィルタコマンドを作成し，SQTDBL_F を用いて FPGA に SQ_F の Tail 値を通知する (A)．FPGA は，SQTDBL_F の更新を検知して，ホストメモリからフィルタコマンドを取得する (B)．このフィルタコマンドの中には，フィルタコマンドを表すオペコード，フィルタ処理対象のデータが格納されている SSD 内の格納アドレスとサイズ，フィルタ処理結果格納用のホストメモリ内の格納アドレス等が含まれる．

(2)　SSD リード処理

SSD リード処理は，SSD からフィルタ処理対象のデータをリードする処理である．FPGA は，FPGA メモリ上の SQ_S にリードコマンドを作成し，SQTDBL_S を用いて SSD に SQ_S の Tail 値を通知する (C)．SQTDBL_S の更新を検知した SSD は，リードコマンドを取得する (D)．次に SSD は，フラッシュメモリからリードしたデータを FPGA メモリに転送する (E)．その後，SSD は，FPGA の CQ_S にコンプリションを転送する (F)．FPGA は，定期的にコンプリションの有無を確認しており (G)，コンプリションを受け取った後，CQHDBL_S を用いて SSD に CQ_S の Head 値を通知する (H)．

なお，一度にフィルタ処理するピクチャのデータサイズが SSD の最大リクエストサイズよりも大きい場合は，リードコマンドを複数発行することで対応する．この場合はすべてのリードコマンドのコンプリションを確認した後，SSD リード処理が完了となる．

(3)　フィルタ処理

フィルタ処理は，FPGA メモリからデータを読み出してフィルタ処理を行う処理である．フィルタ処理後は，ホストメモリにフィルタ処理結果を転送する (I)．

(4)　ホスト通信後処理

ホスト通信後処理は，フィルタコマンドの完了をホスト CPU に通知する処理である．FPGA は，ホストの CQ_F にコンプリションを転送し，割込みを発行する (J)．割込みを受けたホスト CPU は，コンプリションの内容を確認する (K)．最後にホスト CPU は，FPGA の CQHDBL_F を用いて FPGA に Head 値を通知する (L)．

8.　FPGA の実装

8.1　ハード/ソフト処理の切り分け

本試作では，図 7 の処理フローを実現する FPGA を実装する必要があるが，FPGA での処理をすべてハードウエアで実装すると設計工数が大きくなる．そこで，処理に応じて FPGA 内の組み込みプロセッサを使用する方針にした．

表 1　ハード/ソフト処理切り分け
Table 1　Hardware/software selection.

処理内容	ハード/ソフト
ホスト通信前処理/後処理	ソフトウエア
SSD リード処理	ソフトウエア
フィルタ処理回路制御	ソフトウエア
フィルタ処理回路コア	ハードウエア
データバス	ハードウエア

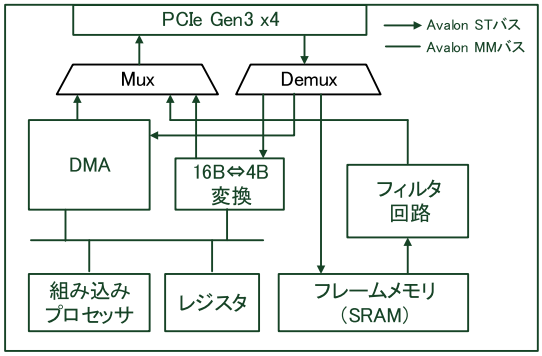

図 8　FPGA の回路構成
Fig. 8　Block diagram of the FPGA.

各処理のハード/ソフト処理の切り分け結果を**表 1** に示す．SSD からリードしたデータが流れるデータバスとフィルタ処理回路コアの部分は，2.6 GB/s 以上の処理性能が必要なため，ハードウエアで設計した．一方，処理内容が複雑かつ，性能要件が厳しくない処理（ホスト通信前処理/後処理，SSD リード処理，フィルタ処理回路制御）に関してはソフトウエアで設計した．

8.2　FPGA の回路構成

図 8 に FPGA の回路構成を示す．FPGA は Stratix[*2] 5（5SGXEA7K2F40C2N）を使用し，200 MHz の動作周波数で設計した．FPGA 内部は，PCIe Gen3 ×4，組み込みプロセッサ，DMA（Direct Memory Access），Mux（Multiplexer），Demux（Demultiplexer），16 B⇔4 B 変換，レジスタ，フレームメモリ（SRAM）とフィルタ回路の 9 つのモジュールで構成した．すべてのモジュールを同一バスに接続してしまうと性能保障が難しくなるため，用途に応じて 2 つのバスに分離した．まず，SSD のリードデータが通るモジュールのバスは，ストリーミング形式の Avalon[*3] ST バスを採用し，2.6 GB/s 以上の処理性能を実現するために 16 Byte 幅で設計した．一方，プロセッサがアクセスするモジュールのバスは，ハンドシェイク形式の Avalon MM バスを採用し，プロセッサのアクセス幅に合わせて 4 Byte で設計した．さらに，ホスト CPU と組み込みプロセッサが通信できるようにするために，16 B⇔4 B 変換モ

[*2] Stratix は，Altera Corporation の登録商標です．
[*3] Avalon は，Altera Corporation の登録商標です．

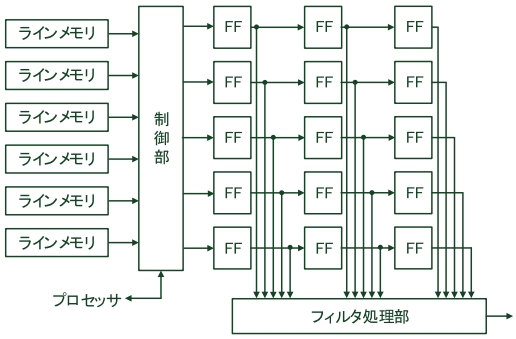

図 9　フィルタ回路の構成
Fig. 9　Block diagram of the filter processing.

ジュールを両バス間に挿入した．

なお，表 1 に記載のソフトウエアで行う 3 つの処理は，組み込みプロセッサ，レジスタ，DMA を連動させて動作させることで実現できるように設計している．

8.3　フィルタ回路

フィルタ回路は，5 Tap × 5 Tap のガウシアンフィルタを実装した．フィルタ回路の構成を**図 9** に示す．フィルタ処理部に 5 ライン分の画素を同時に入力するために，1 KB（16 Byte × 64 word）のラインメモリ 5 本と，次ラインの先読み用に 1 本，計 6 本のラインメモリを使用した．また，すべての信号線を 16 Byte で設計することで，3.2 GB/s 処理可能なフィルタ回路を実現した．

制御部は，プロセッサからの指示で処理を開始する．6 本のラインメモリからフィルタ処理部へ入力する 5 本のラインを選択してフリップフロップ（FF）に画素を供給する．また，画像端のパディング処理の制御も行う．

フィルタ処理部は，16 画素を同時にフィルタ処理するため，400 個（16 画素 × 25 係数）の掛け算ハードマクロ（DSP BLK）を使用し，1 クロックでフィルタ係数の分子の掛け算を行う．その後，25 係数の足し算を 2 クロックかけて行い，最後にフィルタ係数の分母の割り算をして結果を出力する．

9.　評価環境および評価条件

9.1　評価環境

図 10 に，図 2 の評価システム構成を実際に構築した実機評価環境を示す．ホスト PC に PCIe Gen3 ×16 ケーブルで PCIe スイッチに接続し，PCIe スイッチに FPGA ボードと SSD を接続した．各機材の仕様を，**表 2** にまとめる．

9.2　評価条件

本研究では，提案する FPGA アクセラレータを用いた提案方式，比較方式として，ホスト CPU でフィルタ処理を行ったソフト処理方式，SSD のリードのみを行った SSD リード処理の 3 方式を評価する．

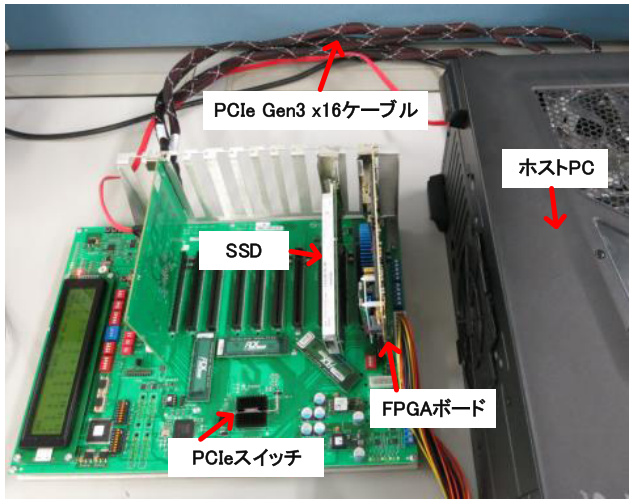

図 10　実機評価環境
Fig. 10　Evaluation environment.

表 2　評価環境の仕様
Table 2　Specification of the evaluation environment.

機材名	仕様
ホスト CPU	Intel Core i7 3930[*4]（3.2GHz）
ホストメモリ	DDR3 1333MHz（8GB）
PCIe スイッチ	PCIe スイッチ評価キット ：PEX 8748-BA RDK （ホスト接続 PCIe Gen3 x16）
FPGA ボード	Stratix 5 GX FPGA 開発キット ：DK-DEV-5SGXEA7N
SSD	DC P3700（800GB）

表 3　評価条件
Table 3　Evaluation conditions.

項目	評価条件
ホスト CPU	1 コアに制限
PCIe スイッチ	提案方式のみ使用
評価画像	1024×512 画素，輝度のみ， 非圧縮画像，1000 枚（512MB）
コンパイラ	g++（-O3 オプション）
フィルタ	ガウシアンフィルタ（5Tap×5Tap）
コマンド多重数	1〜5 多重

表 3 に評価条件を示す．表 3 に示すように，ホスト CPU は，すべての方式を同一条件で比較するため，Linux の設定で 1 コアに制限した．PCIe スイッチは，提案方式のみ使用し，ソフト処理方式および SSD リード処理は使用せずに，SSD をホスト PC のマザーボードに直結した．評価画像は，1024 × 512 画素の輝度のみの非圧縮画像（512 KB）とした．また，評価枚数は，測定結果のばらつきが十分に

[*4] Intel Core i7 は，Intel Corporation の登録商標です．

小さくなる枚数として，1,000 枚（512 MB）を使用した．コンパイラは，g++ を使用し，-O3 オプションを用いてコンパイルした．

ソフト処理方式のフィルタ処理は，OpenCV 2.4 [13] の 2D フィルタ関数（cvFilter2D）を用いた．その際，2D フィルタ関数のフィルタ係数は，5 Tap × 5 Tap のガウシアンフィルタの係数を設定した．また，ホスト CPU は 1 コアのみに制限しているため，複数コアによるフィルタ処理の並列化は行っていない．

また，SSD のリード性能は，コマンドを多重で発行しないと最高性能を得られないため，リード性能が安定する 5 多重までコマンドの多重数を変更して評価した．

10. 評価結果

10.1 フィルタ処理性能

図 11 にコマンド多重数におけるフィルタ処理性能のグラフを示す．横軸にコマンド多重数，縦軸に各方式の処理性能を示している．

10.1.1 FPGA 利用によるフィルタ処理高速化の評価

提案方式とソフト処理方式を比較する．提案方式は，ソフト処理方式に対して，最大 14 倍の性能改善を実現していることを確認した．これは，ホスト CPU で実行していたフィルタ処理を，FPGA にオフロードした効果だと考える．

10.1.2 FPGA 追加による処理性能への影響評価

提案方式と SSD リード処理を比較する．コマンド多重数が 2 多重以上になると，提案方式と SSD リード処理が同等の処理性能になることから，SSD 内部の FPGA で処理する方式と同等の処理性能が実現できたといえる．コマンド多重数 1 のとき，提案方式が SSD リード処理に比べて処理性能が劣っているのは，FPGA でフィルタ処理をしたことによるレイテンシの増加が原因であると考える．2 多重以上では，このレイテンシ増加分を隠蔽できたため，同等の処理性能が得られたと考える．

10.2 CPU 使用率

次に，SSD の性能が安定した状態（コマンド多重数 4 のとき）の CPU 使用率を評価する．**図 12** は，CPU 処理時間が記録されている stat ファイルの内容を 0.1 秒ごとに取得して CPU 使用率をグラフ化したものである．横軸に処理時間，縦軸に CPU 使用率を表す．

図 12 のグラフから，ソフト処理方式はつねに 100%の使用率に対して，提案方式は，CPU 使用率が少ないことが分かる．これは，ホスト CPU が FPGA にフィルタ処理を指示した後，FPGA から完了の割込みが入るまで，IDLE 状態になるためである．この結果より，提案方式は処理時間が短縮されるだけではなく，CPU 使用率を抑える効果もあることが確認できた．

また，提案方式は，SSD リード処理と比較して CPU 使用率が少ない結果となった．これは，提案方式のコマンドの処理負荷が SSD リード処理より少ないためだと考える．本研究で用いた SSD の最大リクエストサイズは，128 KB であり，1 ピクチャ（512 KB）リードするためには，ホスト CPU は，4 回のリードコマンドを発行する必要がある．一方，提案方式のホスト CPU は，1 ピクチャ分のフィルタコマンドを 1 回発行し，FPGA が，4 回のリードコマンドを SSD に発行する．そのため，提案方式の CPU 使用率が SSD リード処理に比べて少なくなったと考える．

10.3 実装規模

最後に，FPGA の実装規模を評価する．**表 4** にブロックごとのロジック（ALMs），SRAM，DSP BLK の使用数を示す．表 4 の PCIe バスは，Demux，Mux，16 B⇔4 B 変換，Avalon ST バスの合計，プロセッサは，組み込みプロセッサ，レジスタ，Avalon MM バスの合計を表している．PCIe Gen3 ×4 は FPGA のハードマクロを使用したため，使用数のカウントに入らない．表 4 より，ロジック使用率 11%，SRAM 使用率 65%，DSP BLK 使用率 100%で FPGA に実装できた．

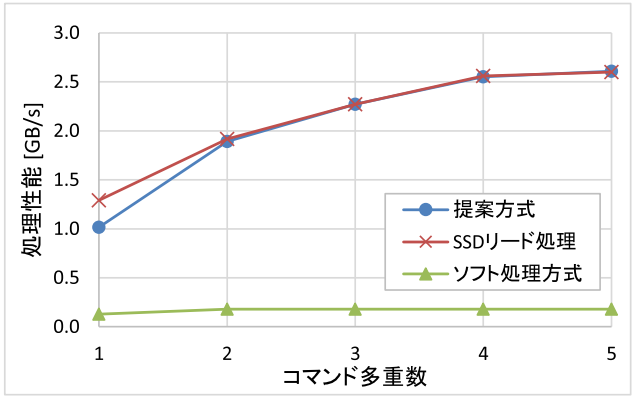

図 11　フィルタ処理性能
Fig. 11　Filtering performance.

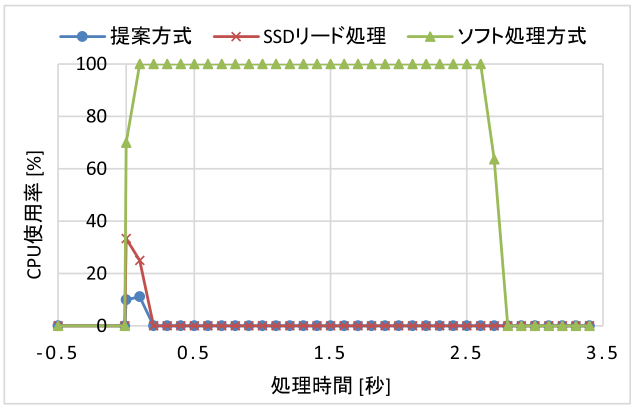

図 12　CPU 使用率
Fig. 12　CPU utilization.

表 4　実装規模
Table 4　Resource utilization.

ブロック名	ロジック（ALMs）	SRAM [KB]	DSP BLK
PCIe バス	9963	98	0
DMA	5366	868	0
プロセッサ	5496	586	2
フィルタ回路	3723	10	254
フレームメモリ	7	2560	0
合計	24555(11%)	4122(65%)	256(100%)

※合計の括弧内は FPGA の使用率

11. 結論および今後の課題

本論文では，コンシューマ製品を用いたニアデータ処理向け FPGA アクセラレータを提案した．FPGA が SSD からデータをリードして処理することで，SSD のシーケンシャルリード性能と同等の処理性能が得られることを実証した．これにより，専用の SSD を自製する必要がある SSD 内部の FPGA で処理する方式と同等の処理性能を実現できたといえる．

今後は，画像マイニングシステムに本 FPGA アクセラレータを搭載して画像マイニングシステムの性能評価を行う予定である．

参考文献

[1] Brooks, D., Chen, Y., Cong, J., Fang, Z., Reagen, B. and Shao, Y.S.: Rapid Exploration of Accelerator-rich Architectures: Automation from Concept to Prototyping, Introduction, *Tutorial on the 42nd International Symposium on Computer Architecture* (*ISCA 2015*), available from ⟨http://accelerator.eecs.harvard.edu/isca15tutorial/⟩.

[2] Jun, S.-W., Liu, M., Lee, S., Hicks, J., Ankcorn, J., King, M. and Xu, Arvind, S.: BlueDBM: An Appliance for Big Data Analytics, *Proc. 42nd International Symposium on Computer Architecture* (*ISCA 2015*), pp.1–13 (June 2015).

[3] Li, T., Huang, M., El-Ghazawi, T. and Huang, H.H.: Reconfigurable Active Drive: An FPGA Accelerated Storage Architecture for Data-Intensive Applications, *Proc. 2009 Symposium on Application Accelerators in High-Performance Computing* (*SAAHPC'09*) (July 2009).

[4] Giefers, H., Polig, R. and Hagleitner, C.: Accelerating arithmetic kernels with coherent attached FPGA coprocessors, *Proc. Design Automation & Test in Europe* (*DATE 2015*), pp.1072–1077 (Mar. 2015).

[5] Putnam, A., Caulfield, A.M., Chung, E.S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G.P., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Larus, J., Peterson, E., Pope, S., Smith, A., Thong, J., Xiao, P.Y. and Burger, D.: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, *Proc. 41st International Symposium on Computer Architecture* (*ISCA 2014*), pp.13–24 (June 2014).

[6] Cisco, EMC and Intel: The Performance Impact of NVMe and NVMe over Fabrics, available from ⟨http://www.snia.org/sites/default/files/NVMe_Webcast_Slides_Final.1.pdf⟩.

[7] Non-Volatile Memory Express (NVMe), available from ⟨http://www.nvmexpress.org/⟩.

[8] Bittner, R. and Ruf, E.: Direct GPU/FPGA Communication Via PCI Express, *Proc. 41st International Conference on Parallel Processing Workshops* (*ICPPW 2012*), pp.135–139 (Sep. 2012).

[9] Thoma, Y., Dassatti, A. and Molla, D.: FPGA2: An Open Source Framework for FPGA-GPU PCIe Communication, *Proc. 2013 International Conference of Reconfigurable Computing and FPGAs* (*ReConFig 2013*), pp.1–6 (Dec. 2013).

[10] NVIDIA: GPUDirect, available from ⟨https://developer.nvidia.com/gpudirect⟩.

[11] Netezza: The Netezza FAST Engines™ Framework, available from ⟨http://www.monash.com/uploads/netezza-fpga.pdf⟩.

[12] Kao, W.-C., Tai, H.-S., Shen, C.-P., Ye, J.-A. and Ho, H.F.: A Pipelined Architecture Design for Trilateral Noise Filtering, *Proc. 2007 IEEE International Symposium on Circuits and Systems* (*ISCAS 2007*), pp.3415–3418 (May 2007).

[13] OpenCV, available from ⟨http://opencv.org/⟩.

岡田 光弘　（正会員）

2004 年東京理科大学理工学部電気工学科卒業．2006 年同大学大学院修士課程修了．同年（株）日立製作所に入社．画像の高効率符号化および IT プラットフォームに関する研究に従事．

野村 鎮平

2012 年慶応義塾大学理工学部情報工学科卒業．2014 年同大学大学院修士課程修了．同年（株）日立製作所に入社．ストレージシステム分野の研究に従事．

鈴木 彬史

2006 年千葉大学工学部電気電子工学科卒業．2008 年同大学大学院修士課程修了．同年（株）日立製作所に入社．ストレージシステム分野の研究に従事．

藤本 和久　（正会員）

1985 年九州大学工学部電気工学科卒業．1987 年同大学大学院修士課程修了．同年（株）日立製作所に入社．2007～2012 年東北大学電気通信研究所教授．現在，（株）日立製作所に所属．IT プラットフォームに関する研究に従事．電子情報通信学会会員．