

単一ソートセル構造による 省ロジックFPGA マージソート

伊藤 愛^{1,a)} 林崎 弘成^{1,b)} 小原 盛幹^{1,c)}

概要: FPGA で高速なソートを実現するアルゴリズムとして、マージソートがある。このうち、マージソートツリーでは、ソートセルを2分木構造状に接続し、葉にあたる部分からデータを入力し根にあたる部分からデータを出力することで K -way のマージを実現する。 K の値を大きくすることにより、一度により多くのデータをマージすることができるようになるが、その分ソートセルの数が増加するためロジックの消費量が線型に増大してしまう。本発表では、よりロジック効率の良い多-way マージソートを実現するため、ロジック数を削減した単一ソートセル・マージソートツリーを提案する。具体的には、従来は木構造の各段で複数のソートセルが実装されていたものを、平均的にはあるクロックでは高々1つのソートセルしか動作しないことに着目し、各段1つの物理ソートセルのみでまかなうようにする。また、この構造を正しくかつ高効率に動作させるための各物理ソートセルの制御アルゴリズムを示す。これによって、スループットを保ったまま、マージソートに必要なロジック消費量を削減することができた。

1. はじめに

近年、FPGA によってソート処理を高速化する研究が活発に行われている。ソートの性能は多くのアプリケーションにとって重要であり、データベース、画像処理、データ圧縮などで FPGA による高速化手法が研究されてきた [1][2][3]。

FPGA での実装が行われているソートアルゴリズムはさまざまあるが、一番ロジック効率及び実行効率がよいのは、マージソートツリーである [4]。これは、2入力1出力で比較を行うソートセルで構成され、このソートセルを二分木状に接続してマージソートを行う。

マージソートツリーでは、複数のソートセルを接続することによって、キーの比較を複数並列で行う。しかし、あるクロックスライスに着目してみると、二分木の最上位にあるソートセルでは一度に左右どちらかの1つのキーしか消費されないため、その他の段では平均的には1クロックあたり高々1つのソートセルしか実行されておらず、ロジックの無駄が生じている。これは、ソートの way 数を増加させる時に問題となる。

本発表では、ロジック効率を高めるために、物理ソート

セルを各段に1つのみ配置する単一ソートセル・マージソートツリーを提案する。物理ソートセル間に中間状態のキーを保持する RAM ブロックを持つことで、仮想的に二分木状のソートセル接続を実現する。各段の物理ソートセルは、その段の論理ソートセルで共有され、論理ソートセル番号が与えられると、必要なキーを RAM ブロックから読み出して、指定された論理ソートセルに相当する処理を行う。ソートが実行された場合、消費されたキーに対応する論理ソートセル番号を下位段の物理ソートセルに送る。

この手法では、より大きいマージソートツリーで、物理ソートセルの数をより多く減らすことができる。物理ソートセルはコンパレータを含み、マージソートツリーの中でも特にロジック消費量が多いため、マージソートツリー全体のロジック量を大きく削減することができる。論理ソートセル番号の伝達のために追加のロジックを実装する必要があるが、これは少ないロジック量で十分実装可能である。なお、1クロックスライスに各段が1つのキーを出力するという状態には変わらないため、スループット性能への影響はない。

提案手法の単一ソートセル・マージソートツリーを実装し、Altera Stratix V 上で実験を行った。その結果、従来のマージソート実装に比べてロジック消費量を削減することができた。また、データ列を再帰的にソートする実験を行い、スループットおよび実行時間のペナルティがないことを確認した。

¹ 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research - Tokyo

a) megumii@jp.ibm.com

b) hayashiz@jp.ibm.com

c) ohara@jp.ibm.com

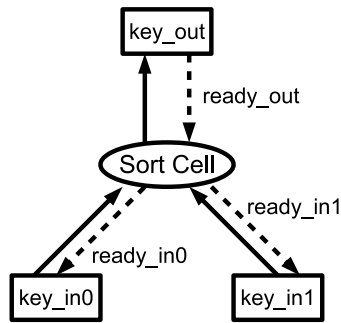


図1 ソートセルの構造

以下、第2章で関連研究について紹介する。第3章で既存のFPGAによるマージソート実装について述べた上で、第4で我々の省ロジック実装を提案する。提案手法について、ロジック消費量と実行性能を評価した結果を第5章で示し、最後に第6章でまとめる。

2. 関連研究

KochらはFIFOベースのマージソートとマージソートツリーを組み合わせたFPGAによるソート手法について提案している[4]。まず最初にFIFOベースのマージソートを用い、その後マージソートツリーを複数回使用してソートを行う。我々の手法をこのマージソートツリーに適用することによって、同じロジック面積により大きなソーターを実装でき、より大きなデータセットのソートを行うことができる。

FPGA上のソーティングネットワークについては、[1]で詳しく議論されている。ソーティングネットワークでは、コンパレーターの接続方法をさまざまに工夫することで、Even-Odd Merge, Bitonic Sort, Bubble Sort, Insertion Sortなどを実現することができる。しかし、これらのいずれにおいても、実装されている入力幅分のデータセットしかソートできず、より大きなデータセットをソートしたい場合には更にCPUなどでマージする必要がある、FPGAのみでソートを完結できない。

LeeらはOdd-Even Mergeソーティングネットワークを拡張し、ソートされたキー配列複数のマージを可能にした[5]。K-way MergerとCombining Networkを再帰的に構築することによって、どのような配列数にも対応することができる。各配列の長さは実装によって固定である。我々の用いる木構造ベースのマージソートでは、各配列の長さに制限はないため、非常に大きなデータセットもソートすることが可能である。

3. 従来のマージソートツリー

マージソートツリーは、2つのキーの比較と選択を行うソートセルによって構成される。ソートセルの構造は、図1のようになっている。ソートセルは2つの入力(key_in0, key_in1)を入力から受け取り、それらのキーの比較結果に

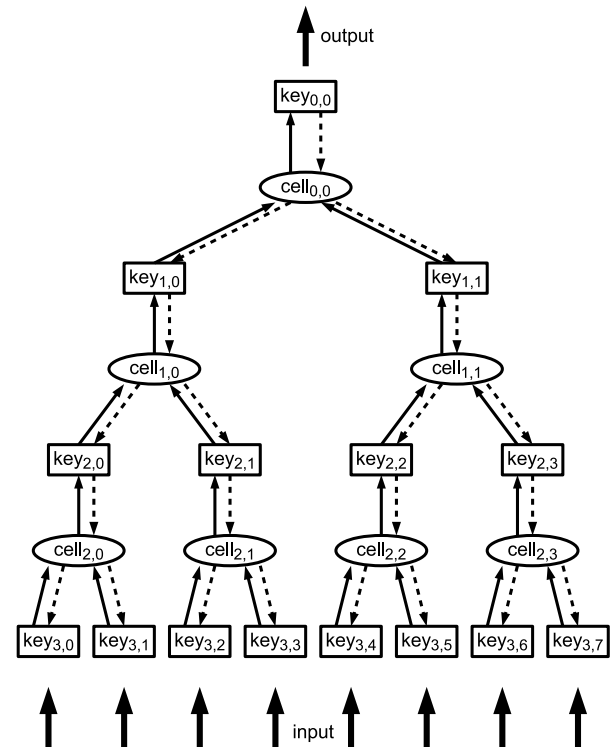


図2 複数ソートセル・マージソートツリー

したがって、どちらかのデータを選択し key_out に出力する。これらのデータの参照・移動を実線矢印で表す。これらの入力元・出力先（四角で表されている key_in0, key_in1, key_out）は、FIFOでデータの入出力を行う。これをバッファキューと称する。

ソートセルの実行は、出力先のコンポーネントから提供される出力 Ready シグナル (ready_out) によって制御される。出力 Ready シグナルが1である場合にのみ、ソートセルはキーの比較と出力を行う。ソートセル自身も、次のデータを要求するため、出力として選択した方の入力に入力 Ready シグナル (ready_in0 または ready_in1) を送る。これらの Ready 信号を図1では点線矢印で表す。

ソートセルを2分木状に接続することで、K-way マージソートツリーが実現できる。図2に $K=8$ の場合を示す。ここで、各ソートセル（縦方向の位置 s と横方向の位置 i とで $cell_{s,i}$ とラベル付ける）は、キーを格納するバッファキュー（同様に $key_{s,i}$ とラベル付ける）を介して前後のソートセルとつながっている。横方向に複数のソートセルがあるため、本発表では、図2のような構造のマージソートツリーを複数ソートセル・マージソートツリーとよぶ。

このマージソートツリーでは、あらかじめソートされている複数のデータ列（図2では $key_{3,0} \sim key_{3,7}$ に入力される）を1つのデータ列 ($key_{0,0}$ から出力される) にマージすることができる。ソートされていないデータ列は、ソートされた長さが1と考えることで対応できる。マージソートの1段目の入力が K 個あるとすると、一度に K 個のデータ列をマージすることができる。データ列の区切りを適切

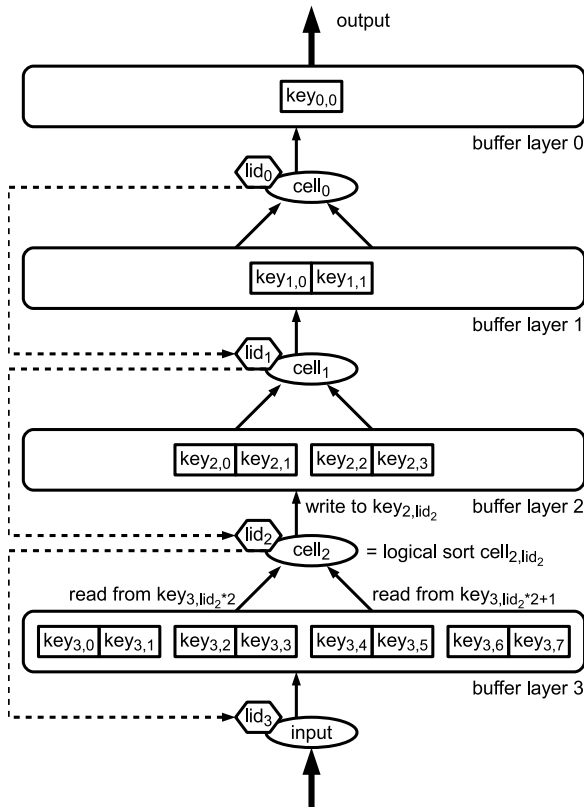


図 3 単一ソートセル・マージソートツリー

に管理することで、複数のデータ列を連続して入力に読み込むことができる。

ソートしたいデータセットのキー数が K より大きい場合は、再帰的にマージソートを実行することでデータセット全体のソートを行うことができる。必要な実行回数は、データが N 個存在する場合、 $\log_K(N)$ で求められる。まず最初に、ソートされた長さが 1 だとしてマージソートを実行し、ソートされた長さが K のデータ列を N/K 個得る。このデータ列に対して再度マージソートを実行し、ソートされた長さが K^2 のデータ列が N/K^2 個得られる。これを $\log_K(N)$ 回実行すると、 N 個のデータをソートすることができる。

4. 提案手法: 単一ソートセル・マージソートツリー

本章で、我々は単一ソートセル・マージソートツリーというマージソート手法を提案する。

図 3 に単一ソートセル・マージソートツリーの構造を示す。物理ソートセル ($cell_0, cell_1, cell_2$) を各段に 1 つのみ配置し (4.1 節)、ソートセル間のバッファキューをバッファレイヤーとして 1 つにまとめ、FPGA 内部の RAM ブロックを用いて実装する (4.2 節) ことで、ロジックの消費量を削減する。これにより、ロジック消費量を従来の $O(K)$ から $O(\log K)$ に削減する。バッファキュー実装のための RAM ブロックの消費量は $O(K)$ であるが、従来のラッチによる

に比べて効率的である。

各物理ソートセルは、各クロックにおいて実行する論理ソートセル番号 (lid_0, lid_1, lid_2) が与えられることで仮想的に図 2 のような論理ソートセル構造を実行するが、この論理ソートセル番号を決定するアルゴリズムを 4.3 節で述べる。このアルゴリズムは、本マージソートツリーの正しくかつ高スループットな実行を実現する。

4.1 単一物理ソートセルによるマージソートツリーの実行

単一ソートセル・マージソートツリーでは、論理的には図 2 に示されるソートセルと同様の処理を実行する。以降、図 2 におけるソートセルを論理ソートセルと称する。あるクロックスライスに着目してみると、3 章で述べたマージソートでは、二分木の最上段にある論理ソートセル ($cell_{0,0}$) では一度に左右どちらかの 1 つのキーしか消費されないため、その他の段 $s = 1, 2$ でも平均的には各段高々 1 つの論理ソートセルだけが実行される。

これを利用して、単一ソートセル・マージソートツリーでは、各クロックにおいて各段 $s = 0, 1, 2$ で高々 1 つの論理ソートセル $cell_{s,lid_s}$ のみ実行することとし、この 1 つの論理ソートセル $cell_{s,lid_s}$ を物理ソートセル $cell_s$ で実行する。変数 lid_s は、段 s でそのクロックにおいて実行すべき論理ソートセル番号 (Logical sort cell ID, lid) を示す。この lid_s を各クロックごとに切り替えながら実行することで、少ない物理ソートセル数で多数の論理ソートセルと同等の処理を実行する。

これにより、ソートセルの数 (従ってロジック使用量) を K -way マージソートツリーに対して $O(K)$ から $O(\log K)$ に削減できる。

4.2 RAM によるバッファレイヤーの実装

複数ソートセルの場合 (図 2)、各論理ソートセル $cell_{s,i}$ はバッファキュー $key_{s+1,i*2}$ 及び $key_{s+1,i*2+1}$ を入力とし、 $key_{s,i}$ を出力としていた。これらのバッファキュー $key_{s,i}$ は各ソートセルにより独立にアクセスされるため、それぞれをロジックで実装する必要がある。

しかし、単一ソートセルの場合 (図 3) は、段 s における物理ソートセル $cell_s$ が論理ソートセル $cell_{s,lid_s}$ を実行する時、 $key_{s+1,0}$ から $key_{s+1,2^{s+1}-1}$ の 2^{s+1} 個のバッファキューのうちで key_{s+1,lid_s*2} と key_{s+1,lid_s*2+1} の 2 つのバッファキューからのみ値を読み、 key_{s,lid_s} のみに出力すればよい。

この特性を利用して、提案手法では、これらのバッファキューを段ごとにまとめ、各段 1 つずつのバッファレイヤーとして FPGA 内部の RAM ブロックによる実装を行う。バッファレイヤー s は、論理バッファキュー $key_{s,0} \sim key_{s,2^s-1}$ に対応し、各バッファキュー $key_{s,i}$ はそのアドレス i によりアクセスできる。バッファレイヤーは全体で

高々 2 読み込みポート, 1 書き込みポートを実装すればいいため, このバッファキューのレイは RAM ブロックとして, アドレス i は RAM ブロックのアドレス線として実装できる.

複数ソートセルではロジックを用いて実装されていたバッファキューを RAM ブロックで実装することにより, ロジックの消費量を削減できる. また, 読み書きするバッファキューの選択を RAM ブロックのアドレス線を用いて実装するため, これをセレクタとして実装するよりも効率的である.

4.3 論理ソートセル番号の制御

単一ソートセル・マージソートツリーを正しく動作させるためには, 各段の論理ソートセル番号 lid_s を適切に制御する必要がある. 本手法では以下の三つの観点から設計を行った.

- (1) 正しく動作すること, 特にデッドロックやバッファキューあふれを起こさないこと.
- (2) (steady state において) 最大のスループットを達成できること.
- (3) 制御にかかるハードウェア量が小さいこと.

単純に「入力側バッファキューに有効なデータがあり, 出力側バッファキューに空きがある論理ソートセルを実行する」という方式では, デッドロックの危険がある. また, どの論理ソートセルがこの条件を満たすかの判定を個別に行うためには, 論理ソートセル数 (従って K) に比例するハードウェアが必要となる.

本提案手法では, 各物理ソートセル $cell_s$ がその下段のソートセル $cell_{s+1}$ に対して, 「論理ソートセル i を実行し, $key_{s+1,i}$ に出力を行え」という論理ソートセル番号リクエスト i を送信する (図 3 の点線). これは, 図 1 のソートセルにおいて, 上段からの Ready 信号を受けて処理を行い, その結果によって下段に Ready 信号を送信するのと同様である. 最上段の物理ソートセル $cell_0$ に対しては, 常に「論理ソートセル 0 を実行せよ」というリクエストが発行される. また, 最下段の物理ソートセル $cell_2$ からのリクエスト i は, ソートの入力を最下段のバッファレイヤーに供給するソートツリー外部の回路 (図 3 の “input”) に対して, i 番目に対応するデータ列供給の要求に使われる.

論理ソートセル番号リクエストが物理ソートセル $cell_s$ に到達すると, リクエストキュー (最大要素数 R の固定長キュー) に追加され, やがてその段における論理ソートセル番号 lid_s となってリクエストが実行される. その際, 以下のようなアルゴリズムで更に下段に対するリクエストを生成する.

Case 1 下段のリクエストキューが満杯であれば, ソートセルは動作せず次のクロックを待つ.

Case 2 下段のリクエストキューに空きがあり, 入力バッ

ファキュー, すなわち key_{s+1, lid_s*2} と key_{s+1, lid_s*2+1} のどちらか (あるいは両方) が空であった場合は, 空であった方のインデックス ($i = lid_s*2$ または $i = lid_s*2+1$) を下段への論理ソートセル番号リクエストとして発行する. リクエストは完了せず, リクエストキューの中にそのまま残る. 次クロックでも空であれば, 同様に下段へのリクエストを発行し続ける.

Case 3 下段のリクエストキューに空きがあり, 入力バッファキューがどちらも空でなければ, key_{s+1, lid_s*2} と key_{s+1, lid_s*2+1} からキーを読んで比較し, 小さいかった方^{*1}のインデックスを i とする. $key_{s+1,i}$ から読んだキーを key_{s, lid_s} に出力し, $key_{s+1,i}$ の先頭要素を削除し, i を下段への論理ソートセル番号リクエストとして発行する. リクエストは完了し, リクエストキューから削除される.

以下で, このアルゴリズムが正しく動作することを示す. まず, 各物理ソートセルは自分の入力バッファキュー及び下段のリクエストキューの状態しか見ない (特に出力先のバッファキューの状態を見ない) ため, デッドロックしない. また, 出力先バッファキューの状態を見ずに出力を行うが, バッファキューサイズ B を十分大きく取ることで (具体的な値の決定については後述する), リクエスト実行時に常に出力先バッファキューに空きがあることを保証できる.

これを, 物理ソートセル $cell_{s+1}$ に対するリクエスト i が, 物理ソートセル $cell_s$ において上記アルゴリズムのどのケースで発行されるかの場合分けで示す.

Case 1 リクエストは発行されない.

Case 2 リクエスト i は, $key_{s+1,i}$ が空の状態から始めて, リクエスト先の下段の物理ソートセル $cell_{s+1}$ により $key_{s+1,i}$ にキーが出力されて, 空でない状態が物理ソートセル $cell_s$ から見えるようになるまで発行され続ける. この間に実行されるリクエスト数の最大数を $M(R)$ とすると, バッファキューに最大 $M(R)$ 要素が入ることになるので,

$$M(R) \leq B \quad (1)$$

が成り立てばよい. リクエストキューが満杯になると Case 1 に相当し, リクエストの発行がされないので, リクエストキューの長さ R の調整により $M(R)$ を調整できる.

Case 3 この場合, 出力バッファキュー $key_{s+1,i}$ の先頭要素を削除してからリクエスト i を発行するため, 常に出力バッファキューに空きがある.

また, 同様に R, B の調整により, このアルゴリズムは, steady state において毎クロック 1 個のデータを出力する

^{*1} ここでは説明のために昇順ソートを仮定するが, 降順ソートでも同様に適用可能である

ことができる (最大スループットの達成):

- (1) start up 時には, Case 2 により, $M(R)$ 個のキーが各バッファキューに格納される.
- (2) その後は steady state となり, Case 3 によって削除された分だけキーが下段から供給される. リクエストの発行から完了までの最低レイテンシを L クロックとすると, $L+1$ 個以上のキーがバッファキューに入っていれば, あるバッファキューからの削除が連続しても空にはならない. よって,

$$L+1 \leq M(R) \quad (2)$$

であればよい.

以上から,

$$L+1 \leq M(R) \leq B \quad (3)$$

と設定すればよいことが分かる^{*2}.

また, このアルゴリズムはリクエストにより指定された論理ソートセルのみに関する処理を行い, 全論理ソートセルに関する条件判定を行わないため, 少ないハードウェア量で実装できる.

5. 評価

本章では, 単一ソートセル・マージソートツリーが複数ソートセル・マージソートツリーに比べて, 性能劣化なくロジック消費量の削減ができていることを評価する.

5.1 評価環境

評価には, PCI-Express Gen2 を通して IBM®PowerLinux 7R2^{*3} (3.55 GHz POWER7 プロセッサ) に接続された, Altera Stratix V を搭載した FPGA ボードを用いた. 本ボード上では, FPGA から PCI-Express 越しにメモリアクセスを行うことができる. この FPGA の実行環境, 及び複数ソートセル・マージソートツリーの基本となる実装は IBM Research - Austin で開発されたものを用いた.

K -way の単一ソートセル・マージソートツリーを本 FPGA 上に実装し, $K=8, 16$ とパラメータを変化させた時のロジック消費量及び実行時間を測定し, 複数ソートセル・マージソートツリーを用いた場合と比較した. マージソートツリーの入出力はメモリ上のバッファに対して行い, 8 バイトのキーと 8 バイトのペイロードで構成される 16 バイトを 1 単位とするデータ列を扱う.

^{*2} これらの式は, 実装やレイテンシの数え方などで定数分ずれることがある

^{*3} IBM, IBM ロゴおよび ibm.com は, 世界の多くの国で登録された International Business Machines Corporation の商標です. 他の製品名およびサービス名等は, それぞれ IBM または各社の商標である場合があります. 現時点での IBM の商標リストについては, www.ibm.com/legal/copytrade.shtml をご覧ください.

以下の三つについて比較を行った.

- A. 既存手法: 複数ソートセル・マージソートツリー. 図 2 に相当する部分.
- B. 提案手法 (コア): 単一ソートセル・マージソートツリー. 図 3 に相当する部分.
- C. 提案手法 (全体): 単一ソートセル・マージソートツリーの図 3 に相当する部分に, A. と同じインターフェースになるように接続回路を加えたもの. A. (図 2) では, 最下段のバッファキューそれぞれに入力信号線 (太矢印) が張られているが, B. (図 3) では, 最下段の入力信号線 (太矢印) は一本のみである. これらの入力信号線をどのように入出力インターフェースと接続するか, という接続回路部分は入出力インターフェース依存の問題であり本論文の比較上では本質的でないと考えられるが, A. と同一インターフェースの置換可能なもの同士を比較するため, C. を実装して評価した.

FPGA の回路全体としては, これらの部分の他に外部入出力 (PCI-Express) 用のインターフェースが含まれるが, 今回の評価では本質的でないので除外した.

5.2 ロジック消費量

ロジック消費量は, Altera 社の Quartus II 11.1 を用いて回路合成を行った際の Fitter Report の “Fitter Resource Utilization by Entity” を元に, マージソートツリー部分に相当する部分のハードウェア消費量を算出した. ロジック使用量の指標としては ALM (アダプティブ・ロジック・モジュール; 使用した FPGA における基本的なロジック・ユニット) の数を用いた. RAM 使用量としては, Memory Bits (RAM の使用ビット数) を使用した.

ソートを行うコア部分のみの比較として, A. と B. 列を比較すると, コア部分の比較では ALM 数が大きく削減できている. また, A. 列の ALM 数の増加は K に対してほぼ線型であるが, B. 列の ALM 数の増加は K に対して線型よりも緩やかである. K がより大きいほど, 単一ソートセル・マージソートツリーによるロジック消費量の削減効果が大きいことが分かる.

外部接続インターフェースをそろえた A. と C. の比較では, インターフェースとマージソートツリーの間の接続回路のために, $K=8$ では C. の方が ALM 数が多く, $K=16$ では A. と C. の ALM 数がほぼ同じになっている. 今回は複数ソートセル・マージソートツリーと単一ソートセル・マージソートツリーで同じ外部接続インターフェースを採用したためこのような結果になったが, 単一ソートセル・マージソートツリーのための外部接続インターフェースを採用することでこの部分のロジック消費量は削減することができる.

表 1 ロジック消費量

Way 数 K	A. 既存手法		B. 提案手法 (コア)		C. 提案手法 (全体)	
	ALM 数	Memory Bits	ALM 数	Memory Bits	ALM 数	Memory Bits [KB]
8	12939	0	9195	48208	15526	49488
16	27049	0	14268	99056	27746	101616

表 2 相対実行時間 (既存手法 = 1.00)

Way 数 K	既存手法		提案手法	
	sort 済	ランダム	sort 済	ランダム
8	1.00	1.00	1.01	0.99
16	1.00	1.00	0.96	0.97

5.3 実行性能評価

表 2 に, $K = 8, 16$ に対して 2^{24} 要素 (1 要素 16 バイト) をソートさせた時の相対実行時間 (複数ソートセル・マージソートツリー = 1.0) を示す. データサイズは, 各 K に対して無駄がないように K の累乗になるように選んだ. この結果から, 単一ソートセル・マージソーターが複数ソートセル・マージソートツリーに対して, 実行性能上の劣化がないことが示された.

6. おわりに

本発表では, マージソートを高スループットかつ少ないロジック消費で行うことができる単一ソートセル・マージソートツリーを提案した. 実機 FPGA 上での実装と評価を行い, 従来の複数マージセル・マージソートツリーによる実装に比べ, ロジック消費量を削減でき, かつ, 実行性能の劣化はないことを示した. また, way 数の増加によるロジック消費量の増加が複数ソートセル・マージソートツリーに比べて緩やかであることを示し, 本手法が特に多 way のマージソートツリーを実装するのに適した手法であることを示した.

- [6] *tributed Systems*, 6(2): 211-215, 1995.
S. Wong, S. Vassiliadis, and J. Hur. Parallel Merge Sort on a Binary Tree On-Chip Network. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, pp. 365-368, November 2005.

参考文献

- [1] R. Mueller, T. Jens, and A. Gustavo. Sorting Networks on FPGAs. In *The VLDB Journal—The International Journal on Very Large Data Bases*, 21(1): 1-23, 2012.
- [2] K. Ratnayake, and A. Aishy. An FPGA Architecture of Stable-Sorting on a Large Data Volume: Application to Video Signals. In *Proceedings of the 41st Annual Conference on Information Sciences and Systems (CISS)*, 2007.
- [3] J. Martinez, R. Cumplido, and C. Feregrino. An FPGA-based Parallel Sorting Architecture for the Burrows Wheeler Transform. In *Proceedings of International Conference on Reconfigurable Computing and FPGAs, (ReConFig)*, 2005.
- [4] D. Koch, and J. Torresen. FPGASort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on FPGAs for Large Problem Sorting. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 45-54, February 2011.
- [5] D. L. Lee, and E. B. Kenneth. A Multiway Merge Sorting Network. In *IEEE Transactions on Parallel and Dis-*