社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

カラム指向型データベース向けハードウェアキャッシュ機構の検討

濱田　耀彦†　　松谷　宏紀†,††,†††

† 慶應義塾大学大学院 理工学研究科　〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
†† 科学技術新興機構さきがけ
††† 国立情報学研究所
E-mail: †{hamada,matutani}@arc.ics.keio.ac.jp

あらまし　カラム指向型ストレージは、構造型ストレージ（NOSQL）の一種であり、行ごとに可変個のカラムを持つことができる。データは行キーを基に行単位でソートされているためスキャン範囲を指定したデータの取得を行うことができる。本研究では、FPGA（Field-Programmable Gate Array）を用いてカラム指向型データベースを高速化することを検討する。具体的には、10GbE のネットワークインタフェースを持つ FPGA ボード上にカラム指向型データベースのハードウェアキャッシュ（HBase Cache、HBC）を構築することを検討する。この FPGA ボードは PCI Express 経由でホスト PC に接続され、本 HBC では一定数の最新の連続した行を 1 つのブロックとしてホスト PC の主記憶上にキャッシュする。キャッシュにヒットすれば、ソフトウェアによるカラム指向型データベース処理を行わずに HBC がキャッシュしているデータを直接利用できるため、データアクセスを高速化できる。本論文では、データベースへのスキャン要求が HBC にヒットした場合、しなかった場合のスキャン操作に要す実行時間をモデル化し、HBC によって大幅な性能向上が可能であること示す。
キーワード　FPGA、アクセラレータ設計、キャッシュ、データベース、構造型ストレージ

A Hardware Cache Mechanism for Column-Oriented Databases

Akihiko HAMADA† and Hiroki MATSUTANI†,††,†††

† Graduate School of Science and Technology, Keio University 3-14-1, Hiyoshi,Yokohama, JAPAN 223-8522
†† PRESTO, Japan Science and Technology Agency
††† National Institute of Informatics
E-mail: †{hamada,matutani}@arc.ics.keio.ac.jp

Abstract　A column-oriented store is one of structured storages (NOSQLs), in which a variable number of columns are stored for each row. As rows in a column-oriented store are sorted, a scan operation between given start and stop rows is possible. In this paper, we discuss ways to accelerate a column-oriented store using an FPGA (Field-Programmable Gate Array). More specifically, we focus on a hardware-based cache mechanism of a column-oriented store (HBase Cache, HBC) implemented on an FPGA platform that equips 10GbE network interfaces. The FPGA board is mounted on a host computer via PCI Express. HBC caches a certain number of latest successive rows in a main memory of the host computer. When HBC hits, cached data retrieved from the host memory are available without software-based database processing; thus the column-oriented store can be accelerated. We develop a simple performance model of HBC that takes into account the cache hit and miss ratios of database scan requests, and then we show that a significant performance improvement can be achieved by the proposed HBC.
Key words　FPGAs, Accelerator design, Cache, Databases, Structured storages

1. はじめに

ネットワークサービス、金融、交通、気象などの分野においては日々膨大な量の情報が生成されており、近年では、このようなビッグデータを利活用するための取り組みが盛んに研究されている。

ビッグデータの格納先として、従来の関係データベース管理システムに加え、水平スケーラビリティに優れた構造型スト

レージが注目されている [1]。構造型ストレージの代表例として、キーとバリューの組を管理するキーバリューストア型 [2], [3]、行キーに対してカラム（名前と値の組）の集合を管理できるカラム指向型 [4], [5] がある。とりわけ、カラム指向型ストレージは、高いスケーラビリティを維持しつつ、行ごとに可変個のカラムを持つことができる。また、データは行キーを基に行単位でソートされているためスキャン範囲を指定したデータの取得を行うことができるなど高い柔軟性を兼ね備えている。このような構造型ストレージはビッグデータの格納先として有望であるため、より大きなデータを高速に処理するためにカラム指向型ストレージの更なる低遅延化、高スループット化が望まれている。

これらの構造型ストレージでは、計算時間に比して、メモリアクセスや通信（I/O）に要する時間が多いため、I/O 処理とデータベース処理の密な結合によって高性能化が実現できると考えられる。そこで、本研究では、4 本の 10 ギガビットイーサネット（10GbE）インタフェースを持つ FPGA（Field-Programmable Gate Array）ボードを対象に、カラム指向型の構造型ストレージのアクセラレータを構築することを検討する。具体的には、FPGA 上にカラム指向型データベースのハードウェアキャッシュ（HBase Cache、HBC）機構を構築することを検討する。

この FPGA ボードは PCI Express 経由でホスト PC に接続され、本 HBC では一定数の最新の連続した行を 1 つのブロックとしてホスト PC の主記憶上にキャッシュする。キャッシュにヒットすれば、ソフトウェアによるカラム指向型データベース処理を行わずに HBC がキャッシュしているデータを直接利用できるため、データアクセスを高速化できる。

本論文では、データベースへのスキャン要求が HBC にヒットした場合、しなかった場合のスキャン操作に要す実行時間をモデル化し、HBC によって大幅な性能向上が可能であること示す。ただし、本論文では、カラム指向型アクセラレータ開発の第一歩として FPGA ボードを用いた実機動作までは行わず、簡易ハードウェアモデルの論理合成および論理シミュレータによる評価までを行う。また、通信部分を含めたカラム指向型アクセラレータ全体の設計と実装は今後の課題としている。

本論文の構成は以下の通りである。2. 章で関連研究を紹介し、3. 章でカラム指向型データベース向けハードウェアキャッシュ機構を提案する。4. 章では、提案キャッシュ機構のモデル式と性能見積りを示し、5. 章で本論文をまとめる。

2. 関 連 研 究

本章では、関係データベースや構造型ストレージを FPGA を用いて高速化した研究例を紹介する。

関係データベース（RDB）におけるクエリ処理を高速化するために、2009 年ごろから FPGA を用いたデータベースのハードウェア処理が報告されるようになった。文献 [6] では、Glacier と呼ばれるコンパイラおよびライブラリが提案されている。これは、RDB のクエリを入力として受け取り、そのクエリを処理する専用ハードウェアを FPGA 上に実現する。また、文献 [7] では FPGA の再構成なしに異なるクエリを処理できるようにしている。

2013 年ごろからキーバリューストアの一種である Memcached を専用ハードウェアで高速化する試みが報告されるようになった。文献 [8] では、Memcached の応答時間のうち 64usec がネットワークインタフェースの処理、30usec が Linux カーネル（プロトコルスタック）、30usec が Memcached サーバと報告している。Memcached は計算に比して通信の比率が大きいため、1GbE インタフェースを備えた FPGA ボード上に Memcached 処理用のハードウェアを実装している。

文献 [9], [10] では、10GbE インタフェースを備えた FPGA ボードを用いてネットワーク性能を向上させている。さらに、Memcached のパケット分解、ハッシュ計算、メモリアクセス、応答パケット生成をパイプライン処理している。その結果、一般的なサーバ用 CPU と比べてスループットは 9.7 倍向上、電力当たりのスループットは 36.4 倍向上したと報告している。文献 [9], [10] は FPGA ボードでの単体動作を想定した Memcached アクセラレータであるが、文献 [11] では、Memcached のために SoC（System-on-Chip）を想定したアクセラレータを提案しており、この論文では Memcached 処理におけるソフトウェアとハードウェア処理の切り分けについて議論している。

以上のように、構造型ストレージのうち最も単純なデータ構造を持つ Memcached では、計算量に比して通信などの I/O が占めるウェイトが高い。このような用途には、GPU やメニーコアのような計算能力の高いデバイスよりは、FPGA のようにネットワーク（I/O）と計算が密に結合したデバイスが有利であると考えられる。

また、これまでに報告された構造型ストレージのハードウェア化は、主として、揮発性（ディスクへの永続化はしない）のキーバリューストア型である Memcached を対象にしてきた。一方、カラム指向型ストレージは、ウェブのインデックスを格納するためのストレージとして Google が BigTable を提唱して以来、HBase [4] や Cassandra [5] などのフリーな実装も登場し様々な分野で利用されているが、カラム指向型ストレージに関するハードウェア化の検討は、著者の知る限り報告されていない。そこで、本研究ではカラム指向型ストレージを FPGA を用いて高速化することを検討する。

3. カラム指向型ストレージ向けハードウェアキャッシュ

カラム指向型ストレージは単純なキーバリューストアと比べて豊富な機能を有しているため、Memcached 同様にカラム指向型ストレージを高速化することは有用であると考える。本論文では、BigTable に近い特徴を備えた HBase を対象にカラム指向型ストレージ向けハードウェアキャッシュ機構を検討する。

3.1 カラム指向型ストレージのデータ構造

典型的なカラム指向型ストレージのデータ構造を図 1 に示す。カラム指向型ストレージでは、キーバリューストアとは違い、行キー（Row#101 など）に対してカラム（名前と値の組、Col#A や Col#B など）を複数個管理できる。このようなカラムを動的に追加することができるが、カラムをグルーピングするためのカラムファミリーは動的に変更できない。また、

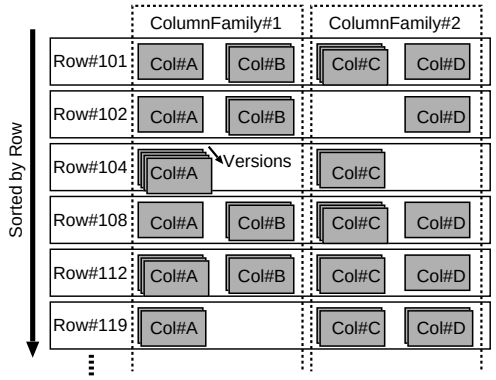

図 1　カラム指向型ストレージの
データ構造（Flat-wide 型）

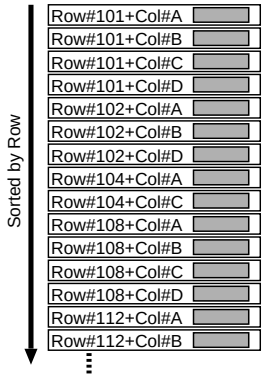

図 2　カラム指向型ストレージの
データ構造（Tall-narrow 型）

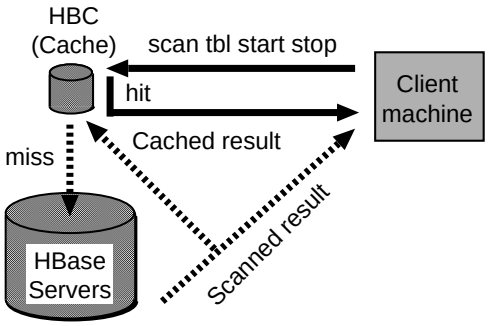

図 3　カラム指向型ストレージ向け
キャッシュ機構の動作（scan 要求）

カラムの値にはタイムスタンプが付与され、タイムスタンプを指定することで過去のリビジョンを取り出すこともできる。データは行キーによってソートされて格納される。したがって、Row#102 から Row#112 の間をスキャンするというような範囲（startRow と stopRow）を指定したデータ取得が可能であるという点もキーバリューストアには無い特色である。

図 1 のようなデータ構造を Flat-wide 型と呼ぶこともある。これは、単一の行が複数カラムを保持している様子が、縦方向が短く（flat）、かつ、横方向に長い（wide）からである。カラム指向型ストレージでは、これと同一のデータ構造を図 2 のような Tall-narrow 型として実現することもできる。図 2 では、元々の行キーとカラム名を連結したもの（例えば、Row#101+Col#A）をキーバリューストアの「キー」とし、そのカラム名に対応する値をキーバリューストアの「バリュー」として、キーバリューストアのようなデータ構造にすることもできる。この場合、カラム数は 1 となり（narrow）、その分行数が増える（tall）ため、Tall-narrow 型と呼ばれる。

このようにカラム指向型ストレージのデータ構造は、キーバリューストアに近い Tall-narrow 型に変換できる。ただし、本来のキーバリューストア型はハッシュ値をもとにデータの格納先をルックアップするが、Tall-narrow のカラム指向型ではデータは行キーによってソートされた状態で格納される。Tall-narrow 型と Flat-wide 型を比べると、前者のほうが構造がシンプルであり、ハードウェア化に向いている。本論文でも Tall-narrow 型を基本とする。

3.2　カラム指向型ストレージの単純な利用例

以降の説明のためにカラム指向型ストレージの単純な利用例を紹介する。以下の仕様を満たす短文投稿サイトを考えたい。

- UserID（短文の投稿者を識別）と TimeStamp（短文の投稿日時）を連結した文字列を行キーとする。
- 行キーで識別される行ごとに最大 256Byte のデータ（短文など）が保持される。
- 短文を投稿する際は、UserID と TimeStamp を連結して行キーを生成し、この行キーに対応するメモリ番地に最大 256Byte の短文を保存する。これを set 操作と呼ぶ。データは行キーによってソートされているため、同一ユーザから投稿された短文が投稿順（TimeStamp 順）にソートされて格納される。
- あるユーザが投稿したすべての短文を取得するには、startRow としてそのユーザの UserID、stopRow として UserID+1 を指定し、UserID と UserID+1 の間の行データをすべて取得する。これを scan 操作と呼ぶ。
- あるユーザがある一定期間内に投稿した短文を取得するには startRow として「そのユーザの UserID+スキャンを開始したい TimeStamp」、stopRow として「UserID+スキャンを終了したい TimeStamp」を指定して scan 操作を行えばよい。

本論文では、上記のような scan 操作へのレスポンス時間を改善するために、FPGA を用いたカラム指向型ストレージのハードウェアキャッシュ機構を検討する。具体的には、特定ユーザに関する最新の投稿データ（最新 64 件など）を 1 つのブロックとしてまるごとキャッシュする。キャッシュアーキテクチャはダイレクトマップ方式もしくは 2-way 程度のセットアソシアティブ方式を採用し、scan 操作が行われたユーザの UserID をもとにキャッシュインデックスを決定する。

Weblog（ブログ）などでも最初のページに表示されるのは一定量の最新の投稿だけであり、それより過去の投稿を見るには例えば「次の 10 件」もしくは「もっと見る」などのボタンを明示的にクリックする必要がある。したがって、最も頻繁に読み込まれるのは「一定量の最新の投稿」のみであり、この部分の scan 操作を高速化することにはメリットがあると考えられる。

3.3　提案キャッシュ機構 HBC の概要

図 3 にカラム指向型ストレージ向けハードウェアキャッシュ機構 HBC の概要を示す。

3.3.1　Scan 要求（HBC でヒット）の場合

クライアントが発した scan 要求はまず HBC へ転送され、要求されたブロックを HBC がキャッシュしていたらそのデータ（Cached result）をクライアントに返す（図 3）。HBC がヒットした場合、クライアントは HBase サーバを介さずにデータを取得できるためアプリケーションの高速化が期待される。

3.3.2　Scan 要求（HBC でミス）の場合

要求されたブロックを HBC がキャッシュしていない場合、オリジナルの scan 要求は従来どおり HBase サーバへ転送され、HBase サーバがスキャンした結果（Scanned result）をクライアントへ返す。この Scanned result は、HBC へも転送さ

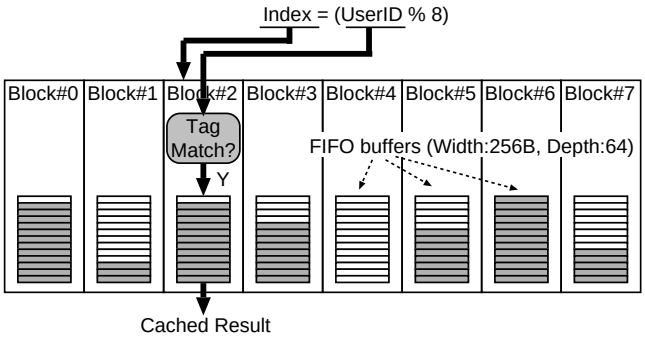

図 4　HBC のキャッシュアーキテクチャ

れ、HBC にもキャッシュされる（図 3）。

3.3.3　Set 要求（HBC でヒット）の場合

クライアントが発した set 要求は HBase に加え HBC にも転送される。HBC が該当ブロックをキャッシュしている場合は、HBC の内容を更新する。

3.3.4　Set 要求（HBC でミス）の場合

HBC が該当ブロックをキャッシュしていなければ何もしない。

3.4　提案キャッシュ機構 HBC の設計

3.4.1　ハードウェア構成

本研究では、HiTech Global 社の NetFPGA10G と呼ばれる FPGA ボード上に HBC を構築する。この NetFPGA10G ボードは、Xilinx 社の Virtex-5 XC5VTX240T FPGA 1 個、10GbE PHY（SPF+ポート）4 個、オンボード SRAM、オンボード DRAM などを装備している。また PCI Express Gen2 x8 インタフェースを有し、ホスト PC に搭載される。本 HBC では一定数の最新の連続した行を 1 つのブロックとしてホスト PC の主記憶上にキャッシュする。ホスト PC の主記憶と NetFPGA10G ボード間のデータのやりとりには PCI Express を介した DMA 転送を行う。試験的に、ホスト PC の主記憶と NetFPGA10G ボード間で DMA 転送を行ったところ、100MByte/sec 強の転送レートを実現できた。

3.4.2　HBC のキャッシュブロックの設計

HBC のキャッシュアーキテクチャを図 4 に示す。本設計では、行ごとのデータサイズ（同一行に属す全カラムのデータサイズの合計）S_{row} は 256Byte とした。HBC のキャッシュブロックサイズ（HBC キャッシュの 1 ブロックに含まれる行の数）S_{blk} は 64 [Row] とした。したがって、キャッシュブロック 1 個当たりのメモリサイズは $(S_{row} \times S_{blk})$ [Byte] となる。HBC では、ホスト PC の主記憶上にこのようなキャッシュブロックを多数持つことを想定している。HBC が管理するキャッシュブロックの個数を N_{blk} とし、説明の都合上、図 4 では N_{blk} は 8 個としている。

HBC では、特定ユーザに関する最新の投稿データ（最新 S_{blk} 件）を 1 つのキャッシュブロックとしてまるごとキャッシュする。そのため、個々のキャッシュブロックは要素サイズ S_{row} [Byte]、要素数 S_{blk} の FIFO バッファとして実装する。前述のとおり、行キーに TimeStamp が含まれるため、新規追加される行は FIFO バッファの最後尾に追加され、一方で FIFO バッファの先頭にある最も古いデータは自動的に上書きされる。つまり、S_{blk} 個より古い行は自動的に HBC から消去される（この行にアクセスするには HBase に問い合わせる必要がある）。

3.4.3　HBC のキャッシュブロックのルックアップ

キャッシュアーキテクチャはダイレクトマップ方式もしくは 2-way 程度のセットアソシアティブ方式を採用し、scan 操作が行われたユーザの UserID をもとにキャッシュインデックスを決定する。ここでは単純なダイレクトマップ方式の例を説明する（図 4）。あるユーザ（UserID は U とする）の最新データに対して scan 要求があったとする。この場合、まず $(U \bmod N_{blk})$ を計算し、キャッシュインデックス（0 から $N_{blk}-1$ までの範囲のいずれか）を得る。キャッシュインデックスから該当キャッシュブロックを見つけ、そのキャッシュブロックが現在保持しているデータの UserID と U を比較し、もし一致したら、そのキャッシュブロックにユーザ U の最新データがキャッシュされているということになる。

3.4.4　HBC のキャッシュブロックの置き換え

クライアントからの scan 要求に対し、HBC がキャッシュミスを起こすと、その要求は HBase へ送られ、そこで該当ブロックをスキャンして結果をクライアントへ返す。このスキャン結果は HBC にも送られるため、このタイミングで HBC の入れ替えが生じる。上記はダイレクトマップ方式を想定したが、セットアソシアティブ方式を用いる場合は擬似 LRU などの方法を用いてキャッシュブロックを入れ替える必要がある。

4.　予備評価（性能見積り）

4.1　HBC の理想的な性能モデル

HBC への get 操作、set 操作、scan 操作がすべて HBC のキャッシュにヒットする理想的な場合を想定し、以下の実行時間を計算する。

- T_{get1}: HBC への get 操作に要する時間。
- T_{set1}: HBC への set 操作に要する時間。
- $T_{scan1(N)}$: HBC への scan 操作に要する時間。ただし、N は HBC にヒットするブロック数とする（後述）。

これらを計算するために、さらに以下のパラメータを導入する。

- T_{cyc}: HBC のサイクル時間 [nsec]。ここでは、簡易的なクロックサイクル時間の見積りのために、HBC のキャッシュブロック（FIFO バッファ）を小規模にしたものを FPGA 内のブロック RAM を用いて実装した。設計には Verilog HDL を使用し、論理合成には Xilinx ISE Design Suite 13.4 を用いた。HBC モジュールを論理合成した結果、最大動作周波数は 221MHz となったため、ここでは T_{cyc} を 5nsec と想定した。なお、本実装は簡易的なものであるため実際の動作周波数は 200MHz を下回る可能性もある。より詳細な設計は今後の課題である。
- C_{dma}: ホストの主記憶から FPGA へのメモリ転送時間 [cycle/Byte]。実機実験より NetFPGA10G ボードの DMA 転送能力は 100MByte/sec 強となったため、1Byte 転送するのに 10nsec を要する。よって C_{dma} は 2 [cycle/Byte] とする。
- C_{chk}: 該当ブロックを HBC がキャッシュしているかどうか判定するための時間 [cycle]。HBC の RTL シミュレーショ

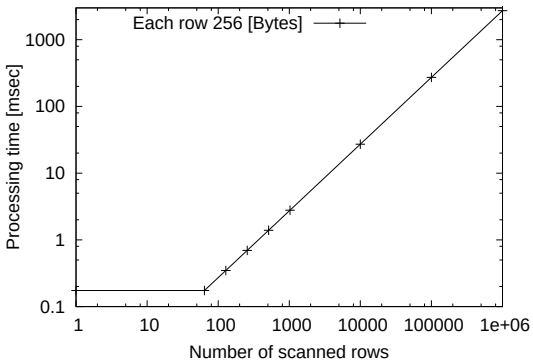

図 5　HBC における scan 操作の実行時間（ヒット率 100%）。XY 対数軸。

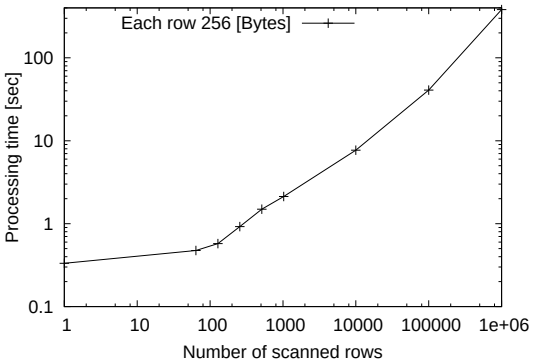

図 6　HBase における scan 操作の実行時間（実測値）。XY 対数軸。

ン結果より、ここでは 2 [cycle] とする。

- S_{row}: 行ごとのデータサイズ（同一行に属す全カラムのデータサイズの合計）[Byte]。今回の HBC の実装では 256Byte とする。
- S_{blk}: HBC のキャッシュブロックサイズ（HBC キャッシュの 1 ブロックに含まれる行の数）[Row]。今回の実装では 64 [Row] とする。
- B_{bus}: HBC のメモリデータバス幅 [Byte]。今回の実装では 64-bit バスを想定しているため、8Byte とする。

get 操作および scan 操作ではホストの主記憶から FPGA へのメモリ転送（DMA 転送）が生じる。このようなメモリ転送は $(S_{row} \times S_{blk})$ [Byte] 単位の DMA 転送を用いて行うものとする。この場合、get 操作では不要な行データまで読み込んでしまうため無駄が多いが、DMA を $(S_{row} \times S_{blk})$ [Byte] 単位に固定化することでハードウェアを単純化できる。

get 操作に要す時間は以下のとおりである。

$$\begin{aligned} T_{get1} &= (C_{chk} + S_{row}S_{blk}C_{dma} + S_{row}/B_{bus})T_{cyc} \quad (1) \\ &= (2 + 256 \times 64 \times 2 + 256/8) \times 5 \\ &= 0.16401[\text{msec}] \end{aligned}$$

HBC への set 操作は以下の 3 ステップで実行される。1) 書き込み対象のキャッシュブロックをホストの主記憶から FPGA へ DMA 転送する。2) FPGA 上に取得したキャッシュブロックのうち、該当エントリのみを更新する。3) 更新済みのキャッシュブロックをホストの主記憶に書き戻す。つまり、ホスト $\mapsto$ FPGA への DMA 転送、および、FPGA $\mapsto$ ホストへの DMA 転送が生じる。よって、set 操作に要す時間は以下のとおりとなる。

$$T_{set1} = 2T_{get1} = 0.32802[\text{msec}] \quad (2)$$

scan 操作に関しては、N を HBC にヒットするブロック数とするとき、実行時間は以下のように計算できる。

$$\begin{aligned} T_{scan1(N)} &= (C_{chk} + S_{row}S_{blk}NC_{dma} + S_{row}S_{blk}N/B_{bus})T_{cyc} \\ &= (2 + 256 \times 64 \times N \times 2 + 256 \times 64 \times N/8) \times 5 \\ &= (10 + 174080 \times N)[\text{nsec}] \quad (3) \end{aligned}$$

式 3 を用いて HBC における scan 操作の実行時間（ヒット率 100%の場合）を計算した。X 軸、Y 軸を対数としたグラフを図 5 に示す。グラフより scan 対象の行数に応じて実行時間が線形に増加していることが分かる。

4.2　HBase の性能

HBC の性能比較相手として、ベースラインの HBase の実行時間を実機で測定した。4.1 節で述べたとおり、行ごとのデータサイズ S_{row} は 256Byte とし、以下の実行時間を実機測定した。

- T_{get0}: HBase に対する get 操作に要する時間。
- T_{set0}: HBase に対する set 操作に要する時間。
- $T_{scan0(N)}$: HBase に対する scan 操作に要する時間。ただし、N は HBase から読み込むブロック数とする。

実験に用いたシステムの諸元を表 1 に示す。今回は 1 台のマシンを用いて HBase をスタンドアローンモードで動作させた。

表 1　HBase の実験環境

プロセッサ	Intel Xeon CPU E5-1620 v2 @ 3.70GHz
メモリ	128GByte
OS	Linux x86_64 2.6.32
HBase	0.94.16 (Standalone mode)

表 1 の環境を用いて、get 操作および set 操作に要する時間を測定した。結果は以下のとおりである。

$$T_{get0} = T_{set0} = 0.333[\text{sec}] \quad (4)$$

同様に、N を HBase から読み込むブロック数とするとき、scan 操作に要する時間 $T_{scan0(N)}$ も実機測定した。測定結果を図 6 に示す。HBC のキャッシュヒット率が 0% のときの性能がこれに相当する。HBC の性能見積り（図 5）と比べると、圧倒的に実行時間が長いことが分かる。つまり、HBase に対する scan 操作（get 操作）が HBC にヒットすると大幅な性能向上が期待できる。しかし、図 5 の結果はすべての scan 操作が HBC にヒットしたという理想的な状況下での性能見積りである。容量の点からキャッシュが 100% ヒットするとは考えにくく、また、HBC でキャッシュミスした場合の実行時間が圧倒的に長いため、少量でもキャッシュミスが生じると性能向上は大幅に小さくなると予想される。これについては次節で検討する。

4.3　キャッシュミスを考慮した HBC の性能見積り

本節では、HBC にキャッシュミスがあることを考慮に入れ

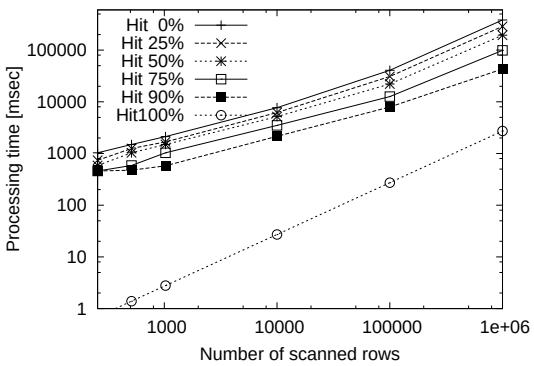

図 7　HBC における scan 操作の実行時間（ヒット率を変化させた評価）。XY 対数軸。

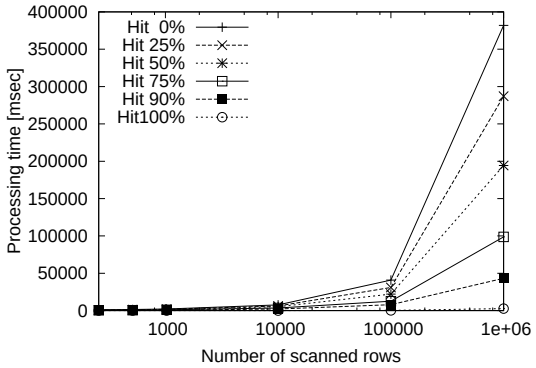

図 8　HBC における scan 操作の実行時間（ヒット率を変化させた評価）。X 対数軸。

て、性能モデルを再構築する。以下の実行時間を計算する。

- T_{get2}: HBC への get 操作に要する時間。
- T_{set2}: HBC への set 操作に要する時間。
- $T_{scan2(N_{hit},N_{miss})}$: HBC への scan 操作に要する時間。

ただし、N_{hit} は HBC にヒットするブロック数、N_{miss} は HBase が処理するブロック数とする。

get 操作については、HBC にヒットする場合は $T_{get2} = T_{get1}$ となり、ミスすると $T_{get2} = T_{get0}$ となる。

set 操作については、HBC への書き込みと HBase への書き込みが同時に実行されると仮定しているため、$T_{set2} = \max(T_{set0}, T_{set1})$ となる。

scan 操作については、HBC への scan と（ヒットしなかった部分に関する）HBase アクセスが逐次的に実行されると仮定すると実行時間は以下のようになる。

$$T_{scan2(N_{hit},N_{miss})} = T_{scan0(N_{miss})} + T_{scan1(N_{hit})} \qquad (5)$$

ここで、N_{hit} の値をヒット率が 0%、25%、50%、75%、90%、100%になるように変化させて $T_{scan2(N_{hit},N_{miss})}$ を計算した。

実行時間のグラフを図 7 と図 8 に示す。図 7 では Y 軸を対数軸としているが、図 8 では Y 軸は線形軸となっている。例えば、図 8 で行数が 1e+06 個のとき、HBC のヒット率が 0%のときの実行時間を 1 とすると、ヒット率 25%、50%、75%、90%の実行時間はそれぞれ 0.75、0.51、0.26、0.11 であり、HBC のヒット率が上がるにしたがい大幅な性能向上が期待できるような結果となった。

5. まとめと今後の課題

本研究では、FPGA を用いてカラム指向型データベースを高速化することを検討した。具体的には、10GbE のネットワークインタフェースを持つ FPGA ボード上にカラム指向型データベースのハードウェアキャッシュ（HBase Cache、HBC）を構築することを検討した。この FPGA ボードは PCI Express 経由でホスト PC に接続され、本 HBC では一定数の最新の連続した行を 1 つのブロックとしてホスト PC の主記憶上にキャッシュする。キャッシュにヒットすれば、ソフトウェアによるカラム指向型データベース処理を行わずに HBC がキャッシュしているデータを直接利用できるため、データアクセスを高速化できる。本論文では、データベースへのスキャン要求が HBC にヒットした場合、しなかった場合のスキャン操作に要す実行時間をモデル化し、HBC によって大幅な性能向上が可能であること示した。ただし、本論文では、カラム指向型アクセラレータ開発の第一歩として FPGA ボードを用いた実機動作までは行わず、論理合成およびハードウェアシミュレータによる評価までを行った。また、通信部分を含めたカラム指向型アクセラレータ全体の設計と実装は今後の課題である。

謝辞　本研究の一部は、JST 戦略的創造推進事業さきがけ「多様な構造型ストレージ技術を統合可能な再構成可能データベース技術」の補助による。

文　献

[1] Pramod J. Sadalage and Martin Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley, 2012.
[2] D. Interactive, “Memcached - A Distributed Memory Object Caching System”. http://memcached.org/.
[3] S. Sanfilippo, “Redis”. http://redis.io/.
[4] A.S. Foundation, “The Apache HBase Project”. http://hbase.apache.org/.
[5] A.S. Foundation, “The Apache Cassandra Project”. http://cassandra.apache.org/.
[6] R. Mueller, J. Teubner, and G. Alonso, “Streams on Wires: A Query Compiler for FPGAs,” Proceedings of the International Conference on Very Large Data Bases (VLDB’09), pp.229–240, Aug. 2009.
[7] B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Iyer, B. Brezzo, D. Dillenberger, and S. Asaad, “Database Analytics Acceleration Using FPGAs,” 2012.
[8] S.R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, “An FPGA Memcached Appliance,” Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA’13), pp.245–254, Feb. 2013.
[9] M. Blott, K. Karras, L. Liu, K. Vissers, J. Baer, and Z. Istvan, “Achieving 10Gbps Line-rate Key-value Stores with FPGAs,” Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud’13), June 2013.
[10] M. Blott and K. Vissers, “Dataflow Architectures for 10Gbps Line-rate Key-value-Stores,” Proceedings of the IEEE Symposium on High Performance Chips (HotChips’13), Aug. 2013.
[11] K. Lim, D. Meisner, A.G. Saidi, P. Ranganathan, and T.F. Wenisch, “Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached,” Proceedings of the International Symposium on Computer Architecture (ISCA’13), pp.36–47, June 2013.