Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 7, Number 1, pages 12–19, January 2018

An FPGA Implementation of a Parallel Column Sort Algorithm with Off-chip DRAMs

Naoaki Harada, Koji Nakano, and Yasuaki Ito
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527 Japan
Email: {harada, nakano, yasuaki}@cs.hiroshima-u.ac.jp

***Abstract*—The main contribution of this paper is to show an FPGA implementation of a parallel sorting algorithm with off-chip DRAMs. In the implementation, we use the idea of the column sort and multiple data sets stored in the distinct DRAMs are concurrently sorted by FIFO-based pipeline sorters in the FPGA. We have implemented the proposed circuit in a Xilinx Virtex Ultra Scale+ family FPGA XCVU9PL2FLGA2104E with eight off-chip DRAMs. The experimental results show that the proposed implementation can achieve a speed-up factor of 84 over the sequential CPU implementation by quick sort.**

***Index Terms*—parallel sorting algorithm; hardware algorithm; FPGA; DRAM**

I. はじめに

FPGA とは，製造後にハードウェア記述言語によって，顧客または設計者が任意に論理回路を再構成可能とする LSI である．FPGA は低価格及び再構成可能といった特徴を保持しており，画像処理や教育のような頻繁にアップデータを行う必要のある分野において，幅広く使用されている．最新の FPGA アーキテクチャは CLB，ブロック RAM，DSP，IOB 及びそれらを接続するスイッチマトリックスや配線セグメントで構成されている．これらを並列化，パイプライン化して動作させることで FPGA は計算の高速化を可能とする．

ソートはデータベース操作，画像処理，統計的方法論などのようなコンピュータ工学における重要な課題の一つである．そのため，数多くのアルゴリズムが研究されている [1], [2]．ただし，多くのソートの FPGA 実装は巨大なデータ列のソートを対象としていない．そこで，本論文では巨大なデータ列を対象としたソート手法の効率的なハードウェア実装を提案している．ソートの FPGA 実装の例として，FIFO を用いたマージソーターは $\log_2 K$ 個のコンパレータと総容量 $2K + \log_2 K - 2$ の FIFO を用いて，K 個のキーがソート可能なことが知られている [3]．また，論文 [5] では，深さの小さい FIFO を複数使用することで，使用する FIFO の効率化の手法が提案されている．他に近年では，論文 [4] で木構造のソーターの効率的な FPGA 実装が提案されている．これらの実装は，FIFO を多く用いるため巨大なデータ列のソートに対しては向いていない．また，論文 [6] では，ソーティングネットワークの FPGA 実装が提案されている．この実装では多くのコンパレータを使用するため巨大なデータ列をソートできない．一方，参考論文 [7] では巨大なデータ列に対するマージソートをベースとした FPGA 実装が提案されている．この手法では，まず，FIFO を用いたマージソーターによってある程度の大きさまでソートを行うことでソート済みのデータ列を作る．その後，全てのソート済みデータ列の先頭のデータを木構造のコンパレータを用いて比較し，最小のデータのみを出力する．これを繰り返すことで，全体のソートを行う．しかし，この手法では Fig. 1 のように使用可能な出力ポートが 1 つであり，並列化による高速化を行うことが難しい．そこで，本研究では，複数の出力ポートを用いた column ソートの FPGA 実装を行っている．

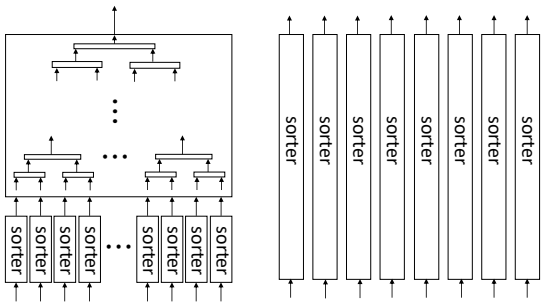

Fig. 1. 左：既存手法　　右：提案手法

column ソートは Leighton によって提案された N 個のデータ列（$r \times s$ の行列）をソートする手法である [9]．この手法は，r が偶数，$r \geq 2s^2$，r を s で割り切れるという 3 つの制約を満たす必要がある．column ソートは下記の 8 つの Step に従って，ソートを複数回行うことで，N 個のデータを完全にソートすることが可能である．ただし，Step 1,3,5,7 はそれぞれ s 個のデータ列をソートする動作を行う．

Step 2:　Fig. 2 に示すように，それぞれの列を連続する r/s 個の行にラスタスキャン順にマッピングする．

Step 4:　Step 2 の逆の操作を行う．つまり，連続する r/s 個の行を単一の列にマッピングする．

Step 6:　Fig. 3 に示すように，それぞれの列を $r/2$ 個分下方向にシフトする．その際に，それぞれの列の下半分である $r/2$ 個は右側にある次の列の上半分に移動する．ただし，シフト操作が行われる際には，最も左の列の上半分には $-\infty$ が挿入され，最も右の列の下半分には ∞ が挿入される．

Step 8:　Step 6 の操作の逆を行う．つまり，それぞれの列を $r/2$ 個分上方向にシフトする．

前述からわかるように，Step1,3,5,7 における r 個のデータからなるデータ列の s 回のソートは依存関係がなく，完全に並列に行うことが可能である．そこで，FPGA に FIFO を用いたマージソーターを複数用意することによって並列にソートを行う手法を提案する．また，論文 [10] で示され

$$\begin{bmatrix} a & g & m \\ b & h & n \\ c & i & o \\ d & j & p \\ e & k & q \\ f & l & r \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

Fig. 2. Step 2

$$\begin{bmatrix} a & g & m \\ b & h & n \\ c & i & o \\ d & j & p \\ e & k & q \\ f & l & r \end{bmatrix} \xrightarrow{\text{Step 6}} \begin{bmatrix} -\infty & d & j & p \\ -\infty & e & k & q \\ -\infty & f & l & r \\ a & g & m & -\infty \\ b & h & n & -\infty \\ c & i & o & -\infty \end{bmatrix}$$

Fig. 3. Step 6

ているソーターを用いることで Step5,6,7,8 を一つの Step にまとめることが可能であり，その FPGA 実装も提案する．結果として，クイックソートの逐次実装と比較して 256M 個の 36bit データでは 63-84 倍，512M 個の 36bit データでは 40-62 倍の高速化を達成した．また，提案実装にかかる clock cycle 数は既存実装と比較して約 0.152-0.203 倍を達成した．

本論文の構成について，第 2 節では d-sort アルゴリズムを用いた column ソートについて説明を行う．第 3 節では column ソートを行う際のメモリの読み書きについて述べ，第 4 節ではマージベースのソートの FPGA 実装を説明する．第 5 節では提案手法を Virtex Ultra scale+ファミリ に実装し評価実験を行った結果を示す．

II. d-SORT アルゴリズムを用いた COLUMN ソート

論文 [10] では d-sort アルゴリズムという全体はある程度ソートされているが一部が未ソートであるようなデータ列に対するソート手法を提案している．この手法を用いることで，column ソートの Step5,6,7,8 を一つのステップにまとめることが可能である．初めに，全体はある程度ソートされているが一部が未ソートであるようなデータ列の定義について説明する．まず，データ列に対して未ソート間の最大距離 d を定める．以後，未ソート間の最大距離が d のシーケンスを d-sorted シーケンス，その FPGA 実装を d-sorter と表す．ここで n 個のデータ列を $x_0, x_1, \ldots, x_{n-2}, x_{n-1}$ とすると，d-sorted シーケンスは $j-i \geq d$ を満たすどのような i と j に対しても $t_i < t_j$ を維持するデータ列と定めることができる．また，$t_i > t_j (i < j)$ という大小関係のとき $j-i < d$ を満たすことは明らかである．Figure 4 は 4-sorted シーケンスの例である．$t_1 > t_4$ であるため，3-sorted シーケンスではない．また，$j-i \geq 4$ を満たすどのような i と j に対して，$t_i < t_j$ を維持しているため，4-sorted シーケンスであることがわかる．

次に，論文 [10] で示されている d-sorted シーケンスのマージベースのソート手法を説明する．n 個のデータを持つデータ列を T と置く．初めに，データ列 T をそれぞれが d 個の

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
0	4	1	2	3	5	6	10	7	8	9	11

Fig. 4. 4-sorted シーケンス

データを持つ $\frac{n}{d}$ 個のサブシーケンス $T_0, T_1, \ldots, T_{\frac{n}{d}-1}$ に分割する．次に，それぞれのサブシーケンスをソートする．その後，2 つのサブシーケンス T_i と T_{i+1} のマージを $i = 0$ から $i = \frac{n}{d} - 2$ まで逐次に行っていく．Figure 5 では 4-sorted シーケンスのソートの動きを示している．

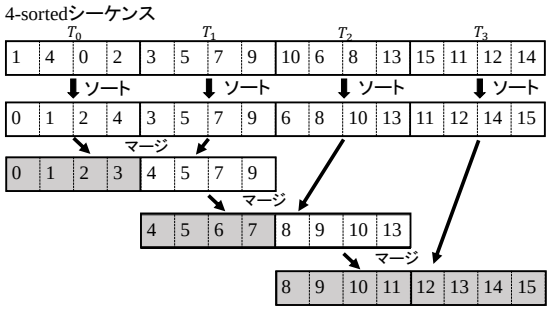

Fig. 5. 4-sorted シーケンスのマージベースのソート

d-sorted シーケンスをソートする手法を用いることで Step5,6,7,8 を一つのステップにまとめることが可能であることを示す．これは 01 原理を考えることで正しくソート可能であることがわかる．01 原理とはソートアルゴリズムが 0 と 1 で構成された全てのパターンのセットをソート可能であれば，いかなるデータであってもソート可能であるというものである．0 と 1 をソートすることはソートが完了したときに，0 と 1 の 2 つの塊にデータ列を分けることに等しい．column ソートでは Step1,2,3,4 によって 0 と 1 が混在している箇所の大きさを小さくしている．その後，Step5,6,7,8 によって 0 と 1 の 2 つの塊にソートすることで，全体のソートを行っている．Figure 6 は Step2 と 3 の実行後を示している．また，Fig. 7 は Step4 の実行後を示しており，0 と 1 の混在部分の縦の長さが最大でも s^2 であることがわかる．従って，Step4 の実行後では少なくとも s^2 離れたデータどうしの大小関係が正しいことになる．つまり，$j-i \geq s^2$ を満たすどのような i と j に対して，$t_i < t_j$ を維持しているため，Step4 の実行後のデータ列は d-sorted シーケンス ($d = s^2$) である．従って，d-sorter($d = s^2$) を用いることで Step5,6,7,8 を一つの Step に変更することが可能である．以後，Step5,6,7,8 を一つの Step に変更したものを Step5' とする．ただし，複数の d-sorter で並列にソートを行う場合，それぞれの DRAM の境界部分のマージが必要となる．この時は，それぞれの DRAM の境界部分の右端の列の下 s^2 個とその隣の DRAM の左端の列の上 s^2 個をマージすることでデータ列を完全にソート可能である．これは d-sorted シーケンスの定義から明らかである．

III. COLUMN ソートを行う際のメモリの読み書き

本節では，本論文で提案する column ソートにおける効率的なメモリの読み書きについて説明する．本論文で提案する手法では，FPGA を用いて Step1,3,5,7 におけるデータ列

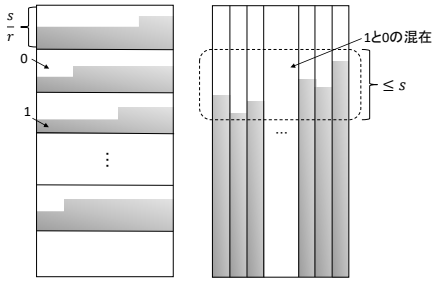

Fig. 6. 01 原理の場合，左：Step2 の実行後，右：Step3 の実行後

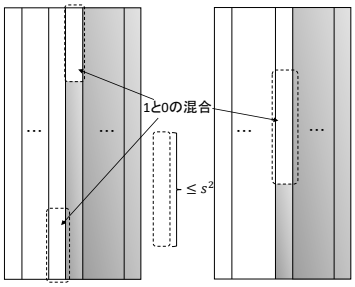

Fig. 7. 01 原理の場合，Step4 の実行後

のソートを行う．その際に，並列かつ高速にデータを入出力する必要がある．従って，FPGA の入出力をメモリから高速に読み書きすることをこれから考える．また本論文では，巨大なデータ列をソート対象としているため記憶容量の小さい SRAM ではなく，DRAM を想定している．DRAM は SRAM と比較して，アクセス速度が遅いという特徴があるが，バースト転送と呼ばれる高速にアクセス可能な方法が存在する．バースト転送とは，連続したアドレスへの読み書きを行う場合，ランダムなアドレスへの読み書きより高速にデータの読み書きを可能とする転送方法のことである．従って，連続したアドレスへの読み書きを考慮した効率的なデータの読み書きを考える．使用する DRAM の個数はデータを読み書きする空間を別々にしたいため，Fig. 8 のように並列数の 2 倍の個数を用意する．ソーターの FPGA 実装については第 4 節で説明する．また，以後説明で用いる図では行のアドレスが連続であるようにアドレスのマッピングがされていると仮定している．

まず, FPGA に実装するソーターの数を並列数 M と仮定する．Step1,3,5,7 で行われるソートにおいて，それぞれの列のソートは完全に独立しているため，並列にソートが可能である．また, 用いる DRAM とそれぞれの行列を Di_{pq}(i=DRAM の番号，p=行番号,q=列番号) とする．

具体例として，1024 個のデータ列（128×8 の行列）を 4 並列でソートすることを考える．初めに Step2 を考える．ここで Step2 の実行後のデータを Fig. 9 のように置くとする．Figure 9 を見てわかるように，初めに，A1,A17,A33,A49, 次に B1,B17,B33,B49, その次に C1,C17,C33,C49 という風にデータは 4 並列に入力される．Step3 で縦に A1 から A128 をソートすることを考えると，連続したアドレスのデータの読み書きを行うためにバッファーを用意する必要がある．連続する r/s 個の行の上の行から W 個の行分データを格納

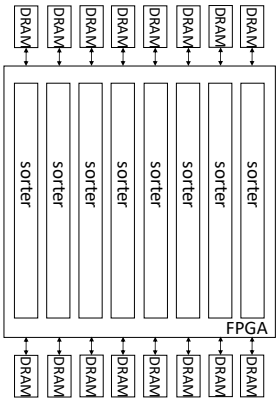

Fig. 8. 8 並列の場合の column ソートの全体図

するために横 $\times W\times$ 並列数の容量を持つバッファーを用いる．そうすると，連続に書き込むデータ数は $W\times$ 並列数となる．

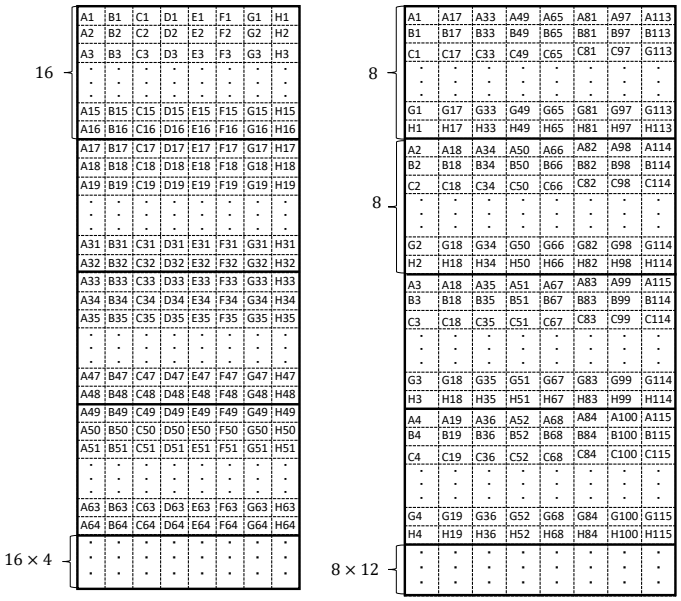

Fig. 9. Step2 と Step4 の実行後のデータ

今回の例では，$8 \times 4 \times 4 = 128$ の容量のバッファーを用いるとする．そうした場合，実際の DRAM へのデータの書き込みは Fig. 10 のようになる．実際の書き込みは，Fig. 10 にあるように $W\times$ 並列数 $= 4 \times 4 = 16$ 連続のアドレスに書き込んだ後，右の列に移動して再び連続で書き込む．これを s/M 回繰り返す．最上部の r/s 行が埋まると，次の r/s 行に同じ方法で書き込みを行っていく．これを繰り返すことで DRAM への書き込みを行う．

次に，Step3 と Step4 でのデータ読み書きを考える．Step3 における FPGA への入力を行うためのデータ読出しは，Fig. 10 より単純に縦に連続に読み出せばいいことがわかる．従って，Step4 のための書き込みについて考える．まず，Fig. 9 の左の図を Step3 の実行後のデータ列とする．そうすると Fig. 9 の右の図は Fig. 9 の左の図のデータが Step4

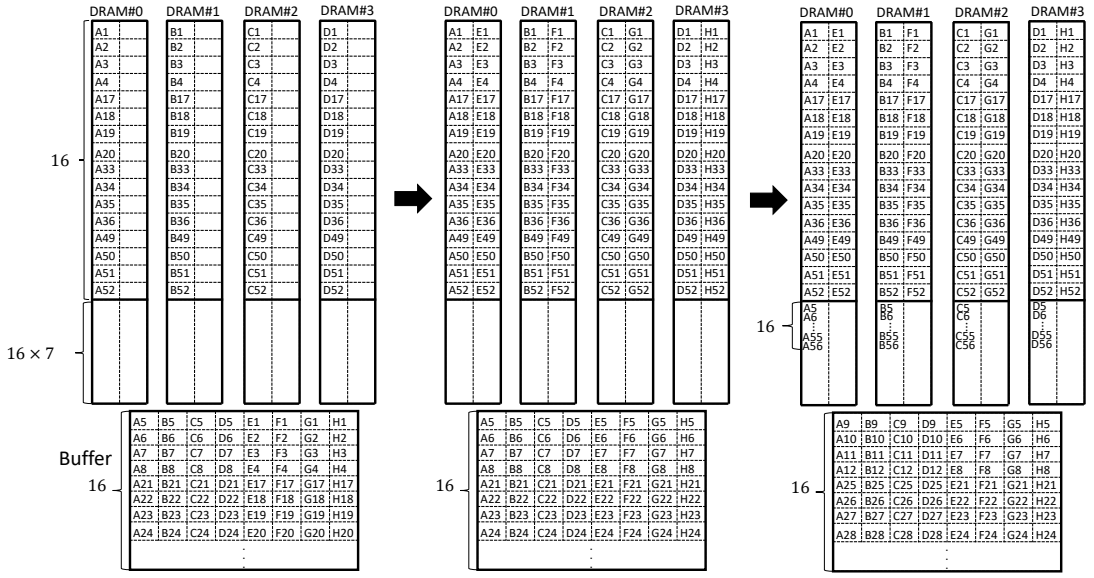

Fig. 10. Step2 の実行後の DRAM へのデータ書き込みの一部

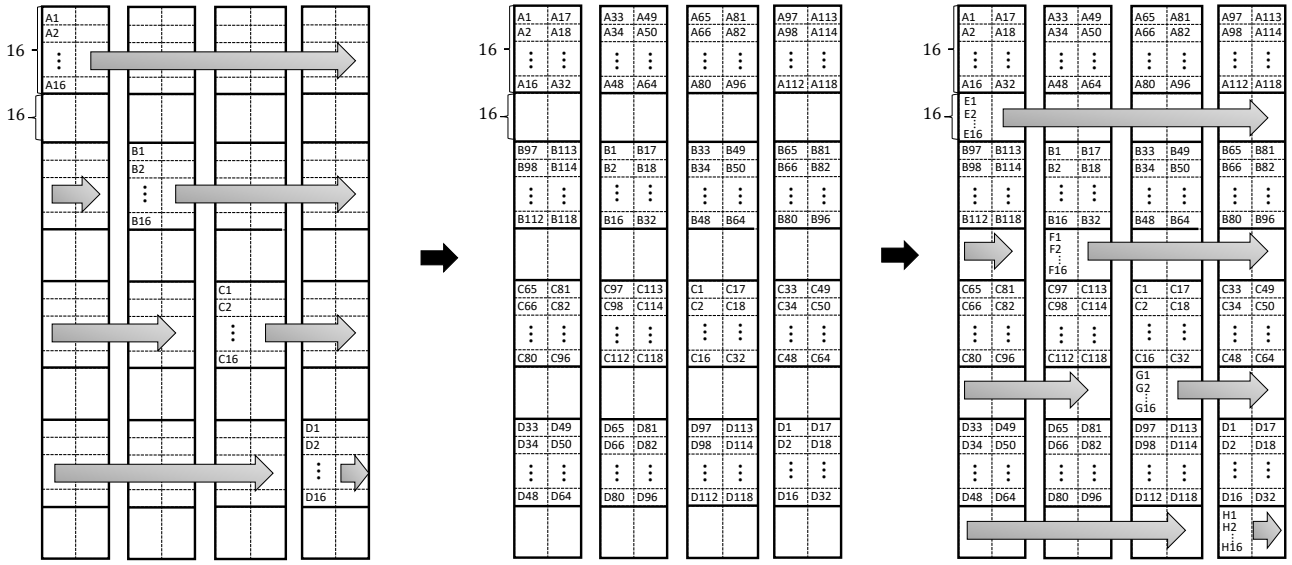

Fig. 11. Step4 の実行後の DRAM へのデータ書き込みの一部

によってどのようにマッピングされるかを示した図と見なせる．その場合，実際の DRAM へのデータの書き込みは Fig. 11 のようにするのが効率が良い．ここで，Fig. 9 からわかるように，Step3 で並列にデータを読み出してソートした場合，それぞれの列の初めの r/s 行は自身の列に Step4 でマッピングされる．次の r/s 行は自身の右隣りの列にマッピングされることとなる．例えば，A1 から A16 は最も左の列に Step4 でマッピングされ，A17 から A32 は左から 2 番目の列にマッピングされる．従って，Fig. 11 のようにデータの書き込みは $Di_{p0}(i = 0, 1, ..., M-1)(p = ir/M)$ から r/s 行分書き込む．次は $Di_{p1}(i = 0, 1, ..., M-1)(p = ir/M)$ から r/s 行分データを書き込む．これを繰り返していく．従って，データを r/s 個連続して書き込むことになる．また次のデータ列のソートが始まると，Fig. 11 の右端の図のように $Di_{p0}(i = 0, 1, ..., M-1)(p = ir/M + r/s)$ から同じように書き込みを繰り返す．

Step5 と Step6 でのデータ読み書きを考える．Step5 における FPGA への入力を行うためのデータ読出しは，Fig. 9 の右の図からわかるように A1,A2,...,A16,B1,B2,...B16,...H16 をソートすることになる．従って，Fig. 11 より $r/s \times s/M = r/M$ 個連続で $D0_{00}$ から縦に読み出した後，1 つ右の DRAM の $D1_{p0}(p = r/M)$ から r/M 個連続で読み出す．これを全ての DRAM で繰り返すことで可能である．例えば，Fig. 11 であれば，A1,A2,...,A16,E1,E2,...E16 と 32 個連続で読み出した後，B1,B2,...,B16,F1,F2,...F16，その次は C1,C2,...,C16,G1,G2,...G16 という風に 32 個連続で読み出す．これを繰り返すことで，下方向にジグザグに読み出していく．最も下の行まで読み出したら，$D0_{01}$ から同じ

ことを繰り返す．このようにして全てのデータを読み出す．また，Step6 にあたるデータの書き込みは r 個連続で，$D0_{p0}$，$D0_{p1}$，$D0_{p2}(p = 0, 1, 2..., r-1)$ のようにそれぞれの DRAM の左列から順番に書き込んでいく．これをそれぞれの DRAM で並列に行うことになる．Step6 の $r/2$ 個のシフトを考慮するため，次の読出しの際には中央部分にあたる $Di_{p0}(i = 0, 1, 2...M-1)(p = r/2)$ から始める．

最後に，Step7 と Step8 でのデータ読み書きを考える．Step7 における FPGA への入力を行うためのデータ読出しは $D0_{p0}(p = r/2, r/2+1, ...r-1)$ から $D0_{p1}(p = 0, 1..., r/2-1)$ のようにそれぞれの DRAM で読出しを行う．その後，$D0_{p1}(p = r/2, r/2+1, ...r-1)$ から $D0_{p2}(p = 0, 1..., r/2-1)$ のように右の列に移動しつつそれぞれの DRAM で全てのデータを読み出すまで繰り返す．また，Step8 に当たる書き込みは Step7 での読み出しと同じアドレスに書き込むことになる．

また，d-sorter を用いて Step5,6,7,8 を一つの Step5' として行うことを考える．データの読出しは Step5 における FPGA への入力を行うためのデータ読出しと同じである．書き込みはそれぞれの DRAM に対して並列に $D_{i0}(i = 0, 1, ...r)$ から書き込んでいき，初めの列が終われば，次の左の列に $D_{i1}(i = 0, 1, ...r)$ から書き込む．これをそれぞれの DRAM で繰り返す．また，並列にソートを行う場合は，最後に s^2 個と s^2 個のデータのマージが必要となる. これは単純にそれぞれの連続するアドレスからデータを読出し，FPGA に入力する．その後，FPGA からの出力を元のアドレスに書き込むこととなる．

IV. ソーターのハードウェア実装

本節では, FIFO ベースのマージソーターの FPGA 実装を示す．初めに，d 個のデータ列をソートする d-merge-sorter[11] を説明する．その後，d 個のソート済み列のマージを行う d-sliding-merger[10] を説明する．d-sorted シーケンスをソートする d-sorter は d-merge-sorter と d-sliding-merger で構成されている．

初めに，d-merge-sorter の説明を行う．d-merge-sorter は $\log d$ 個の k-merger を 1-merger, 2-merger, 4-merger, ..., $\frac{d}{2}$-merger という形で順次接続した構造をしている．Figure 12 は 8-merge-sorter を示している．

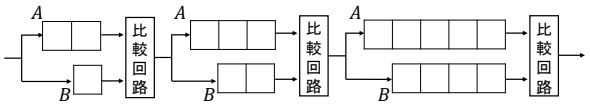

Fig. 12. :8-merge-sorter

k-merger は k 個のデータからなる 2 つのソート済みデータ列を $2k$ 個のデータからなる 1 つのソート済みデータ列にマージする回路である．2 個の FIFO と 1 つのコンパレータから構成される．それぞれの FIFOA, B は深さが $k+1, k$ となっており，その深さ分のデータを保持することが可能である．2 つの FIFO の先頭のデータがコンパレータに入力され，コンパレータは 2 つのデータを比較し，小さい方のデータの出力を行う．入出力は 1 クロックごとに行われる．

次にマージの動作について説明する．n 個の入力データ列を $T = \langle t_0, t_1, \ldots, t_{n-1} \rangle$ と置く．ここで，k 個ごとの部分列をそれぞれ $T_0, T_1, \ldots, T_{\frac{n}{k}-1}$ とする．それぞれの部分列 $T_i = \langle t_{i \cdot k}, t_{i \cdot k+1}, \ldots, t_{i \cdot k+k+1} \rangle$ $(0 \leq i \leq \frac{n}{n} - 1)$ はソート済みである．k-merger は部分列 T_{2i} と $T_{2i+1} (0 \leq i \leq \frac{n}{2k} - 1)$ をペアとして，マージを行う．

最初 FIFOA, B は空の状態である．初めの部分列 T_0 が FIFOA に格納され，次の部分列 T_1 が FIFOB に格納される．同様に，部分列 T_{2i} は FIFOA に，T_{2i+1} は FIFOB にそれぞれ入力が行われる．この動作が 1 クロックごとに繰り返される．k-merger は FIFOA に初めの入力データ列 T_0 が全て格納され，FIFOB に次の入力データ列 T_1 の初めのキー t_k が格納されると出力が始まる．従って，2 つの FIFO 内には常に合計で $k+1$ 個のデータが格納されていることになる．また，k-merger のレイテンシは $k+1$ である．片方の FIFO が空である場合では，他方の FIFO のキーを出力する．T_{2i+1} と T_{2i+2} のようなペアの異なる部分列が比較される場合は，T_{2i+1} を出力する．

k-merger のレイテンシは $k+1$ であるため，k-merger を 1-merger, 2-merger, 4-merger, ..., $\frac{d}{2}$-merger という形で順次接続した d-merge-sorter のレイテンシは $(1+1)+(2+1)+(4+1)+\cdots+(\frac{d}{2}+1) = d + \log d - 1$ となる．

次に，マージを行う d-sliding-merger を説明する．d-sliding-merger は k-merger と非常に似た構造をしている．Figure 13 は 8-sliding-merger の構造を示している．8-merger との構造上の違いは両方の FIFOA, B の深さが d である点である．

d-sliding-merger は k-merger に似た動作をする回路である．k-merger と同様に，初めの部分列 T_0 が FIFOA に格納され，次の部分列 T_1 が FIFOB に格納される．同様に，部分列 T_{2i} は FIFOA に，T_{2i+1} は FIFOB にそれぞれ入力される．FIFOA に初めの入力データ列 T_0 が全て格納され，FIFOB に次の入力データ列 T_1 の初めのデータ t_k が格納されると出力が始まる．従って，FIFO 内のデータの総数は $d+1$ である．また，レイテンシは $d+1$ である．k-merger では，部分列 T_{2i} と T_{2i+1} のマージを行い，異なる部分列 T_{2i+1} と T_{2i+2} を比較する場合では，データの大小に関係なく T_{2i+1} が出力される．一方，d-sliding-merger では常に FIFOA, B の先頭のタイムスタンプの大小で比較を行う．

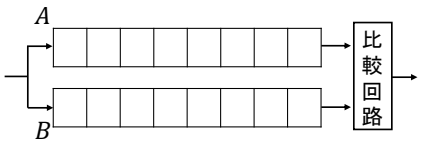

Fig. 13. :8-sliding-merger

d-sorter は d 個のデータ列をソートする d-merge-sorter と d 個のソート済みデータ列をマージする d-sliding-merger で構成されている．従って，d-sorter のレイテンシは $(d + \log d - 1) + (d+1) = 2d + \log d$ となる．

d-sliding-merger では 2 つの FIFO の総容量は $2d$ であったが，実際には FIFO 内に $d+1$ 個のタイムスタンプが常に格納されている．論文 [5] では，より小さな深さの FIFO を複数用いて k-merger を構成することで FIFO の総量の削減を提案している．この手法は d-sliding-merger にも応用が可能である．Figure 14 は 3 つの深さ 4 の FIFO で構成されている 8-sliding-merger を示している．中央の FIFO は FIFOA, B

どちらの役割も果たす可能性がある．Figure 13 が示す通常の 8-sliding-merger は FIFO の総容量が 16 であるのに対して，Fig. 14 は FIFO の総容量を 12 に削減できている．深さ M の FIFO を使用する場合，$\frac{d}{M}+1$ 個の FIFO を用いることで，総容量 $(\frac{d}{M}+1)\cdot M = d+M$ に削減することが可能となる．

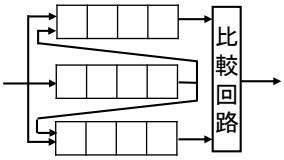

Fig. 14. :3 つの FIFO を用いた 8-sliding-merger

V. 実験結果

本論文では提案実装の比較対象として Intel Core i7-4790 (3.6GHz) を用いてクイックソートの実装を行った．また，FPGA は Virtex Ultra scale+ファミリ XCVU9P-L2FLGA2104E を用いた．この FPGA には 832 個の入出力ポート，2160 個の 36k ビット BlockRAM，960 個の 288k ビット UltraRAM，147780 個の CLB がある．BlockRAM は 1 つの 36k ビット BlockRAM あるいは 2 つの 18k ビット BlockRAM として使用可能である．CLB は 8 つの Look-Up Tables(LUTs) を持っており，分散 RAM を構成可能である．BlockRAM は 36bit のデータ幅を持つため，対象とするデータは 36bit を想定している．従って，36k ビット BlockRAM は 1024 個のデータ，18k ビット BlockRAM は 512 個のデータを格納可能である．また，UltaraRAM は標準ではデータ幅 72bit であるため，アドレス空間を半分に分割することで，8192 個のデータを格納可能である．より大きな FIFO は複数の BlockRAM または UltraRAM を使用することで構成可能である．

本論文では，使用する Virtex Ultra scale+ファミリの入出力ポート数と使用するデータ幅 36bit から，ソーターの並列数を 8 とした．また，使用可能な RAM の総容量とソーターの並列数からソーターの大きさは 1048576-merge-sorter が最大である．column ソートの横の長さは $r \geq 2s^2$ という制約があるため，最大で 512 になる．従って，本論文でソート可能な最大のデータ数は $1048576 \times 512 = 512$M 個である．

また本論文では，表 I と表 II に示すように，k-merger もしくは k-sliding-merger で用いる FIFO に $256 \geq k$ の場合，分散 RAM を使用する．また，$d = 524288$ のときは $8192 \geq k > 256$ の場合で BlockRAM を使用し，$262144 \geq k > 8192$ の場合で UltraRAM を使用する．一方，$d = 1048576$ のときは $65536 \geq k > 256$ の場合で BlockRAM を使用し，$524288 \geq k > 65536$ の場合で UltraRAM を使用する．

k-merger もしくは k-sliding-merger では，2 つの FIFO を用いて構成する手法と複数の FIFO を用いて構成する手法の 2 つがある．複数の FIFO を用いた場合，FIFO の制御に多くの CLB を用いることになるが，使用する RAM の量を減らすことが可能である．例えば，8192-merger を 2 つの FIFO で構成する場合では 16 個の 18k ビット BlockRAM が必要になる．一方，9 つの FIFO で構成した場合，9 個の 18k ビット BlockRAM が必要となる．本論文では，巨大なデータ列を対象としているため，RAM の容量がボトルネックとなることから複数の FIFO を用いて構成する手法を使用する．表 I と表 II は k-merger もしくは k-sliding-merger で用いる FIFO の個数を示している．

次に本論文で提案する 2 つのソート手法について説明する．ここで並列数を M と置き，N 個のデータ列（$r \times s$ の行列）を column ソートでソートするとする．

simple column sort: 単純に column ソートを行う手法では Step1,3,5,7 の計 4 回のソートが必要となる．それぞれの Step でソートされるデータ列の大きさは全て r である．データがパイプライン的に入力されているとすると，必要とするクロックサイクル数は $4(r + logr - 1 + N/M)$ で計算される．

d-sort column sort: d-sorter を用いる column ソート手法では，Step1,3 と Step5' の計 3 回のソートと最後の 1 回のマージが必要となる．データがパイプライン的に入力されたとすると，Step1,3 で必要とするクロックサイクル数は $2(r + logr - 1 + N/M)$ となる．また，Step5' と最後の 1 回のマージで必要とするクロックサイクル数は $2s^2 + logs^2 + N/M + s^2/2 + 1 + 2s^2$ で計算される．ここで，最後のマージは既にソート済みのデータ列をマージするため，d-sorter の最後の部分である d-sliding-merger に直接入力可能である．ただし本論文では，Step1,3 で使用する d-merge-sorter と Step5' と最後の 1 回のマージで必要とする d-sorter を別々に実装するものとする．従って，Fig. 15 のように column ソートの途中で FPGA のコンフィギュレーションを一回行わなければならないことに注意する．

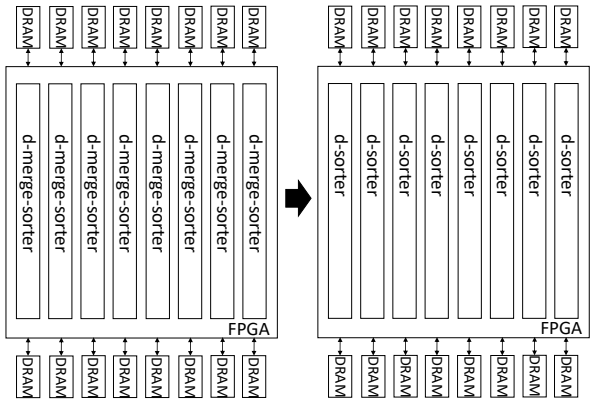

Fig. 15. :8 並列のときの d-sort column sort の全体図

次に，既存実装である FIFO ベースのマージソートとツリーベースのマージソートを組み合わせた手法 [5] との比較を行う．ソーターには 1048576-merge-sorter と 262144-sorter を用いていると仮定する．表 III は提案手法と既存手法にかかる clock cycle 数を示している．ただし，既存手法にかかる clcok cycle 数はレイテンシを含んでいない．ここで，提案手法で 512M 個のソートを行う実装を考えているため，448M 個のソートも可能である．表 III からわかるように，提案実装である simple column sort にかかる clock cycle 数は既存実装の約 $\frac{0.272629\times10^9}{1.342116\times10^9} = 0.203$ 倍である．また，提案実装である d-sort column sort にかかる clock cycle 数は既存実装の約 $\frac{0.204603\times10^9}{1.342116\times10^9} = 0.152$ 倍である．ただし，既存実装におい

TABLE I
$d = 524288$ の場合の FIFO の最適な数

k		1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
k-merger	memory type	distributed RAMs									18kbit block RAMs	36kbit block RAMs			
	FIFOs	2	3	2	2	2	2	2	2	2	2	2	3	5	9
k-sliding -merger	memory type	distributed RAMs									18kbit block RAMs	36kbit block RAMs			
	FIFOs	-	-	-	-	-	-	-	-	-	-	-	-	-	-

k		16384	32768	65536	131072	262144
k-merger	memory type	Ultra RAMs				
	FIFOs	3	5	9	17	33
k-sliding -merger	memory type	Ultra RAMs				
	FIFOs	-	-	-	-	33

TABLE II
$d = 1048576$ の場合の FIFO の最適な数

k		1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536
k-merger	memory type	distributed RAMs									18kbit block RAMs	36kbit block RAMs						
	FIFOs	2	3	2	2	2	2	2	2	2	2	2	3	5	9	17	33	65
k-sliding -merger	memory type	distributed RAMs									18kbit block RAMs	36kbit block RAMs						
	FIFOs	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

k		131072	262144	524288
k-merger	memory type	Ultra RAMs		
	FIFOs	17	33	65
k-sliding -merger	memory type	Ultra RAMs		
	FIFOs	-	33	-

ても提案実装と同じようにコンフィギュレーションが一度行われることに注意する．

TABLE III
提案実装と既存実装にかかるクロックサイクル数

データ数	448M	512M
simple column sort(FPGA)	-	0.272629×10^9
d-sort column sort(FPGA)	-	0.204603×10^9
既存手法 (FPGA)	1.342116×10^9	-

最後に，提案手法とクイックソートの CPU 逐次実装の比較を行う．本論文では，512M 個と 256M 個のデータをソート対象としている．従って，必要な FPGA 実装は 1048576-merge-sorter または 524288-merge-sorter と 262144-sorter である．表 IV は d-merge-sorter の実装結果を示している．今回では，Step2 において 32 連続でデータを書き込むことにしたため $512 \times 4 \times 8 = 16384$ 個の 36bit データを格納できるバッファーを実装している．従って，バッファーに 16 個の 36k ビット RAM を用いている．また，表 V は d-sorter の実装結果を示している．表 I, II, IV より 1048576-merge-sorter では 524288-merge-sorter と比較して，BlockRAM を使用した FIFO の個数が多いため，FIFO の制御がより複雑になることで CLB の個数が約 2 倍とかなり多くなっていることがわかる．

TABLE IV
d-MERGE-SORTER の性能

d		524288	1048576
d-merge-sorter FPGA	CLB (out of 147780)	37909	79562
	Block RAMs (out of 2160)	176	1096
	Ultra RAMs (out of 960)	536	920
	clock (MHz)	250	154

表 VI は提案手法またはクイックソートの CPU 逐次実装を用いた場合における 512M 個（1048575×512）と 256M（524288×512）個のデータのソートにかかる時間も示されている．データ数が 256M 個のとき，simple column sort の手法ではクイックソートの逐次実装と比較して，$\frac{34.449}{0.5452} = 63$ 倍の高速化を達成している．一方，d-sort column sort ではクイックソートの逐次実装と比較して，$\frac{34.449}{0.4099} = 84$ 倍の高速化を達成した．また，データ数が 512M 個のとき，simple column sort の手法では C 実装のクイックソートと比較して，$\frac{71.921}{1.7703} = 40$ 倍の高速化を達成している．一方 d-sort column sort では，$\frac{71.921}{1.1571} = 62$ 倍の高速化を達成している．従って，基本的には d-sort column sort の方が simple column sort より実行時間が短い．ただし d-sort column sort の場合では，d-merge-sorter と d-sorter の FPGA 実装を途中で書き換えなければならない．従って，もしコンフィギュレーションの時間が $1.7703 - 1.1571 = 0.6132$ 秒より大きいならば，512M 個のデータをソートする場合は simple column sort の手法の方がソートにかかる時間が短い．また，256M 個のデータをソートするときも，コンフィギュレーションの時間が $0.5452 - 0.4099 = 0.1353$ 秒より大きいならば，simple column sort の手法の方がソートにかかる時間が短い．

TABLE V
d-SORTER の性能

d		262144
d-sorter FPGA	CLB (out of 147780)	33664
	Block RAMs (out of 2160)	160
	Ultra RAMs (out of 960)	536
	clock (MHz)	253

TABLE VI
COLUMN ソートとクイックソートのソート時間 [S]

データ数		256M	512M
simple column sort(FPGA)	実行時間	0.5452	1.7703
d-sort column sort(FPGA)	実行時間	0.4099	1.1571
quick sort（C 実装）	実行時間	34.449	71.921

VI. 結論

本論文では，巨大なデータ列に対する FPGA を用いた column ソートの手法を提案した．通常の column ソートでは 4 回のソートが必要であるが，d-sorter を用いることで 3 回のソートで完全にソート可能であることを示した．実装結果として，提案実装では，256M 個のデータをソートするとき，クイックソートの CPU 逐次実装と比較して 63-84 倍の高速化を達成した．また，512M 個のデータをソートする場合では，クイックソートの CPU 逐次実装と比較して 40-62 倍の高速化を達成した．また，提案実装にかかる clock cycle 数は既存実装と比較して約 0.152-0.203 倍かかることを示した．

REFERENCES

[1] D. E, Kunth, *The Art of Computer Programming*. *Vol.3: Sorting and Searching. Addison-Wesley*, 1973

[2] S. G, Akl, *Parallel Sorting Algorithms*. Academic Press Inc., 1990.

[3] A. Todd, "Algorithm and hardware for a merge sort using multiple processors, "*IBM Journal of Research and Development*, vol.22, no. 5, pp. 509–517, Sept. 1978.

[4] T. Usui, T. V. Chu, K. Kise, "A Cost-Effective and Scalable Merge Sorter Tree on FPGAs," *Proc. of International Symposium on Computing and Networking (CANDAR)*, November 2016

[5] N. Matsumoto, K. Nakano, and Y. Ito, "Optimal parallel hardware k-sorter and top k-sorter, with FPGA implementations," in *Proc. of International Symposium on Parallel and Distributed Computing*, June 2015, pp. 138–147.

[6] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on FPGAs," *The International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 1–23, Feb. 2012.

[7] D. Koch, Jim Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on FPGAs for Large Problem Sorting," *ACM/SIGDA international symposium*, pp. 45–54, March.2011.

[8] W. M. Zabolotny, "Dual port memory based heapsort implementation for FPGA," in *Proc. SPIE, Photonics Applications, Industry, and High-Energy Physics Experiments*, Oct.2011.

[9] T. Leighton, "Tight bounds on the complexity of parallel sorting,"*IEEE Transactions on Computers*, C-34(4):344–354, April 1985.

[10] N. Harada, K. Nakano and Y. Ito, A hardware sorter for almost sorted sequences, with FPGA implementations, *Proc. of International Symposium on Computing and Networking (CANDAR)*, pp. 565–571, November 2016

[11] S. Todd, "Algorithm and hardware for a merge sort using multiple processors, "*IBM Journal of Research and Development*, vol.22, no. 5, pp. 509–517, Sept. 1978.