Modular Design of High-Throughput, Low-Latency Sorting Units

Amin Farmahini-Farahani, Henry J. Duwe III,
Michael J. Schulte, *Senior Member*, *IEEE*, and Katherine Compton, *Member*, *IEEE*

Abstract—High-throughput and low-latency sorting is a key requirement in many applications that deal with large amounts of data. This paper presents efficient techniques for designing high-throughput, low-latency sorting units. Our sorting architectures utilize modular design techniques that hierarchically construct large sorting units from smaller building blocks. The sorting units are optimized for situations in which only the M largest numbers from N inputs are needed, because this situation commonly occurs in many applications for scientific computing, data mining, network processing, digital signal processing, and high-energy physics. We utilize our proposed techniques to design parameterized, pipelined, and modular sorting units. A detailed analysis of these sorting units indicates that as the number of inputs increases their resource requirements scale linearly, their latencies scale logarithmically, and their frequencies remain almost constant. When synthesized to a 65-nm TSMC technology, a pipelined 256-to-4 sorting unit with 19 stages can perform more than 2.7 billion sorts per second with a latency of about 7 ns per sort. We also propose iterative sorting techniques, in which a small sorting unit is used several times to find the largest values.

Index Terms—VLSI designs, design optimization, parallel sorting algorithms, partial sorting, iterative sorting

1 Introduction

Sorting is an important operation in a wide range of applications including data mining, databases [1], [2], [3], digital signal processing [4], [5], network processing, communication switching systems [6], [7], scientific computing [8], searching, scheduling [9], pattern recognition, robotics [10], image and video processing [11], [12], [13], and high-energy physics (HEP) [14]. For applications that require very high-speed sorting, hardware sorting units are often implemented using either ASICs or FPGAs to meet performance requirements [3], [13], [15], [16], [17]. Based on target applications, hardware sorting units vary greatly not only in architecture but also in the number of inputs and the width of inputs that they can process. For instance, only 9 to 25 inputs need to be processed in certain filters [11], while the number of inputs can vary from 25 to 81 (or even higher) in certain image processing applications [18]. High-speed sorters on FPGAs in HEP applications deal with 128 to 256 data samples in 100-ns processing cycles [14], [19]. Thousands of inputs are sorted in video [13] and database applications [3], [20]. In general, inputs can be b-bit integers ($8 \leq b \leq 64$), floating-point numbers, or even compressed data values.

Most previous research on sorting units has focused on the situation in which the sorting unit must produce all of its inputs in sorted (increasing or decreasing) order. In many applications, however, only the M largest (or smallest) output values need to be selected from a total of N input values, where $M < N$. For example, in many HEP applications, only the M most energetic particles may be of interest. Similarly, in signal processing applications, only the M strongest signals or M closest points in space may need to be analyzed. In data mining, searching, and database systems, only top query outputs that score the most with respect to a given search key may need further processing. Furthermore, depending on the application, the M largest (smallest) outputs may not need to be in order. We refer to units that only return the M largest (smallest) outputs, but do not guarantee that these M outputs are sorted, as max(min)-set-selection units. We refer to units that only return the M largest (smallest) outputs in sorted order as partial sorting units.

This paper focuses on the design of partial sorting and max-set-selection units that return the $M = 2^m$ largest values from $N = 2^n$ inputs, where m and n are each whole numbers and $1 \leq M < N$.[1] We refer to these units as N-to-M partial sorting and max-set-selection units. Our units discard small inputs as early as possible to reduce the sorting units' latency and hardware complexity. We investigate the design and VLSI implementations of partial sorting and max-set-selection units with low latency, high throughput, and modest resource requirements. Our designs are based on Batcher's [25] bitonic and odd-even merge sorting networks, which are widely used in VLSI and FPGA implementations due to their simplicity, regularity, and parallelism. The proposed units are scalable in terms of both the number of inputs and the number of outputs. We also present a generalized

- *A. Farmahini-Farahani, H.J. Duwe III, and K. Compton are with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, 1415 Engineering Drive, Madison, WI 53706. E-mail: {farmahinifar, duweiii}@wisc.edu, kati@engr.wisc.edu.*
- *M.J. Schulte is with Advanced Micro Devices, Inc., Austin, TX 78741. E-mail: michael.schulte@amd.com.*

Manuscript received 8 Aug. 2011; revised 2 Apr. 2012; accepted 12 Apr. 2012; published online 23 May 2012.
Recommended for acceptance by P. Montuschi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-08-0531.
Digital Object Identifier no. 10.1109/TC.2012.108.

1. Straightforward modifications to our designs allow the M smallest values, rather than the M largest values, to be output. It is also feasible to remove the current restriction that M and N are integer powers of two using techniques similar to those presented in [6], [21], [22], [23], [24].

platform-independent methodology for designing high-performance pipelined partial sorting and max-set-selection units for which the width of the data to be sorted and the pipeline depth can easily be varied.

This research is an extension of our previous work on FPGA-based sorting units in the Large Hadron Collider (LHC) [19]. The main contributions of this paper and [19] are:

- Modular techniques for designing N-to-M partial sorting and max-set-selection units. The units are composed of small and regular building blocks connected in a modular fashion, thereby reducing verification time and simplifying the design process. Our designs have low latency, high throughput, and modest resource requirements, can be pipelined easily, have parameterized pipeline depth and data widths, and scale well to large values of N and M. Moreover, our techniques are independent of the bit-width and type of input values.
- A detailed analysis of our proposed partial sorting and max-set-selection units that includes both theoretical and synthesis estimates of our units' latency, throughput, and resource requirements. This analysis indicates that for a given number of outputs, resource requirements scale linearly with the number of inputs, latency scales logarithmically with the number of inputs, and the frequency remains nearly constant. Compared to conventional sorting units, which return all of their inputs in sorted order, our N-to-M partial sorting and max-set-selection units have much lower latency and area.
- A discussion of how the proposed max-set-selection units may be utilized iteratively to find the largest values from a set of data. This approach may lower resource requirements, storage cost, and I/O requirements at the cost of increased latency and decreased throughput.

To the best of our knowledge, this is the first time that N-to-M partial sorting networks have been presented and analyzed. In this work, we propose parallel sorting algorithms for finding/sorting the M largest values from N inputs and then design scalable architectures based on the proposed algorithms. Our N-to-M partial sorting networks have lower latency than any previous sorting designs when producing only the M largest values. Furthermore, our N-to-M max-set-selection units further decrease the latency and resource requirements by not producing their outputs in sorted order. Our parallel units target applications that require very low-latency sorting. Our iterative units target applications that require moderate-latency sorting by trading increased latency for reduced area and I/O bandwidth. Although our sorting units were originally designed for HEP experiments in the LHC, our methodology can be utilized to design high-speed sorting and max-set-selection units for a wide range of applications.

The remainder of this paper is organized as follows: Section 2 describes previous sorting networks and provides background information. Section 3 presents our partial sorting and max-set-selection units. Section 4 gives synthesis results for our units. Section 5 shows how these units are used iteratively to sort data. Section 6 discusses related research on sorting algorithms and architectures. Section 7 concludes the paper.

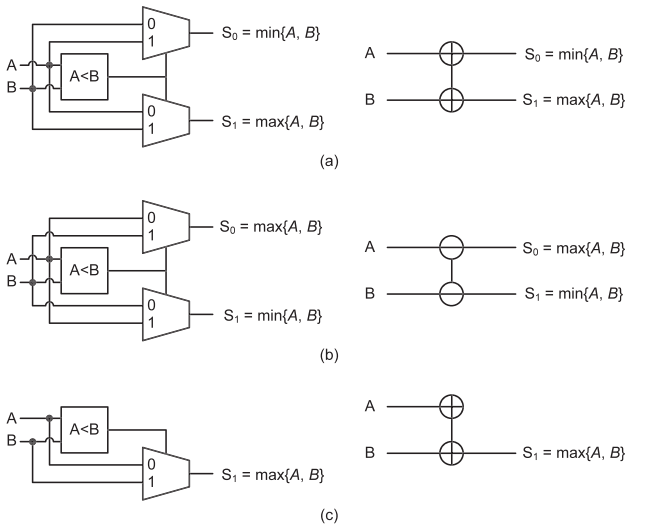

Fig. 1. The high-level implementation (left) and schematic symbol (right) of building blocks for sorting networks.

2 Parallel Sorting Networks

A sorting network is a collection of interconnected compare-and-exchange (CAE) blocks that guides a parallel set of inputs to a parallel set of outputs in sorted order. Each CAE block has two inputs and two outputs. If the input values are already in order, they are directed to the corresponding outputs; otherwise, they are swapped.

There are two types of CAE blocks, called increasing and decreasing CAE blocks, used in hardware-based sorting units. Fig. 1 shows the high-level implementations (left) and schematic symbols (right) for three building blocks used in previous sorting units and in our designs. Fig. 1a shows an increasing CAE block, which outputs its two inputs in ascending order. A decreasing CAE block, shown in Fig. 1b, outputs its inputs in descending order. Decreasing and increasing CAE blocks are identical, except for their wiring. Each CAE block contains a comparator and two multiplexers. We also define Max units which are used in our designs. A Max unit, shown in Fig. 1c, takes two inputs and returns the larger input. Note that the $\oplus$ and $\ominus$ symbols determine the type of the block in Fig. 1.

A sorting network usually consists of a series of stages in which each stage contains a number of CAE blocks that operate in parallel. The latency of a sorting network is proportional to its depth (the number of consecutive CAE blocks). Two popular parallel sorting networks that currently have the lowest known latency for hardware implementation are the bitonic and odd-even merge sorting networks proposed by Batcher [25]. The structure of a sorting network is fixed, regardless of the value of the numbers being sorted and the results of the comparisons. Sorting networks are a common solution for hardware-based sorting. Their parallel nature allows them to perform sorting much faster than the $O(N \times \log(N))$ time achievable by the fastest sequential software-based sorting algorithms. A sorting network may also be pipelined to further increase throughput.

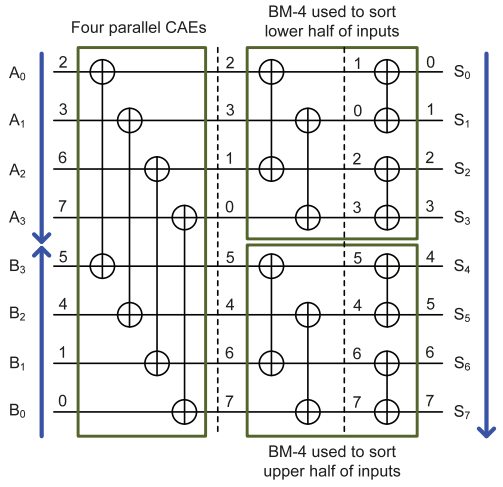

Fig. 2. An increasing 8-input bitonic merging unit ($\oplus \mathrm{BM}-8$) that is composed of four parallel CAE blocks followed by two parallel BM-4 units. The bitonic input sequence {2, 3, 6, 7, 5, 4, 1, 0} is the concatenation of the increasing sequence {2, 3, 6, 7} and the decreasing sequence {5, 4, 1, 0}.

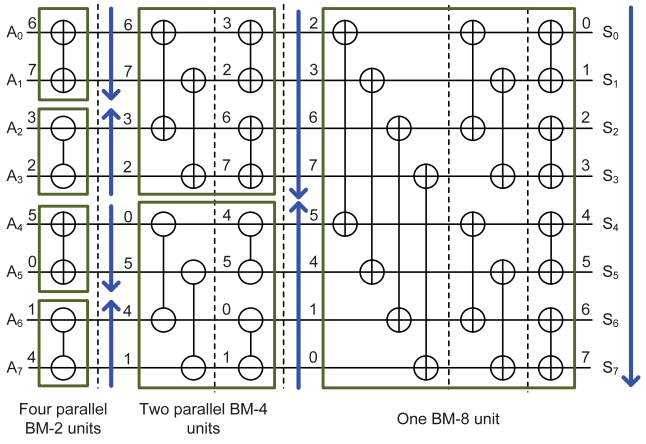

Fig. 3. The CAE network for an 8-input bitonic sorting unit with six CAE stages and 24 CAE blocks, made up of increasing ($\oplus$) and decreasing ($\ominus$) bitonic merging units. Arrows point in the direction of increasing values.

2.1 Bitonic Sorting Networks

Concatenating an ascending and a descending sequence forms a single bitonic sequence. A bitonic sorting network recursively merges an ascending and a descending sequences each of length $N/2$ to make a sorted sequence of length N [25]. Each bitonic sorting network is composed of a number of bitonic merging units to merge bitonic sequences.

A K-input bitonic merging unit (denoted as BM-K) contains $\log_2(K)$ stages of parallel CAE blocks, where each stage corresponds to a CAE stage with $K/2$ CAE blocks. Therefore, a BM-K requires $\log_2(K) \times K/2$ CAE blocks. For instance, an increasing BM-8 (denoted as $\oplus \mathrm{BM}-8$) unit has three CAE stages and requires $3 \times (8/2) = 12$ CAE blocks, as shown in Fig. 2. In this figure, the arrows point in the direction of increasing numbers and the dashed lines separate CAE stages. A BM-K unit is itself composed of a level of $K/2$ parallel CAE blocks followed by two parallel BM-($K/2$) units. The $\oplus \mathrm{BM}-8$ is constructed from a level of four parallel CAE blocks followed by two parallel BM-4 units that have four CAE blocks each. A decreasing BM-8 (denoted as $\ominus \mathrm{BM}-8$) has a similar structure to an increasing BM-8, but it is only constructed from decreasing CAE blocks and the sorted outputs are in descending order.

An 8-input bitonic sorting unit has four parallel BM-2 units, two parallel BM-4 units, and one BM-8 unit, as shown in Fig. 3. Thus, this sorting unit has $1+2+3=6$ CAE stages. Assuming the unit is pipelined so each stage takes one clock cycle, it can generate the sorted outputs in six cycles and can begin a new sort each cycle. In an N-input bitonic sorting unit, there are equal numbers of increasing and decreasing BM units in each level, excluding the last level, which has only either an increasing BM-N unit or a decreasing BM-N unit. In general, an N-input bitonic sorting unit is composed of $N/2$ BM-2 units, $N/4$ BM-4 units, $N/8$ BM-8 units, ..., two BM-$N/2$ units, and one BM-N unit. In this design, BM-K units are followed by BM-($2K$) unit(s), where $K = 2, 4, 8, \ldots, N/2, N$, $2 \le K \le N$ and K and N are integer powers of 2. Therefore, because an N-input bitonic sorting unit has $\log_2(N)$ consecutive BM-K units, where each BM-K unit has $\log_2(K)$ CAE stages, an N-input bitonic sorting unit has $\log_2(N) \times (\log_2(N)+1)/2$ CAE stages. Since each stage has $N/2$ CAE blocks, the total number of CAE blocks in an N-input bitonic sorting unit is $N \times \log_2(N) \times (\log_2(N)+1)/4$. For example, 16-input, 32-input, and 256-input bitonic sorting units require 10, 15, and 36 CAE stages and have 80, 240, and 4,608 CAE blocks, respectively.

2.2 Odd-Even Merge Sorting Networks

An odd-even merge sorting network recursively merges two ascending sequences of length $K/2$ to make a sorted sequence of length K. Each odd-even merge sorting network is composed of a number of odd-even merging units. A K-input odd-even merging unit (OEM-K) merges two ascending input sequences into a single ascending output sequence. It contains $\log_2(K)$ CAE stages, where each stage has between $K/4$ and $K/2$ CAE blocks. An OEM-K takes two length $K/2$ ascending sequences, A and B. The OEM-K merges the input values having odd indices in A with the input values having odd indices in B, and also merges input values in A and B having even indices. The result is a sorted sequence of values with odd indices (S_O) and a sorted sequence of values with even indices (S_E). S_O and S_E are generated recursively, separately, and in parallel. In the final stage, the S_O and S_E sequences are merged to generate a sorted sequence with K values, S_0 to S_{K-1}. The merging process is simply a CAE of values in the S_O and S_E sequences.

An 8-input odd-even merging unit is shown in Fig. 4. An OEM-K unit is composed of two parallel OEM-$K/2$ units, followed by a level of $(K/2-1)$ parallel CAE blocks. Unlike BM units, OEM units are only built from increasing CAE blocks. The total number of CAE blocks for an OEM-K unit is $(K/2) \times (\log_2(K)-1)+1 = \log_2(K/2) \times (K/2)+1$. Thus, an OEM-8 unit has three CAE stages and $2 \times 4 + 1 = 9$ CAE blocks. It is constructed from two parallel OEM-4 units that each have three parallel CAE blocks followed by a level of three parallel CAE blocks.

An 8-input odd-even merge sorting unit has four OEM-2 units, two OEM-4 units, and one OEM-8 unit, as shown in

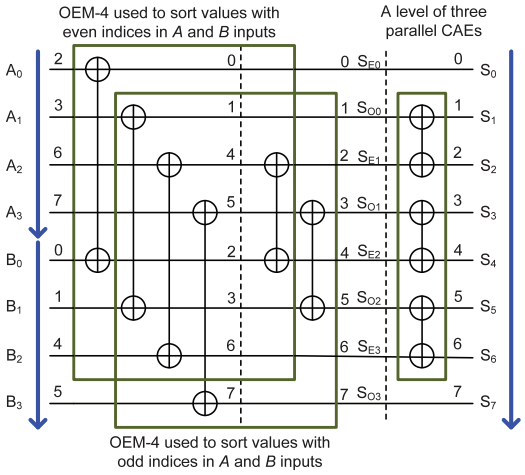

Fig. 4. An 8-input odd-even merge unit (OEM-8) that is composed of two OEM-4 units and a level of three parallel CAE blocks.

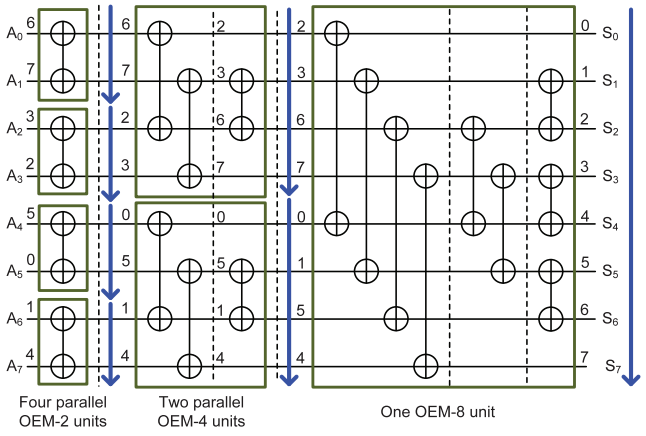

Fig. 5. The CAE network for an 8-input odd-even merge sorting unit with six CAE stages and 19 CAE blocks.

Fig. 5. It requires four parallel OEM-2 units, two parallel OEM-4 units, and a single OEM-8 unit, leading to a sorting unit with $1+2+3=6$ CAE stages. Assuming the unit is pipelined with one stage per clock cycle, it can generate the sorted outputs in six cycles. In general, an N-input odd-even merge sorting unit is composed of $N/2$ OEM-2 units,[2] $N/4$ OEM-4 units, $N/8$ OEM-8 units, ..., two OEM-$N/2$ units, and one OEM-N unit. In this design, OEM-K units are followed by OEM-$2K$ unit(s), where $K = 2, 4, 8, \ldots, N/2, N$, and K and N are integers power of 2. Therefore, because an N-input odd-even merge sorting unit is composed of $\log_2(N)$ consecutive OEM-K units, where each OEM-K unit has $\log_2(K)$ CAE stages, an N-input odd-even merge sorting unit has $(\log_2(N)) \times (\log_2(N)+1)$ stages. This is equivalent to the number of CAE stages in an N-input bitonic sorting unit. In addition, because an OEM-K has $\log_2(K/2) \times (K/2) + 1$ CAE blocks, an N-input odd-even merge sorting unit has $N/4 \times (\log_2(N)) \times (\log_2(N)-1) + N - 1$ CAE blocks.

2.3 Designing Large Sorting Networks

Based on the idea behind the bitonic and odd-even merge algorithms, large sorting units can be built using large merging units [25] that consist of multiple CAE stages of increasingly larger size. Table 1 summarizes the required number of CAE blocks and stages for bitonic and odd-even merge sorting units. Both N-input bitonic and odd-even merge sorting units have a time complexity (depth) of $O(\log_2^2(N))$ CAE stages and have an area complexity of $O(N \times \log_2^2(N))$ CAE blocks. An N-input, N-output bitonic or odd-even merge complete sorting unit is composed of $(\log_2 N) \times (\log_2 N + 1)/2$ CAE stages, where $N = 2^n$. However, the required number of CAE blocks differs for each type of sorting unit. Bitonic and odd-even merge sorting unit with 2^n inputs and 2^n outputs have $2^{n-2} \times n \times (n+1)$ and $2^{n-2} \times n \times (n-1) + 2^n - 1$ CAE blocks, respectively. Thus, odd-even merge sorting units have lower resource requirements than bitonic sorting units, but may have more complex wiring. The difference in the number of CAE blocks between bitonic and odd-even merge sorting units is $2^{n-1} \times (n-2) + 1$, showing that the difference in the number of CAE blocks increases linearly with the number of inputs.

2. BM-2 and OEM-2 units have the same structure.

3 Proposed Partial Sorting and Max-Set-Selection Units

In many applications, it is not necessary to return all of the sorted inputs. Applications often only need to determine the M largest or M smallest numbers from N inputs, where $M < N$ and M and N are both integer powers of two ($M = 2^m$, $N = 2^n$). Partial sorters provide the 2^m largest values in sorted order, and max-set-selection units provide the 2^m largest values in arbitrary order. Partial sorters and max-set-selection units are key components in many applications. For example, in the LHC [26] low-latency max-set-selection units identify important particle interactions that correspond to high-energy collisions [19]. In multimedia applications, partial sorters speed up data sorting algorithms [12]. Moreover, auxiliary max-set-selection units can cooperate with general-purpose processing units in embedded and database management systems to accelerate data search and sort algorithms. In cases such as this, Batcher's algorithms can be optimized to generate only the 2^m largest numbers from 2^n inputs with less latency and fewer CAE blocks than a complete sorting network.

3.1 4-Output Max-Set-Selection and Partial Sorting Units

We first discuss 8-to-4 max-set-selection units and then extend our technique to larger 2^n-to-4 max-set-selection and

TABLE 1
The Required Number of CAE Blocks and CAE Stages for 2^n-Input Bitonic and Odd-Even Merge Sorting Units

# of inputs and outputs (N=M=2^n)	# of CAE Stages	# of CAE blocks		
		Bitonic	Odd-even	Difference
8	6	24	19	5
16	10	80	63	17
32	15	240	191	49
64	21	672	543	129
128	28	1,792	1,471	321
256	36	4,608	3,839	769

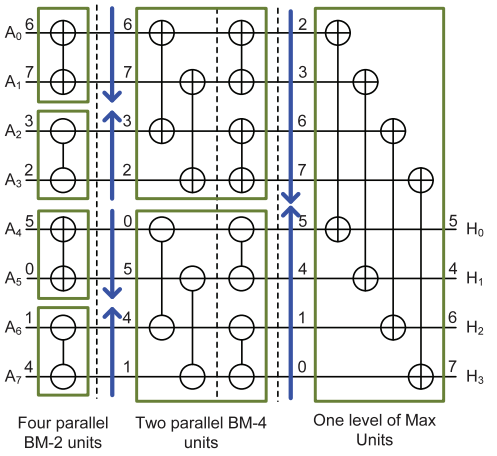

Fig. 6. The CAE network for an 8-to-4 bitonic max-set-selection unit with four CAE stages and 16 CAE blocks.

partial sorting units. To design 2^n-to-4 max-set-selection units, we take advantage of the fact that only the four largest inputs are needed, in no particular order, to decrease the resource requirements and the number of CAE stages.

3.1.1 8-to-4 Max-Set-Selection Units

To illustrate our technique, we first present the design of a refined 8-input bitonic sorting unit, called an 8-to-4 bitonic max-set-selection unit, that only returns the four largest numbers. A special feature of bitonic sequences is that performing Max operations on two sorted sequences (one increasing and the other one decreasing) each of K numbers, generates two new sequences of K numbers in which numbers in one sequence are all less than numbers in the other sequence. In BM-K units, the first level of parallel CAE blocks partitions the input numbers into two $(K/2)$-number subsets: The smaller numbers and the larger numbers. However, the first level of parallel CAE blocks in OEM units must be rewired to correctly generate the smaller and larger subsets of numbers.

Figs. 6 and 7 show 8-to-4 max-set-selection units that use bitonic and odd-even merge algorithms, respectively. As shown in Fig. 6, the BM-8 unit in Fig. 3 is replaced with a level of Max units that has the same wiring as the first level of parallel CAE blocks in the BM-8 unit. Fig. 7

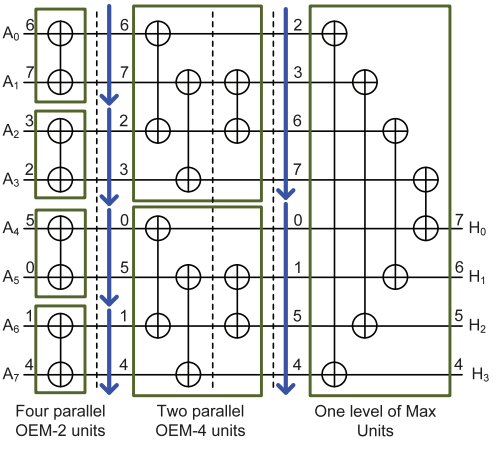

Fig. 7. The CAE network for an 8-to-4 odd-even merge max-set-selection unit with four CAE stages and 14 CAE blocks.

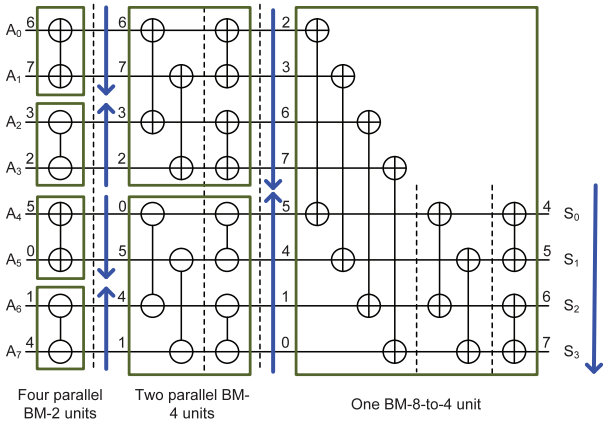

Fig. 8. The CAE network for an 8-to-4 bitonic partial sorting unit with six CAE stages and 20 CAE blocks.

illustrates that the OEM-8 unit in Fig. 5 is replaced with a level of Max units with wirings that differ from the first level of parallel CAE blocks in the OEM-8 unit. These modifications decrease the required number of CAE stages from six in 8-input sorting units to four in 8-to-4 max-set-selection units.

3.1.2 BM-8-to-4 and 8-to-4 Partial Sorting Units

The 8-to-4 max-set-selection units, however, cannot be used directly to form larger sorting or max-set-selection units because the outputs of the 8-to-4 max-set-selection units in Figs. 6 and 7 are not in a specific order. Since inputs to BM units should be a bitonic sequence and inputs to OEM units should be two ascending sequences, these designs cannot be connected directly to other BM or OEM units.

To solve this problem, we have designed a new merging unit called a BM-8-to-4 unit, shown as the right-most block in Figs. 8 and 9. A BM-8-to-4 unit is an 8-input bitonic merging unit that outputs only the four largest values in either ascending $\oplus$ or descending $\ominus$ order. Fig. 8 depicts our proposed 8-to-4 bitonic partial sorting unit that returns the four largest values from its eight inputs in ascending order. Compared to the 8-input bitonic sorting unit shown in Fig. 3, it needs fewer CAE blocks and the BM-8 unit is replaced with a BM-8-to-4 unit. A descending 8-to-4 partial sorting unit has a similar structure. A BM-8-to-4 unit has a

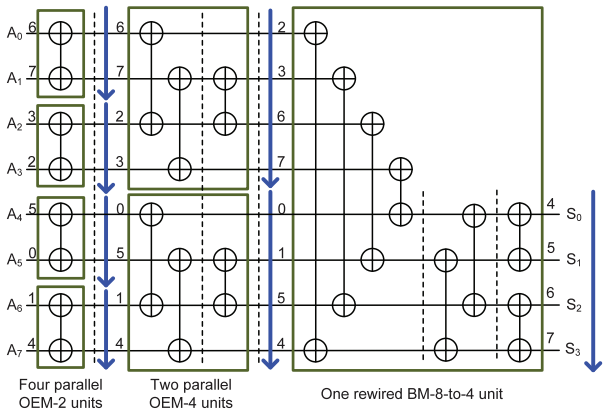

Fig. 9. The CAE network for an 8-to-4 odd-even merge partial sorting unit with six CAE stages and 18 CAE blocks.

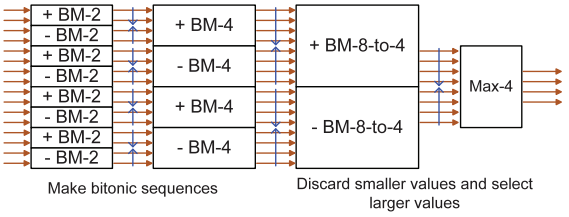

Fig. 10. A 16-to-4 bitonic max-set-selection unit.

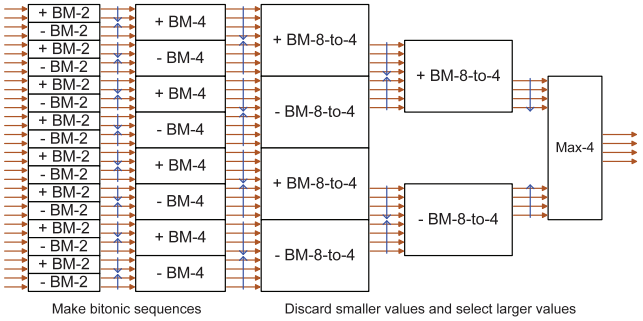

Fig. 11. A 32-to-4 bitonic max-set-selection unit.

level of four parallel CAE blocks followed by a BM-4 unit. With this approach, the output of an 8-to-4 bitonic partial sorting unit can be fed to other bitonic merging units.

Fig. 9 depicts our proposed 8-to-4 odd-even merge partial sorting unit that returns the four largest values from its eight inputs in ascending order. Compared to the 8-input odd-even merge sorting unit in Fig. 5, it needs fewer CAE blocks and the OEM-8 unit is replaced with a BM-8-to-4 unit. In fact, the partial sorting unit shown in Fig. 9 is a hybrid unit composed of bitonic and odd-even merging units. Since a bitonic merging unit takes only a bitonic sequence, the inputs to the BM-8-to-4 unit in Fig. 9 are rewired to convert the two sorted sequences to a bitonic sequence. This way, the output of odd-even merging units can be fed to bitonic merging units. We could also propose an OEM-8-to-4 unit to avoid the rewiring technique. However, since an OEM-8-to-4 unit requires more registers than a BM-8-to-4, only BM-2^{k+1}-to-2^k units are used throughout the paper. Note that only increasing BM-8-to-4 units are used in our proposed odd-even merge sorting unit, while both increasing and decreasing BM-8-to-4 units are used in the bitonic sorting unit.

3.1.3 2^n-to-4 Max-Set-Selection and Partial Sorting Units

To design larger 2^n-to-4 max-set-selection units, we can take advantage of the fact that BM-8-to-4 units can be combined to make larger units. Since BM-8-to-4 units generate sorted outputs, these outputs can feed other BM-8-to-4 units. We can build larger 2^n-to-4 bitonic max-set-selection units using BM-2, BM-4, BM-8-to-4, and Max-4 units. We can also build larger 2^n-to-4 odd-even merge max-set-selection units using OEM-2, OEM-4, BM-8-to-4, and Max-4 units. BM-2, BM-4, BM-8, BM-8-to-4, and Max-4 units require 1, 4, 12, 8, and 4 CAE blocks, respectively. OEM-4 and OEM-8 units require 3 and 9 CAE blocks, respectively.

Figs. 10 and 11 show the structures of 16-to-4 and 32-to-4 bitonic max-set-selection units that utilize multiple BM-8-to-4 units. In these designs, smaller numbers are discarded in

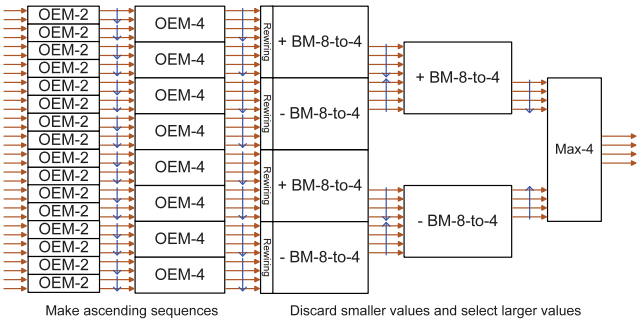

Fig. 12. A 32-to-4 odd-even merge max-set-selection unit.

stages as early as possible to reduce the total number of CAE stages and CAE blocks. A 16-to-4 bitonic max-set-selection unit has one level of parallel BM-8-to-4 units, while a 32-to-4 bitonic max-set-selection unit employs two levels of parallel BM-8-to-4 units.

Fig. 12 depicts the high-level structure of a 32-to-4 odd-even merge max-set-selection unit that utilizes several OEM and BM-8-to-4 units. Unlike bitonic max-set-selection and partial sorting units, odd-even merge units only use increasing merging units, which is an advantage in terms of simplicity of design. On the other hand, as shown in Fig. 12, the outputs of OEM-4 units are rewired to feed BM-8-to-4 units, which is a disadvantage. In general, 2^n-to-4 max-set-selection units have $n-3$ levels of parallel BM-8-to-4 units for $n > 3$.

To make a 2^n-to-4 partial sorting unit, the last level of a 2^n-to-4 max-set-selection unit, which is a Max-4 unit, is replaced by a BM-8-to-4 unit to generate outputs in sorted order. This increases the number of CAE stages and the number of CAE blocks by 2 and 4, respectively.

Table 2 shows the resource requirements and the number of CAE stages for our proposed 2^n-to-4 and 2^n-to-8 max-set-selection units. The required number of CAE stages for a 2^n-to-4 bitonic max-set-selection unit can be calculated based on the fact that it is composed of a level of parallel BM-2 units, a level of parallel BM-4 units, $n-3$ levels of parallel BM-8-to-4 units, and a Max-4 unit. Similarly, the required number of CAE blocks can be calculated based on the fact that there are 2^{n-1} BM-2 units, 2^{n-2} BM-4 units, $2^{n-2}-2$ BM-8-to-4 units, and one Max-4 unit in a 2^n-to-4 bitonic max-set-selection unit. Both 2^n-to-4 bitonic and odd-even merge max-set-selection units have $3 \times n - 5$ CAE stages for $n \geq 3$. 2^n-to-4 bitonic and odd-even merge max-set-selection units require $7 \times 2^{n-1} - 12$ and $13 \times 2^{n-2} - 12$ CAE blocks, respectively.

Tables 1 and 2 indicate significant improvements of max-set-selection units compared to the conventional complete sorting units in terms of both the number of CAE stages and the required number of CAE blocks. For example, while a 256-input complete sorting unit has 36 CAE stages, a 256-to-4 max-set-selection unit has 19 CAE stages, and a 256-to-4 partial sorting unit has 21 CAE stages. As shown in Table 1, a 256-input bitonic sorting units requires 4,608 CAE blocks. However, a 256-to-4 bitonic max-set-selection unit and a 256-to-4 bitonic partial sorting unit require 884 and 888 CAE blocks, respectively. Similarly, as shown in Table 1, a 256-input odd-even merge

TABLE 2
Structure and number of CAE stages and CAE blocks for 2^n-to-4 and 2^n-to-8 bitonic and odd-even merge max-set-selection units made up of smaller merging units

Max-set-selection Unit	Structure		# of CAE Stages	# of CAE Blocks	
	Bitonic	Odd-even		Bitonic	Odd-even
8-to-4	BM-2(4)→ BM-4(2)→ Max-4(1)	OEM-2(4)→ OEM-4(2)→ Max-4(1)	4	16	14
16-to-4	BM-2(8)→ BM-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(8)→ OEM-4(4)→ BM-8-to-4(2)→ Max-4(1)	7	44	40
16-to-8	BM-2(8)→ BM-4(4)→ BM-8(2)→ Max-8(1)	OEM-2(8)→ OEM-4(4)→ OEM-8(2)→ Max-8(1)	7	56	46
32-to-4	BM-2(16)→ BM-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(16)→ OEM-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	10	100	92
32-to-8	BM-2(16)→ BM-4(8)→ BM-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(16)→ OEM-4(8)→ OEM-8(4)→ BM-16-to-8(2)→ Max-8(1)	11	144	124
64-to-4	BM-2(32)→ BM-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(32)→ OEM-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	13	212	196
64-to-8	BM-2(32)→ BM-4(16)→ BM-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(32)→ OEM-4(16)→ OEM-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	15	320	280
128-to-4	BM-2(64)→ BM-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(64)→ OEM-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	16	436	404
128-to-8	BM-2(64)→ BM-4(32)→ BM-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(64)→ OEM-4(32)→ OEM-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	19	672	592
256-to-4	BM-2(128)→ BM-4(64)→ BM-8-to-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(128)→ OEM-4(64)→ BM-8-to-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	19	884	820
256-to-8	BM-2(128)→ BM-4(64)→ BM-8(32)→ BM-16-to-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(128)→ OEM-4(64)→ OEM-8(32)→ OEM-16-to-8(16)→ OEM-16-to-8(8)→ OEM-16-to-8(4)→ OEM-16-to-8(2)→ Max-8(1)	23	1372	1216

The numbers in parentheses under "Structure" show the required number of each unit.

sorting unit requires 3,839 CAE blocks. However, a 256-to-4 odd-even merge max-set-selection unit and a 256-to-4 odd-even merge partial sorting unit require 820 and 824 CAE blocks, respectively.

Table 2 shows how max-set-selection units can be built from BM/OEM and Max units. For instance, to implement a 256-to-4 max-set-selection unit, 128 BM-2/OEM-2 units, 64 BM-4/OEM-4 units, 62 BM-8-to-4 units, and a Max-4 unit are needed. In general, odd-even merge max-set-selection units need fewer CAE blocks than the corresponding bitonic max-set-selection units, but bitonic max-set-selection units have more regular structures and are easier to design. Table 2 indicates that the fastest 256-to-4 max-set-selection units have a latency of 19 clock cycles (assuming each CAE stage takes one clock cycle). In comparison, conventional 256-input sorting units require 36 clock cycles to sort all 256 data values.

3.2 2^n-to-2^m Max-Set-Selection and Partial Sorting Units

Our proposed techniques can be extended to cover a wide range of sorting and max-set-selection units. In the previous section, 2^n-to-4 max-set-selection units (with $n > 3$) utilize BM-8-to-4 units to select the four largest values out of a bitonic sequence with eight values. The sorted outputs of BM-8-to-4 units are then combined to form bitonic sequences that are fed to the next stage of BM-8-to-4 units or a Max-4 unit. The smallest values are discarded in the earlier stages to reduce resources and latency.

3.2.1 Modular Max-Set-Seletion Units

A similar approach can be used to design 2^n-to-2^m max-set-selection units with BM-2^{m+1}-to-2^m merging units. Starting from unsorted inputs and by following Batcher's algorithms, small sorted sequences are constructed using either BM or OEM units. When 2^{n-m} sorted sequences of length 2^m are generated, each pair of sorted sequences of length 2^m is fed to a BM-2^{m+1}-to-2^m unit to produce the 2^m largest values in sorted order and discard the 2^m smallest values. Using BM-2^{m+1}-to-2^m units, this process continues until only two sorted sequences of length 2^m are left. These two sorted sequences contain the 2^{m+1} largest values from the 2^n input values. In the final stage, a Max-2^m unit returns the 2^m largest values in arbitrary order.

With this approach, a 2^n-to-2^m bitonic or odd-even merge max-set-selection unit with $n > m$ has a total of $(2^{n-m} - 2)$ BM-2^{m+1}-to-2^m merging units in $n - m - 1$ levels, where each BM-2^{m+1}-to-2^m unit is composed of a level of 2^m parallel CAE blocks and a BM-2^m unit.[3] For example, a bitonic 128-to-16 max-set-selection unit has 64 BM-2, 32 BM-4, 16 BM-8, eight BM-16, six BM-32-to-16, and one Max-16 units, where each BM-32-to-16 unit has a level of 16 parallel CAE blocks and a BM-16 unit. Similarly, a 128-to-16 odd-even merge max-set-selection unit, shown in Fig. 13, has 64 OEM-2, 32 OEM-4, 16 OEM-8, eight OEM-16, six BM-32-to-16, and one Max-16 unit. Table 2 shows the structure of 2^n-to-4 and 2^n-to-8 max-set-selection units.

3.2.2 Modular Partial Sorting Units

We can easily modify max-set-selection units to produce their outputs in ascending order rather than arbitrary order. To design a 2^n-to-2^m partial sorting unit that takes 2^n inputs and returns 2^m sorted outputs, the Max-2^m unit in a 2^n-to-2^m max-set-selection unit is replaced with a BM-2^{m+1}-to-2^m unit, producing outputs in ascending order. Thus, a 2^n-to-2^m partial sorting unit has a total of $(2^{n-m} - 1)$ BM-2^{m+1}-to-2^m merging units. This increases the number of CAE stages and the number of CAE blocks for a 2^n-to-2^m sorting unit over its corresponding max-set-selection unit by m and $2^{m-1} \times m$, respectively.

3. A BM-2-to-1 is basically a Max-1 unit.

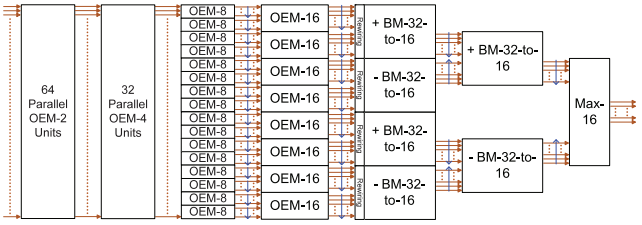

Fig. 13. A 128-to-16 odd-even merge max-set-selection unit.

3.3 Analysis

We can analyze the number of CAE stages and CAE blocks for 2^n-to-2^m max-set-selection and sorting units using our proposed approach. Fig. 14 shows the number of CAE stages for bitonic and odd-even merge max-set-selection and partial sorting units. In this figure, * indicates the required number of CAE stages when outputs have arbitrary order (max-set-selection units) and \$ indicates the number of stages when outputs are sorted (sorting units). In all cases, bitonic and odd-even merge units have the same number of CAE stages. The number of CAE stages in max-set-selection units and partial sorting units is $n(m+1)-m(m+3)/2$ and $(2n-m)(m+1)/2$, respectively. Thus, N-to-M max-set-selection and partial sorting units have a time complexity of $O(\log_2 N \times \log_2 M)$, where $N=2^n$ and $M=2^m$. For a 2^m-output max-set-selection or partial sorting unit, doubling the number of inputs increases the number of CAE stages by $m+1$. For a 2^n-input max-set-selection unit and a 2^n-input partial sorting unit, doubling the number of outputs from 2^m to 2^{m+1} increases the number of CAE stages by $n-m-2$ and $n-m-1$, respectively. The difference in the number of CAE stages between sorting units and max-set-selection units increases logarithmically with the number of outputs.

Fig. 15 shows the required number of CAE blocks for 2^n-to-2^m bitonic partial sorting and max-set-selection units as n and m are varied. The number of CAE blocks in bitonic max-set-selection and partial sorting units is $(m(m+3)+4)\times 2^{n-2}-2^m\times(m+1)$ and $(m(m+3)+4)\times 2^{n-2}-2^{m-1}\times(m+2)$, respectively. Keeping the number of outputs fixed, the number of CAE blocks increases linearly with the number of inputs. Keeping the number of inputs fixed, the number of CAE blocks increases sublinearly with the number of outputs. The difference in

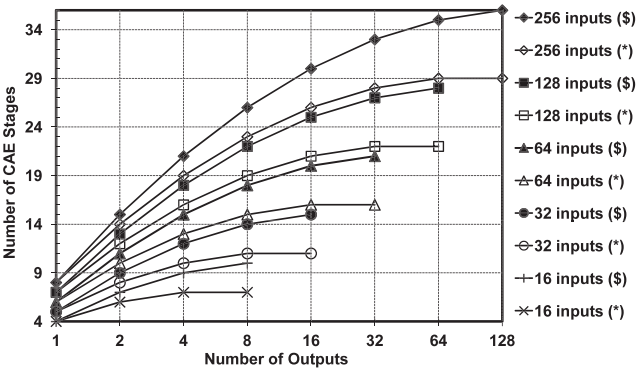

Fig. 14. The number of CAE stages for 2^n-to-2^m partial sorting (\$) and max-set-selection (*) units.

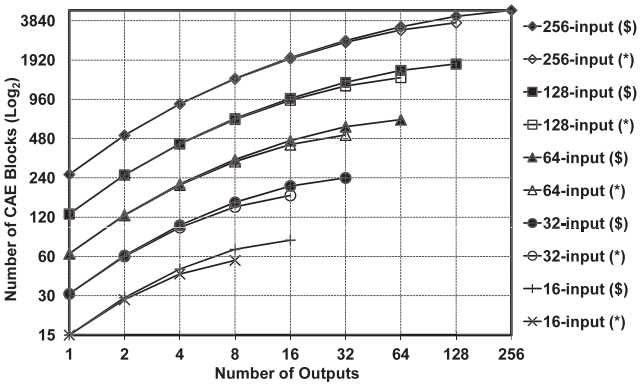

Fig. 15. The number of CAE blocks for 2^n-to-2^m bitonic partial sorting (\$) and max-set-selection (*) units.

the number of CAE blocks between partial sorting units and max-set-selection units increases logarithmically with the number of outputs.

The number of CAE blocks in odd-even merge max-set-selection and partial sorting units is $(m(m+3)+4)\times 2^{n-2}-m\times 2^{n-1}+(1-2^{-m})\times 2^n-2^m\times(m+1)$ and

$$(m(m+3)+4)\times 2^{n-2}-m\times 2^{n-1}+(1-2^{-m})\times 2^n-2^{m-1}\times(m+2),$$

respectively. Thus, N-to-M bitonic and odd-even merge max-set-selection and partial sorting units have an area complexity of $O(N\times\log_2^2(M))$. For all the units, the number of CAE blocks increases linearly with the number of inputs. When the number of outputs is one or two, bitonic and odd-even merge max-set-selection and partial sorting units require the same number of CAE blocks. However, as the number of inputs increases, the benefit of using the odd-even merge algorithm over the bitonic algorithm increases in terms of the required number of CAE blocks for both partial sorting and max-set-selection units.

4 Implementation and Results

To assist with analyzing implementations of our proposed techniques, we developed Verilog register transfer level (RTL) models for 2^n-to-2^m partial sorting and max-set-selection units. The Verilog models are fully parameterized to provide the flexibility needed to design and analyze a wide range of sorting and max-set-selection units. The designer can change (add or remove) each level of pipeline registers to get a design with a different latency, resource requirements, and frequency. This feature helps achieve the desired throughout and latency. The models are composed of small, verified building blocks to simplify the design process and facilitate testing.

The proposed designs are synthesized using the Synopsys design compiler vB 2008.09 SP3 and a TSMC 65-nm standard-cell library. The designs are pipelined and all outputs are registered. For all the synthesis results, the parameterizable data width (i.e, the CAE width), which can be easily changed, is set to 10 unsigned bits, which is commonly used in HEP applications. Tables 3 and 4 show post-place-and-route results for 2^n-to-4 and 2^n-to-8 max-set-selection units, respectively. We report implementation

TABLE 3
Implementation Results of 2^n-to-4 Max-Set-Selection Units

Max-set-selection Unit	Pipeline Depth	Frequency (MHz)		Combinational Area (μm^2)		Non-combinational Area (μm^2)		End-to-end Latency (ns)	
		Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
16-to-4	7	2,941	2,941	5,400	4,862	8,781	8,317	2.38	2.38
16-to-4	4	1,851	2,000	7,881	7,171	4,282	4,671	2.16	2.00
16-to-4	3	1,449	1,449	8,240	6,852	3,514	3,540	2.07	2.07
32-to-4	10	2,857	2,857	13,696	11,229	20,008	17,962	3.50	3.50
32-to-4	5	1,886	1,886	16,681	16,174	11,503	10,648	2.65	2.65
32-to-4	4	1,428	1,449	18,960	18,695	7,764	8,241	2.80	2.76
64-to-4	13	2,777	2,857	24,888	20,140	39,633	37,324	4.68	4.55
64-to-4	7	1,785	1,851	40,880	30,914	22,922	22,204	3.92	3.78
64-to-4	5	1,315	1,449	42,507	35,227	17,143	18,172	3.80	3.45
128-to-4	16	2,702	2,702	46,029	48,178	70,695	68,465	5.92	5.92
128-to-4	8	1,785	1,754	83,975	70,339	37,773	45,253	4.48	4.56
128-to-4	6	1,369	1,315	89,493	69,588	37,164	35,575	4.38	4.56
256-to-4	19	2,702	2,702	94,606	85,178	142,308	137,457	7.03	7.03
256-to-4	10	1,754	1,754	166,578	144,062	90,221	87,715	5.70	5.70
256-to-4	7	1,298	1,315	150,596	139,322	64,517	53,936	5.39	5.32

results for three different pipeline structures (depths) for each bitonic and odd-even max-set-selection unit: One CAE stage between pipeline registers, two CAE stages between pipeline registers, and three CAE stages between pipeline registers. The last column of Tables 3 and 4 lists the end-to-end latency for each design. All max-set-selection units can achieve high frequencies due to the regular pipelined structure of the designs.

As shown in Tables 3 and 4, the clock frequency scales fairly well for larger max-set-selection units. The reason is that the frequency depends on the CAE width and the number of CAE blocks between pipeline registers. Increasing the number of inputs for a max-set-selection unit does not directly affect the frequency, although it might complicate the wire routing. By decreasing the pipeline depth for a given max-set-selection unit, combinational area increases due to trading increased area from buffers and larger/faster gates for the higher clock frequency. As expected, noncombinational area from registers decreases for a given max-set-selection unit by decreasing the pipeline depth. Comparing the two types of max-set-selection units with each other, odd-even merge max-set-selection units have higher clock frequencies and lower areas than bitonic units for most designs and configurations.

Designs with one CAE stage between pipeline registers have higher sorting throughput than similar designs with two or three CAE stages between pipeline registers, although they usually have higher latency and total area. Designs with three CAE stages between pipeline registers have the lowest latency and total area in most cases, while they provide the lowest sorting throughput. Designs with two CAE stages between pipeline registers attain a tradeoff between latency and throughput. For instance, a pipeline depth of seven at about 1.3 GHz provides the lowest latency and lowest resource usage for the 256-to-4 max-set-selection unit, while a pipeline depth of 19 at 2.7 GHz gives the highest throughput.

Tables 3 and 4 show that, for a given number of outputs and pipeline stages, resource requirements scale linearly, latency scales logarithmically, and the frequency scales fairly well with the number of inputs. These results conform to the theoretical analysis of the proposed max-set-selection units in Section 3.3. For a given number of outputs, as the number of inputs increases, units with fewer pipeline stages provide better end-to-end latency. Modular design and intelligent pipelining enable efficient frequency/latency tradeoffs even for large sorting units.

TABLE 4
Implementation Results of 2^n-to-8 Max-Set-Selection Units

Max-set-selection Unit	Pipeline Depth	Frequency (MHz)		Combinational Area (μm^2)		Non-combinational Area (μm^2)		End-to-end Latency (ns)	
		Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
16-to-8	7	1,449	3,030	7,788	5,816	10,953	9,860	2.38	2.31
16-to-8	4	2,000	1,923	11,171	7,916	6,693	6,179	2.00	2.10
16-to-8	3	1,515	1,562	9,252	8,482	4,488	4,328	1.98	1.92
32-to-8	11	2,777	2,702	17,477	14,996	25,697	24,268	3.96	4.07
32-to-8	6	1,818	1,851	30,026	24,047	16,502	15,684	3.30	3.24
32-to-8	4	1,470	1,492	27,086	23,175	12,125	11,373	2.72	2.68
64-to-8	15	2,777	2,702	38,249	31,557	57,829	54,986	5.40	5.55
64-to-8	8	1,754	1,818	65,972	48,038	32,103	32,930	4.56	4.40
64-to-8	5	1,470	1,428	69,731	57,929	20,923	21,640	3.40	3.50
128-to-8	19	2,702	2,702	70,847	82,888	109,827	107,933	7.03	7.03
128-to-8	10	1,754	1,754	135,024	108,498	73,844	68,305	5.70	5.70
128-to-8	7	1,333	1,333	128,380	111,700	45,972	45,229	5.25	5.25
256-to-8	23	2,631	2,702	137,774	126,672	226,955	216,093	8.74	8.51
256-to-8	12	1,724	1,694	267,060	194,382	94,730	112,020	6.96	7.08
256-to-8	8	1,298	1,351	262,126	222,433	88,069	88,545	6.16	5.92

5 Iterative Max-Set-Selection Units

Parallel sorting and max-set-selection units that operate on large blocks of data may receive considerable amounts of input data. Implementing large max-set-selection and partial sorting units in a fully parallel manner requires high I/O bandwidth and area. In addition, for a fixed number of outputs, the resource requirements of these units increase linearly with the number of input values. Thus, fully parallel sorting units may not be practical in large data sorting applications.

In cases in which I/O bandwidth or area is limited and latency requirements are not as stringent, a small max-set-selection unit can be employed using an iterative process to obtain the largest values from a given input data set. This iterative approach is particularly well suited to systems in which only a portion of the total data arrives to the max-set-selection unit each cycle and the sorting throughput requirements are not too high. Furthermore, such iterative max-selection units can provide throughput, latency, and resource requirement tradeoffs. Max-set-selection and partial sorting units in applications such as HEP and video processing often need to get data from different sources over multiple cycles. Thus, our proposed iterative max-set-selection units, which take as inputs new input data and the largest results from previous iterations, are area-efficient designs for these types of applications.

There has previously been successful research on iterative sorting. Huang et al. [27] describe an iterative sorting method that assumes all elements to be sorted are in the device memory and sorts the elements in place. Their memory-based approach does not work efficiently on streaming data that arrive over multiple cycles. Zhang and Zheng [28] implement another iterative sorting method that uses systolic arrays to sort data in memory. Although their approach scales well, it requires special memory hardware. Olariu et al. [17] present a hardware algorithm for sorting N values by repeatedly using a fixed-size P-input sorting network that processes P values each cycle. They show that their algorithm achieves optimal overall performance of

$$\Theta\left(\frac{N \times \log_2 N}{P \times \log_2 P}\right)$$

provided the P-input sorting network has a depth of $O(\log_2^2 P)$ such as bitonic sorting networks. In this paper, we instead focus on iterative max-set-selection units. The main differences between our work and previous research include: 1) our designs are optimized for the case in which only M outputs from N inputs are needed; 2) our designs avoid using additional storage or intermediate memory blocks by receiving the appropriate number of input values each cycle; and 3) our designs iteratively utilize max-set-selection units, rather than complete sorting units, which leads to improved area and latency.

As shown in Fig. 16, our proposed iterative max-set-selection units utilize R-to-M bitonic or odd-even merge max-set-selection units of varying pipeline depths as functional cores. Each design has a finite state machine (FSM) that manages three sequential phases of the execution pipeline: Warm up, steady state, and completion. The warm-up phase occurs when the first P input values arrive and begin to propagate through the core max-set-selection unit's pipeline, but before any intermediate results are generated. When the core max-set-selection unit outputs data from the first set of input elements, the steady-state phase begins. During each cycle of the steady-state phase, a set of P new input values arrives at the inputs and a set of M intermediate result elements are produced and then immediately consumed by the core max-set-selection unit, where $R = P + M$. In this phase, the intermediate results are fed back into the core max-set-selection unit with the new input values to be sorted. Once all the inputs have been received and applied to the core max-set-selection unit, the completion phase begins, in which intermediate result values are stored at the inputs of the sorting unit as the core max-set-selection unit produces them. Once R values are stored, they are sent to the core max-set-selection unit. This process is completed with a final max-set-selection run with R or fewer remaining valid values, resulting in the final M outputs.

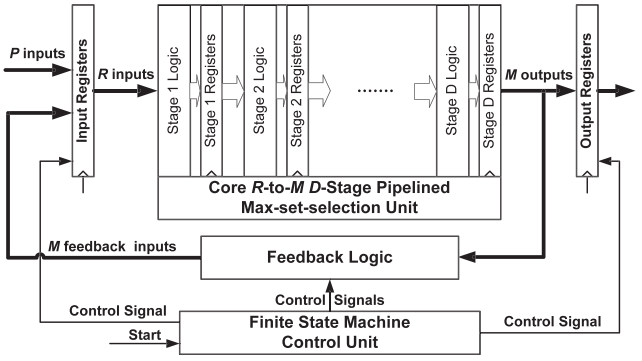

Fig. 16. Iterative max-set-selection unit.

5.1 Discussion

As shown in Fig. 16, we use $R = P + M$ input registers to buffer P input values and M intermediate result values. Removing that level of registers decreases the total number of cycles (latency) to generate the final result. However, it also decreases the overall frequency of the design by adding the delay from the feedback logic into the critical path of the first stage of the core max-set-selection unit. Thus, our designs use input registers to increase the frequency.

Including input registers, the latency of our iterative max-set-selection designs in terms of clock cycles is bounded by

$$Latency = \lceil N/P \rceil + \left\lceil (D+1)^2 \times M/(P+M) \right\rceil + D + 1, \tag{1}$$

where P is the number of new input values received each cycle by the core max-set-selection unit, N is the total number of input values, D is the pipeline depth of the core max-set-selection unit, M is the number of outputs from the core max-set-selection unit, and $\lceil \cdot \rceil$ denotes the ceiling operation. It is important to note that N, P, and M define the problem. The first term of the equation accounts for the cycles required to receive all the inputs, both during the warm-up phase and the steady-state phase. The remaining terms describe the bound on how many cycles the completion phase takes.

TABLE 5
Performance and Resource Requirements of Iterative Max-Set-Selection Units Used to Find the Four Largest Data Values from $N = 256$ Data Inputs with 10-Bit Unsigned CAE Blocks Using a TSMC 65-nm Standard-Cell Library

Core Max-set-selection Unit (R-to-M)	Pipeline Depth (D)	# of New Data Elements per Cycle (P)	Total Cycles	Frequency (MHz)		Combinational Area (μm^2)		Non-combinational Area (μm^2)		End-to-end Latency (ns)	
				Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
8-to-4	4	4	81	2,702	2,857	3,013	3,222	4,386	4,249	29.97	28.35
8-to-4	2	4	72	1,923	2,000	3,640	3,069	2,567	2,429	37.44	36.00
8-to-4	1	4	68	1,408	1,587	4,837	4,306	1,756	1,666	48.28	42.84
16-to-4	7	12	46	2,632	2,439	6,289	5,590	11,119	9,098	17.48	18.86
16-to-4	4	12	36	2,040	2,083	9,561	8,264	6,398	6,160	17.64	17.28
16-to-4	3	12	30	1,724	1,724	10,801	10,219	6,336	6,130	17.40	17.40
32-to-4	10	28	39	2,174	2,083	10,197	8,191	19,849	17,541	17.94	18.72
32-to-4	5	28	21	1,388	1,388	12,148	11,371	11,339	11,141	15.12	15.12
32-to-4	4	28	19	1,052	1,051	12,352	10,152	9,303	9,467	18.05	18.05

As an example, a design with a seven-stage, 16-to-4 max-set-selection core unit is used to iteratively perform max-set-selection on 256 inputs. The design parameters are $R = 16$, $M = 4$, $N = 256$, and $D = 7$, which implies $P = 12$, because $R = P + M$. The warm-up phase lasts seven cycles in which 12 new inputs are applied each cycle. At the end of the warm-up phase, the input registers and seven pipeline stages each have a set of 12 values being sorted to find the largest four. The steady-state phase begins when the first input set's four largest values are reapplied to the inputs of the core unit along with another 12 completely new inputs. This phase lasts 15 cycles until all 256 inputs are received. The completion phase now forms new sets of 16 values by concatenating (over four cycles) the first next four intermediate output sets and reapplying them to the input of the core unit. The second next four intermediate outputs sets, which were originally in the input registers and first three stages of the pipeline, also form another new set of 16 values in four cycles. After another eight cycles, two output sets of the previous concatenations are reapplied to the core unit. The four largest values from the entire 256 inputs are ready after eight more cycles, for a total of 24 cycles in the completion phase and 46 cycles for the entire process.

An iterative max-selection unit performs a significant percentage of the work as it is receiving input values. More overlapping of computation with data reception (during the warm-up and steady-state phases) occurs by increasing the total number of input values. Hence, for a large data stream, such as $N = 8192$, the latency is dominated by the N/P term in (1).

5.1.1 Comparison with Parallel Max-Set-Selection Units

Equation (1) demonstrates that the latency of the iterative max-set-selection units scales linearly with the total number of inputs to sort (N), rather than $O(\log_2^2 N)$, as with parallel max-set-selection units. However, in situations in which P is significantly smaller than N (e.g., I/O bandwidth-limited situations), a parallel max-set-selection unit, which operates on all the input values at one time, must first buffer the input values as they arrive, and then propagate the values through the pipeline only once all input values are present. Moreover, the number of logic inputs of a parallel max-set-selection unit (R) should be as large as the number of data values to sort (N). This imposes a huge impact on the area, because parallel max-set-selection units have a hardware complexity of $O(N \times (\log_2^2 M))$. When the area of a parallel max-set-selection unit can be tolerated, a parallel max-set-selection unit achieves higher throughput and lower latency than an iterative max-set-selection unit.

5.1.2 Iterative Partial Sorting Units

An iterative max-set-selection unit can easily be converted to an iterative partial sorting unit by augmenting the iterative unit with an M-input sorting unit to sort the final M values. This modification increases the latency by $O(\log_2^2 M)$, if a bitonic or odd-even merge sorting unit is used.

5.2 Results

To illustrate the potential tradeoffs that can be made with iterative max-set-selection-units, 18 different iterative max-set-selection units are designed to find the four largest values from a stream of input values. Table 5 summarizes the latency and resource requirements of each of the iterative max-set-selection units when they are used to find the four largest values from 256 10-bit input values. The units are synthesized using the TSMC 65-nm standard cell library. These designs use our proposed pipelined 8-to-4, 16-to-4, and 32-to-4 max-set-selection units, described in Section 3, as functional cores for the iterative units. By a simple modification to the FSM, our iterative technique can be employed to find the M largest values from N input values using an R-to-M max-set-selection unit as a functional core for any integer values of N, R, and M for which $N > R > M \geq 1$ and $R = P + M$.

The area benefit of an iterative max-set-selection over a parallel max-set-selection unit increases linearly with a linear increase in the total number of inputs. For example, in the case of a data stream of $N = 256$ inputs and where only $P = 12$ new inputs can be received each cycle, an iterative bitonic max-set-selection unit using a simple 16-to-4 max selection unit with a pipeline depth of four has a latency of 17.64 ns, whereas a parallel bitonic 256-to-4 max-selection unit with a pipeline depth of seven has a latency of 5.39 ns (for the lowest latency 256-to-4 bitonic unit) as indicated in Tables 3 and 5. However, the iterative max-set-selection unit is more than 13 times smaller than the corresponding parallel unit. Thus, at a small fraction of the resource requirements, the iterative max-set-selection unit could provide reasonable latency and throughput for certain target applications.

TABLE 6
Complexity of Sorting Algorithms (N: Total Number of Inputs, M: Number of Outputs, P: Number of New Elements per Cycle)

Algorithm	Latency	Area	Throughput
Complete sorting network ($N = M = P$) [25]	$O((\log N)^2)$	$O(N \times (\log N)^2)$	$O(N)$
Partial sorting network ($N = P$) *	$O(\log N \times \log M)$	$O(N \times (\log M)^2)$	$O(N)$
Iterative complete sorting network ($N = M$) [17]	$\Theta(\frac{N \times \log N}{P \times \log P})$	$O(P \times (\log P)^2)$ **	$O(P)$
Iterative partial sorting network *	$O(N/P)$	$O((P + M) \times (\log(P + M))^2)$	$O(P)$
Linear complete sorter ($N = M$) [29], [31]	$O(N)$	$O(N)$	$O(1)$
Linear partial sorter [45]	$O(N)$	$O(M)$	$O(1)$

* Our proposed sorting networks.
** It also requires N memory words.

6 Related Research

Sorting algorithms for shared-memory multiprocessor systems and VLSI implementations have been investigated during the past 40 years. Linear array structures and sorting networks are two types of architectures that have been widely used for hardware implementations. Linear sorters that use insertion techniques have been proposed and implemented as FPGA and VLSI designs [29], [30], [31], [32], [33], [34]. Although they are simple to implement and their area complexity is reasonable, linear array structures are not able to process blocks of data in parallel, resulting in low-throughput designs [33]. However, linear array structures are good candidates for sorting streaming or serial data when only one new element is sent to or one sorted element is retrieved from the sorting unit per clock cycle. Linear sorters have a time complexity in the range $O(b(N))$ to $O(N)$, where b is the bit width of input values. Compared to sorting networks, the higher time complexity of linear sorters hinders their use in fast, high-throughput sorting applications.

On the other hand, sorting networks, which use multiple levels of parallel CAE blocks to rearrange data, are suitable architectures for sorting huge amounts of parallel data. Moreover, customizable pipelined sorting networks can meet the requirements of high-throughput applications. Hardware-implemented sorting networks have a time complexity of $O(\log_2^2 N)$, which make them high-speed architectures for large values of N. While theoretical time-optimal $O(\log_2 N)$ sorting networks have also been proposed [35], [36], [37], they cannot be implemented in hardware because of their large hidden constants in O-notation [38].

6.1 Sorting Networks

From complex cube and mesh array structures to linear array structures, and from theoretical log-depth algorithms to practical linear and $\log^2$-depth algorithms, Batcher's compare-exchange sorting networks are of importance for hardware implementations because of their ease of VLSI realization. Since their advent, many sorting designs have evolved from Batcher's algorithms for shared-memory systems and VLSI hardware [38], [39], [40]. Previous research on sorting networks falls into two main categories: Sorting algorithms and sorting architectures.

Extensive research has been performed to optimize parallel sorting algorithms under various implementation assumptions and for different applications. Herruzo et al. [41] propose a novel odd-even merge sorting algorithm based on a divide-and-conquer strategy for shared-memory multiprocessor systems. Ionescu and Schauser [42] propose a parallel bitonic sorting algorithm for coarse-grain parallel machines to optimize communication steps and local computations. Agrawal [6] presents a scheme to design arbitrary-sized bitonic sorting networks. He shows that his method can efficiently be used in the design of asynchronous transfer mode (ATM) switches. Kuo and Huang [21] introduce a modified odd-even merge sorting network for an arbitrary number of inputs. Their modular approach can be used to implement custom sorting units in hardware.

Significant research has also investigated sorting architecture designs for VLSI and FPGA implementation. Latency, throughput, scalability, and resource requirements are the main factors considered for these hardware implementations. Ratnayake and Amer [13] present an FPGA-based implementation of a modified counting sort algorithm that is used to sort large amounts of data. Layer et al. [16] study the hardware implementation of an iterative sorting unit that provides tradeoffs between data throughput and area, and they present a pipelined architecture that utilizes multilevel bitonic sorting networks on FPGAs. Lee and Batcher [38] propose a novel recirculating bitonic sorting network made up of a level of CAE blocks followed by an Ω-network of $\log_2 N - 1$ switch levels. The purpose of the recirculating network is to reduce the area complexity of the original bitonic sorting networks to $O(N \times \log N)$. Although the proposed network theoretically has the same time complexity as the original bitonic sorting networks, the latency of the switches may degrade the performance of the sorter. The interested reader may refer to [39], [40] for a more detailed survey of hardware sorting algorithms and architectures.

6.2 Partial Sorting and Max-Set-Selection Units

In some cases, partial sorting units are needed to find the M largest (or smallest) numbers from N numbers in sorted order, where $M < N$. Max-set-selection is a related operation that outputs the M largest numbers but not necessarily in sorted order. Partial sorting and selection algorithms have previously been addressed by a wealth of software approaches [43]. However, hardware designs for partial sorters and selection algorithms have barely been discussed in the literature.

Linear sorters, with slight modifications, are capable of performing partial sorting [44]. Perez-Andrade et al. [45]

propose a linear FIFO-based sorter to partially sort an arbitrary number of input values. Their algorithm discards the smallest sorted element on receiving a new element and finds a position for the new element. Colavita et al. [46] propose a VLSI sorting architecture for streaming data. Their regular design consists of several small elementary sorting units, and its area and latency increase linearly with the number of values to sort. Dong et al. [12] propose a parallel partial sorting design using internal FPGA blocks to improve the performance of normalized cross-correlation image matching systems.

Each of these partial sorters are based on linear arrays, resulting in a time complexity of $O(N)$. To find/sort the M largest numbers from N numbers, our proposed partial sorters, which are based on sorting networks, have a time complexity of $O(\log_2 N \times \log_2 M)$ and area complexity of $O(N \times \log_2^2 M)$, compared with the $O(\log_2^2 N)$ time complexity and $O(N \times \log_2^2 N)$ area complexity of the original bitonic and odd-even merge sorting networks. Our sorters also have better latency, frequency, and throughput than the partial sorters presented above, but our sorters have higher resource requirements as summarized in Table 6.

7 Conclusions

The paper has presented the design and implementation of flexible, low-latency, high-throughput N-to-M sorting, and max-set-selection units and discussed the structure, performance and resource requirements of these units. In this paper, we propose modular techniques for designing N-to-M sorting and max-set-selection units based on the Batcher's bitonic and odd-even merge sorting algorithms. We present new regular bitonic merging units that are used to construct efficient sorting and max-set-selection units. Although built from Batcher's merging units, our proposed parallel designs modify the original units to obtain efficient max-set-selection and partial sorting units, reducing time and area complexities of the original algorithm to $O(\log_2 N \times \log_2 M)$ and $O(N \times \log_2^2 M)$, respectively. The analysis performed shows that our designs have lower latency and area than previous designs. For instance, a 256-to-4 max-set-selection unit is more than two times faster and five times smaller than the corresponding 256-input complete sorting network.

We employ a modular design methodology that allows our units to be readily utilized in applications with different requirements. Our units meet stringent latency and throughput constraints, are suitable for a wide range of applications, and give designers the flexibility to easily change the sorter architecture. Moreover, our designs can be applied to two's complement or floating-point numbers by simply changing the comparators used in the CAE blocks.

Parallel max-set-selection units have high I/O bandwidth and resource requirements. To reduce I/O bandwidth and area, we propose an iterative max-set-selecting method that receives P new input values per cycle. Our iterative design reuses a small max-set-selection unit over a number of iterations to generate the outputs. The proposed iterative units, which have time and area complexities of $O(N/P)$ and $O((P+M) \times \log_2^2(P+M))$, have much lower resource and I/O requirements than the corresponding parallel units. The iterative max-set-selection units target applications that require selection units with moderate latency and throughput, but need low area and I/O.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF), under grant #0824040. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] S. Azuma, T. Sakuma, T. Takeo, T. Ando, and K. Shirai, "Diaprism Hardware Sorter - Sort a Million Records within a Second," http://sortbenchmark.org/Y2000_Datamation_DiaprismSorter.pdf, 2000.

[2] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha, "GPUTeraSort: High Performance Graphics Co-Processor Sorting for Large Database Management," *Proc. Conf. Management of Data,* pp. 325-336, 2006.

[3] D. Koch and J. Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-Time Reconfiguration on FPGAs for Large Problem Sorting," *Proc. Symp. Field Programmable Gate Arrays,* pp. 45-54, 2011.

[4] D. Pok, C.-I. Chen, J. Schamus, C. Montgomery, and J. Tsui, "Chip Design for Monobit Receiver," *IEEE Trans. Microwave Theory and Techniques,* vol. 45, no. 12, pp. 2283-2295, Dec. 1997.

[5] I. Pitas and A.N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and Applications.* Kluwer Academic Publishers, 1990.

[6] J.P. Agrawal, "Arbitrary Size Bitonic (ASB) Sorters and Their Applications in Broadband ATM Switching," *Proc. IEEE Int'l Conf. Computers and Comm.,* pp. 454-458, Mar. 1996.

[7] K. Yun, K. James, R. Fairlie-Cuninghame, S. Chakraborty, and R. Cruz, "A Self-Timed Real-Time Sorting Network," *IEEE Trans. Very Large Scale Integration Systems,* vol. 8, no. 3, pp. 356-363, June 2000.

[8] A. Colavita, E. Mumolo, and G. Capello, "A Novel Sorting Algorithm and Its Application to a Gamma-Ray Telescope Asynchronous Data Acquisition System," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment,* vol. 394, no. 3, pp. 374-380, 1997.

[9] D.C. Stephens, J.C. Bennett, and H. Zhang, "Implementing Scheduling Algorithms in High-Speed Networks," *IEEE J. Selected Areas in Comm,* vol. 17, no. 6, pp. 1145-1158, June 1999.

[10] V. Brajovic and T. Kanade, "A VLSI Sorting Image Sensor: Global Massively Parallel Intensity-to-Time Processing for Low-Latency, Adaptive Vision," *IEEE Trans. Robotics and Automation,* vol. 15, no. 1, pp. 67-75, Feb. 1999.

[11] C. Chakrabarti and L.-Y. Wang, "Novel Sorting Network-Based Architectures for Rank Order Filters," *IEEE Trans. Very Large Scale Integration Systems,* vol. 2, no. 4, pp. 502-507, Dec. 1994.

[12] S.-N. Dong, X.-T. Wang, and X.-B. Wang, "A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA," *Proc. Int'l Cong. Image and Signal Processing,* pp. 1-4, Oct. 2009.

[13] K. Ratnayake and A. Amer, "An FPGA Architecture of Stable-Sorting on a Large Data Volume: Application to Video Signals," *Proc. Ann. Conf. Information Sciences and Systems,* pp. 431-436, 2007.

[14] A. Gregerson, M. Schulte, and K. Compton, "High-Energy Physics," *Handbook of Signal Processing Systems,* pp. 179-211, Springer, 2010.

[15] R. Marcelino, H.C. Neto, and J.M.P. Cardoso, "Sorting Units for FPGA-Based Embedded Systems," *Proc. IFIP Cong. Distributed Embedded Systems: Design, Middleware and Resources,* pp. 11-22, Sept. 2008.

[16] C. Layer, D. Schaupp, and H.-J. Pfleiderer, "Area and Throughput Aware Comparator Networks Optimization for Parallel Data Processing on FPGA," *Proc. Int'l Symp. Circuits and Systems,* pp. 405-408, May 2007.

[17] S. Olariu, M.C. Pinotti, and S.Q. Zheng, "An Optimal Hardware-Algorithm for Sorting Using a Fixed-Size Parallel Sorting Device," *IEEE Trans. Computers,* vol. 49, no. 12, pp. 1310-1324, Dec. 2000.

[18] V. Pedroni, R. Jasinski, and R. Pedroni, "Panning Sorter: A Minimal-Size Architecture for Hardware Implementation of 2D Data Sorting Coprocessors," *Proc. Asia Pacific Conf. Circuits and Systems,* pp. 923-926, 2010.
[19] A. Farmahini-Farahani, A. Gregerson, M. Schulte, and K. Compton, "Modular High-Throughput and Low-Latency Sorting Units for FPGAs in the Large Hadron Collider," *Proc. IEEE Int'l Symp. Application Specific Processors,* pp. 38-45, June 2011.
[20] G. Graefe, "Implementing Sorting in Database Systems," *ACM Computing Survey* vol. 38, article 10, Sept. 2006.
[21] C.J. Kuo and Z.W. Huang, "Modified Odd-Even Merge-Sort Network for Arbitrary Number of Inputs," *Proc. IEEE Int'l Conf. Multimedia and Expo,* pp. 929-932, Aug. 2001.
[22] T. Nakatani, S.-T. Huang, B. Arden, and S. Tripathi, "K-Way Bitonic Sort," *IEEE Trans. Computers,* vol. 38, no. 2, pp. 283-288, Feb. 1989.
[23] X. Hongwei and X. Yafeng, "An Improved Parallel Sorting Algorithm for Odd Sequence," *Proc. Int'l Conf. Advanced Computer Theory and Eng.,* pp. 356-360, Dec. 2008.
[24] K.J. Liszka and K.E. Batcher, "A Generalized Bitonic Sorting Network," *Proc. Int'l Conf. Parallel Processing,* pp. 105-108, Aug. 1993.
[25] K.E. Batcher, "Sorting Networks and Their Applications," *Proc. AFIPS Proc. Spring Joint Computer Conf.,* pp. 307-314, 1968.
[26] C. Lefevre, "LHC: The Guide," Jan. 2008.
[27] C.-Y. Huang, G.-J. Yu, and B.-D. Liu, "A Hardware Design Approach for Merge-Sorting Network," *Proc. IEEE Int'l Symp. Circuits and Systems,* vol. 4, pp. 534-537, May 2001.
[28] Y. Zhang and S.Q. Zheng, "An Efficient Parallel VLSI Sorting Architecture," *VLSI Design,* vol. 11, no. 2, pp. 134-147, 2000.
[29] B. Ahn and J.M. Murray, "A Pipelined, Expandable VLSI Sorting Engine Implemented in CMOS Technology," *Proc. IEEE Int'l Symp. Circuits and Systems,* pp. 134-137, May 1989.
[30] L. Yen-Chun, "On Balancing Sorting on a Linear Array," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 5, pp. 566-571, May 1993.
[31] S.W. Moore and B.T. Graham, "Tagged Up/Down Sorter - A Hardware Priority Queue," *The Computer J.,* vol. 38, pp. 695-703, Sept. 1995.
[32] B. Parhami and D.-M. Kwai, "Data-Driven Control Scheme for Linear Arrays: Application to a Stable Insertion Sorter," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 1, pp. 23-28, Jan. 1999.
[33] J. Ortiz and D. Andrews, "A Configurable High-Throughput Linear Sorter System," *Proc. Symp. Parallel Distributed Processing,* pp. 1-8, 2010.
[34] T. Demirci, I. Hatirnaz, and Y. Leblebici, "Full-Custom CMOS Realization of a High-Performance Binary Sorting Engine with Linear Area-Time Complexity," *Proc. Symp. Circuits and Systems,* vol. 5, pp. 453-456, 2003.
[35] M. Ajtai, J. Komlós, and E. Szemerédi, "An O(n Log n) Sorting Network," *Proc. Ann. ACM Symp. Theory of Computing,* pp. 1-9, May 1983.
[36] T. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Computers,* vol. C-34, no. 4, pp. 344-354, Apr. 1985.
[37] M. Paterson, "Improved Sorting Networks with O(log N) Depth," *Algorithmica,* vol. 5, no. 1, pp. 65-92, 1990.
[38] J.-D. Lee and K.E. Batcher, "Minimizing Communication in the Bitonic Sort," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 5, pp. 459-474, May 2000.
[39] D. Bitton, D.J. DeWitt, D.K. Hsaio, and J. Menon, "A Taxonomy of Parallel Sorting," *ACM Computing Surveys,* vol. 16, no. 3, pp. 287-318, 1984.
[40] C.D. Thompson, "The VLSI Complexity of Sorting," *IEEE Trans. Computers,* vol. C-32, no. 12, pp. 1171-1184, Dec. 1983.
[41] E. Herruzo, G. Ruiz, J.I. Benavides, and O. Plata, "A New Parallel Sorting Algorithm Based on Odd-Even Mergesort," *Proc. Int'l Conf. Parallel, Distributed and Network-Based Processing,* pp. 18-22, 2007.
[42] M.F. Ionescu and K.E. Schauser, "Optimizing Parallel Bitonic Sort," technical report, Univ. of California at Santa Barbara, 1997.
[43] D.E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching,* second ed. Addison-Wesley, May 1998.
[44] L. Ribas, D. Castells, and J. Carrabina, "A Linear Sorter Core Based on a Programmable Register File," *Proc. Conf. Design of Circuits and Integrated Systems,* pp. 635-640, 2004.
[45] R. Perez-Andrade, R. Cumplido, F. Del Campo, and C. Feregrino-Uribe, "A Versatile Linear Insertion Sorter Based on a FIFO Scheme," *Proc. IEEE CS Ann. Symp. Very Large Scale Integration (VLSI),* pp. 357-362, Apr. 2008.
[46] A. Colavita, A. Cicuttin, F. Fratnik, and G. Capello, "SORTCHIP: A VLSI Implementation of a Hardware Algorithm for Continuous Data Sorting," *IEEE J. Solid-State Circuits,* vol. 38, no. 6, pp. 1076-1079, June 2003.

Amin Farmahini-Farahani received the BS degree in computer engineering from Iran University of Science and Technology and the MS degree in electrical and computer engineering from the University of Tehran, and is working toward the PhD degree at the Electrial and Computer Engineering Department of the University of Wisconsin-Madison. His research interests include computer architecture, embedded processing, and reconfigurable computing.

Henry J. Duwe III received the BS degree in applied mathematics, engineering, and physics from the University of Wisconsin-Madison, and is currently a graduate student at the University of Illinois at Urbana-Champaign. His research interests include high-performance computer architectures and runtime systems.

Michael J. Schulte received the BS degree in electrical engineering from the University of Wisconsin-Madison, and the MS and PhD degrees in electrical engineering from the University of Texas at Austin. He is currently a fellow design engineer with AMD Research and an associate professor (on leave) in electrical and computer engineering with the University of Wisconsin-Madison. His research interests include domain-specific processors, heterogeneous computing, computer arithmetic, hardware acceleration, and computer architecture. He received the US National Science Foundation (NSF) CAREER Award, the Alfred Nobel Robinson Award, and the Frank Hook Assistant Professorship. He has served an associate editor for the *IEEE Transactions on Computers* and the J*ournal of VLSI Signal Processing*, and as a program chair and general chair for the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), the IEEE International Symposium on Computer Arithmetic (ARITH), and the Asilomar Conference on Signals, Systems and Computers. He is a senior member of the IEEE.

Katherine Compton received the BS, MS, and the PhD degrees from Northwestern University, in 1998, 2000, and 2003, respectively. She is currently an associate professor at the University of Wisconsin-Madison in the Department of Electrical and Computer Engineering. She and her graduate students are investigating new architectures, logic structures, integration techniques, and systems software techniques for reconfigurable computing. She serves on a number of program committees for FPGA and reconfigurable computing conferences and symposia. She is a member of both the IEEE and the ACM.