Design of memory efficient FIFO-based merge sorter

Youngil Kim\textsuperscript{a)}, Seungdo Choi, and Yong Ho Song

Department of Electronics and Computer Engineering, Hanyang University,
222 Wangsimini-ro, Seongdong-gu, Seoul 133–791, Korea

a) yikim@enc.hanyang.ac.kr

Abstract: Sorting is an important operation used in various applications including image processing and databases. It represents a large portion of the total execution time of these applications. To improve the performance of sort operations, a dedicated hardware sorter can be used. When implemented in hardware, a FIFO-based merge sorters often shows excellent hardware resource utilization efficiency but requires high buffer memory usage. In this paper, we presents a cost-effective hardware architecture of a FIFO-based merge sorter. Our proposed architecture minimizes buffer memory requirement. We evaluate the design by implementing the architecture on an FPGA platform. FPGA synthesis results show that the proposed approach reduces the average flip-flop and LUT-RAM by 5\% and 14\%, respectively.

Keywords: sorting, accelerator architectures, FPGAs

Classification: Integrated circuits

References

[10] M. Bednara, \textit{et al.}: “Tradeoff analysis and architecture design of a hybrid...


1 Introduction

Many applications use various sort operations to reorder a set of data based on a specific rule. Some popular examples include database and big data systems. Database systems sort data to process during many data manipulation operations [1, 2]. And MapReduce, which is also used as a parallel programming model, heavily relies on sort operations to organize intermediate data [3].

In a data management system, the performance of the entire system depends on the performance of the sort algorithm [1]. In the case of MapReduce, the sort phase constitutes a large portion of the overall execution time in addition to the map and partition phases [3]. In [4], it is revealed that the sort phase can account for more than 90% of the total execution time depending on the application. Therefore, it is very important to enhance the performance of sort operations for these types of applications.

Many studies have been conducted on improve the performance of sort operation using hardware accelerators [5, 6, 7, 8, 9, 10, 11]. The accelerator implements software sort algorithms in hardware so that it can sort data independently of the processor. The parallelized deployment of accelerators improve the performance of sort operations even more.

There are various types of hardware sort accelerators, depending on target algorithms and implementation methodology. There have been the accelerators aiming to enhance the FIFO-based merge sort operation as well [5, 7, 8]. They usually achieve higher hardware resource utilization compared to other types of sorters in implementing algorithm logic [12]. However, the use of FIFO buffers requires a lot of memory space. As reported in [6] and [12], a large portion of total hardware resources consumed by FIFO-based merge sorter is memory resource. In addition, there is also additional requirement that the data in FIFO should be sorted before being merged. To meet this requirement, a smaller FIFO is placed to provide sorted data to each FIFO, which further increases the memory requirement. Moreover, previous studies assume the data count to be $2^k$ for a FIFO-based merge sorter [5, 8], not considering the case where the number of maximum input data to be sorted is not a power of two. In this paper, we propose a hardware sort accelerator for a FIFO-based merge sorter that considers the above case. The proposed technique optimizes the structure of the sorter to minimize memory requirement.

To evaluate the effectiveness of the proposed technique, we design a sorter capable of sorting multiple data by cascading several FIFO-based merge sorters and apply a memory minimization technique to it. In addition, this study demonstrates
the validity of the proposed technique using mathematical induction. The operation of the sorter is verified through the FPGA platform and the memory resource usage on the FPGA is measured using the same platform. Experimental results show that the sorter with the memory minimization technique consumes an average of 5% fewer flip-flops and 14% fewer LUT-RAMs than the sorter without the proposed technique.

This paper is organized as follows. Section 2 briefly describes the structure of a FIFO-based merge sorter and cascading FIFO-based merge sorter. The architecture of the baseline sorter and the problem with this architecture is detailed in Section 3. Section 4 presents a memory optimized sorter architecture based on the analysis of the cascading FIFO-based merge sorter. Section 5 shows the experimental results. Section 6 concludes this work.

2 Sorting unit architectures

2.1 FIFO-based merge sorter

A FIFO-based merge sorter (FMS) merges ordered data sets into a single ordered one. Fig. 1 illustrates an organization of FMS which consists of two input FIFOs, one output FIFO, a comparator, and a multiplexer. The input FIFOs store two groups of sorted data. The comparator determines which FIFO the next data should be fetched from. Depending on the comparison results, either a larger or a smaller value is selected and passed through the multiplexer all the way to the output FIFO. Basically one data element is passed to the output FIFO every clock cycle.

As aforementioned, the data stored in the input FIFOs must be sorted in ascending or descending order before getting merged. That is, each data group is an output of a sort operation. Many sorting methods can be used for pre-sorting input data. For example, one can use a software sorting method or use hardware sorting methods such as an insertion sorting module or a sorting network [12].

2.2 Cascading FIFO-based merge sorter

The cascading FIFO-based merge sorter (CFMS) meets the pre-sorting requirement mentioned above using serial instances of the FMS. Fig. 2 shows the organization of CFMS. This sorter is composed of several FMS’s, each connected to a demultiplexer. The combination of one FMS and its demultiplexer is defined as a merge stage. The FMS of each merge stage feeds output data to the input FIFOs of the next stage.

The CFMS can sort the entire input data set regardless of the alignment of the input data. The data to be sorted is stored in the first-stage input FIFOs. The data processed at each stage is passed to the input FIFOs of the next stage. When the first input FIFO is full, each FMS can start sort operations immediately after the first input value of the second FIFO is stored. In this manner, the sort operation of each stage can be pipelined [5]. At each step, $N$ comparison operations must be repeated to sort $N$ data.

As the FMS consists of one comparator and one multiplexer, the hardware cost required for hardware logic is low. However, the main hardware component of the FMS is the FIFO, which consists of memory. For this reason, a large amount of
memory resources is required to implement CFMS, which is a combination of multiple FMS's. Therefore, the amount of memory used by this sorter should be minimized.

3 Motivation and problem definition

3.1 Baseline cascading FIFO-based merge sorter

A baseline FMS, called B.FMS [6], has two input FIFOs of the same size and one output FIFO of twice the size of an input FIFO. Several studies have presented CFMS using the B.FMS [5, 8]. The feature of this type of CFMS is that the input FIFO at each stage is twice the size of an input FIFO in the previous stage. In this paper, CFMS with this structure will be denoted as a baseline CFMS, or B.CFMS.

This baseline sorter takes two groups of \(2^k\) data in the \(k\)-th stage and generates \(2^{k+1}\) sorted data to be transferred to the next stage. Therefore, the baseline sorter with \(n\) stages will eventually generate \(2^n\) sorted data. Fig. 3 illustrates the structure of B.CFMS and the data flow at each stage.

3.2 Problem of baseline cascading FIFO-based merge sorter

CFMS is used in various applications under different conditions. In fact, the number of data belonging to the data groups merged by the sorter is not always \(2^n\). For the baseline sorter, when the maximum number of data to be sorted is not a power of two, the sorter must use more memory than is needed to sort the data. For
example, assume that \( N \) data enter the sorter and \( N \) is greater than 256 but less than 512. To sort the \( N \) data, the sorter must be designed to generate a maximum of 512 sorted data with at least 9 stages. The ninth stage has two input FIFOs with 256 entries, respectively. The first input FIFO of the ninth stage stores 256 data sorted in the previous stage. However, only the remaining \((N-256)\) sorted data are stored in the second FIFO. In the ninth stage, one group of 256 data and one group of \((N-256)\) data are merged to finally generate \( N \) sorted data. In this case, the second input FIFO uses only \((N-256)\) entries out of 256 entries, which is a waste of memory resources.

For this reason, it is appropriate to re-design CFMS to minimize memory consumption when the number of input data is not a power of 2. That is, the sorter needs to have \( n \) stages to sort up to \( 2^n \) data. However, it is not necessary that the input FIFO size of the \( k \)-th stage be \( 2^{k-1} \) or that the two input FIFOs have the same number of entries.

4 Proposed method

4.1 Analysis of cascading FIFO-based merge sorter

To discuss memory minimization of CFMS, the sort operation should be analyzed first. It is assumed that the sorter to be analyzed consists of \( n \) stages and that each input FIFO of the first stage has one entry and finally produces \( S \) sorted data.

With this assumption, we observe the following things. First, to sort \( K \) data at a stage, the total number of entry FIFOs belonging to that stage should be equal to or larger than \( K \). This is because for the FMS to generate \( K \) sorted data, where the number of data belonging to two data groups to be merged is \( K \), the size of an input FIFO must be at least the same size as the data group to be stored. Second, when any input FIFOs have \( M \) and \( N \) entries respectively, the sum of the entries of the two input FIFOs belonging to the previous stage is greater than the larger of \( M \) and \( N \). The reason for this is that when the input of any stage is \( M \) and \( N \), the previous stage must be able to generate \( M \) and \( N \) sorted data and transmit it to the next stage. Finally, regardless of the number of stages of the CFMS, the total entry sum of the input FIFOs in the last stage is \( S \) or more.

![Baseline cascading FIFO-based merge sorter and its data flow](image_url)
Fig. 4 depicts the data flow in the last three stages of the CFMS. Each letter represents the depth of the input FIFOs of each stage (e.g., in alphabetical order), and ‘+’ indicates a sorting operation. According to the analysis, each variable has the following relationship.

- $N + M \geq \max(O, P)$
- $O + P \geq \max(Q, R)$
- $Q + R \geq S$

As the purpose of this study is to minimize the size of the FIFO, we exclude the case where the size of the input FIFO storing the data group is larger than the size of the data group.

### 4.2 Memory minimization

We assume that the CFMS is the same as the sorter presented earlier in the CFMS analysis. The depths of the input FIFOs in the $k$-th stage are defined as $a_k$ and $b_k$, respectively. For convenience, we assume $a_k \geq b_k$. Also assume that $a_{n+1}$ is the entry of the final output FIFO of the sorter. The assumptions and analysis of the previous section can then be summarized as follows.

\[
\begin{align*}
    a_1 &= 1, \quad b_1 = 1 \\
    a_k &= a_{k-1} + b_{k-1} \\
    S &= a_n + b_n = a_{n+1} \\
    a_k &\geq b_k \\
    a_k \in \mathbb{N}, \quad b_k \in \mathbb{N}
\end{align*}
\]

The goal is to minimize the total number of entries in the FIFO that make up the CFMS. The total number of FIFO entries in the CFMS consists of $n$ stages as follows.

\[
\sum_{k=1}^{n} (a_k + b_k)
\]

Using the above assumptions, we can derive the following equations.

\[
\begin{align*}
    S &= a_n + b_n \\
    a_n &= a_{n-1} + b_{n-1} \\
    S &= a_{n-1} + b_{n-1} + b_n \\
    &\vdots
\end{align*}
\]
\[ a_1 + b_1 + b_2 + \cdots + b_n = a_1 + \sum_{k=1}^{n} b_k \]

Therefore, the total sum of FIFO entries is as follows.

\[
\begin{align*}
\sum_{k=1}^{n}(a_k + b_k) &= \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \\
&= \sum_{k=1}^{n} a_k + S - a_1 
\end{align*}
\]

(6) and (7) imply that the total sum of \(a_k\) should be minimized to minimize the total sum of FIFO entries. In conclusion, the condition of \(a_k\) to minimize \(\sum_{k=1}^{n}(a_k + b_k)\) is as follows. Variable \(t\) denotes the exponent value of the power of two that is greater than \(S\) and closest to \(S\). It changes according to the maximum number of data that the sorter can handle.

**Theorem 1.** For \(t \in \mathbb{N}\), interval \((1, \text{inf})\) can be covered with the non-overlapping half-closed interval \((2^{t-1} < S \leq 2^t)\). Therefore, there exists a \(t\) such that \((2^{t-1} < S \leq 2^t)\). Let \(k \in \mathbb{N}\) and \(1 \leq k < n\). If \(a_k = \left\lfloor \frac{S}{2^{t-1}} \right\rfloor\) for \(t\) such that \((2^{t-1} < S \leq 2^t)\), then \(\sum_{k=1}^{n} a_k\) is minimized.

First, let us prove that \(\sum_{k=1}^{n} a_k\) is minimized when \(a_{k-1} = \left\lfloor \frac{a_k}{2} \right\rfloor\).

**Lemma 1.** For all \(k \in \mathbb{N}\), if \(a_k\) satisfies \(a_{k-1} = \left\lfloor \frac{a_k}{2} \right\rfloor\), then \(\sum_{k=1}^{n} a_k\) is minimized.

It is assumed that there is another sequence \(a'_k\) satisfying the above five conditions. To demonstrate the above lemma, we show that the sequence \(a'_k\) is always equal to or greater than the sequence \(a_k\). Therefore, the proposition to prove is as follows.

**Proposition 1.** If \(a_{k-1} = \left\lfloor \frac{a_k}{2} \right\rfloor\) for all \(k \in \mathbb{N}\), then \(a_k \leq a'_k\).

This proposition is denoted as \(P(k)\). We will prove this proposition using mathematical induction. Let us first define the relationship between \(a_{k+1}\) and \(a_k\), which will be universally used.

\[
\begin{align*}
    a_{k+1} &= a_k + b_k (\because (2)) \\
    a_k &\geq b_k (\because (4)) \\
    \therefore a_{k+1} &\leq 2a_k 
\end{align*}
\]

(8) and (9) imply that the total sum of \(a_k\) should be minimized to minimize the total sum of FIFO entries. In conclusion, the condition of \(a_k\) to minimize \(\sum_{k=1}^{n}(a_k + b_k)\) is as follows. Variable \(t\) denotes the exponent value of the power of two that is greater than \(S\) and closest to \(S\). It changes according to the maximum number of data that the sorter can handle.

\[
\begin{align*}
    \sum_{k=1}^{n}(a_k + b_k) &= \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \\
    &= \sum_{k=1}^{n} a_k + S - a_1 
\end{align*}
\]

First, let us prove that \(\sum_{k=1}^{n} a_k\) is minimized when \(a_{k-1} = \left\lfloor \frac{a_k}{2} \right\rfloor\).
\[
\frac{1}{2} a_i' \leq a_{i-1}' \quad (\because \text{(8)})
\]
\[
\iff \left[ \frac{a_i'}{2} \right] \leq a_{i-1}' \quad (\because \text{(4)})
\]
\[
\iff \left[ \frac{a_i'}{2} \right] \leq a_{i-1}' \quad (\because a_i \leq a_i')
\]
\[
\iff a_i - 1 \leq a_{i-1}' \quad (\because a_i = \left[ \frac{a_i'}{2} \right])
\]

Assume that \( k \in \mathbb{N}, 1 \leq k \leq n \) and \( a_{k-1} = \left\lfloor \frac{n}{2} \right\rfloor \). Then \( a_k \leq a_k' \), which implies that \( \Sigma_{k=1}^n a_k \leq \Sigma_{k=1}^n a_k' \). Therefore, proposition 1 is established when the number of terms in the sequence \( a_k \) and the sequence \( a_k' \) are the same. Next, let us prove that proposition 1 is established when the number of terms in the sequence \( a_k \) and the sequence \( a_k' \) are different.

Assume that the number of terms in the sequence \( a_k' \) is \( n+N \) for \( N \in \mathbb{N}, N \geq 1 \) and \( 1 \leq k \leq n \), which means \( a_{k+N} = S \). Using mathematical induction, we can prove in the same way that \( a_k \leq a_k'_{k+N} \) and \( \Sigma_{k=1}^n a_k \leq \Sigma_{k=1}^{n+N} a_k' \). Therefore, we can see that \( \Sigma_{k=1}^n a_k \) is minimized if \( a_{k-1} = \left\lfloor \frac{n}{2} \right\rfloor \).

Assume that the number of terms in the sequence \( a_k' \) is \( n+N \) for \( N \in \mathbb{N} \) and \( N \leq n \). Using the mathematical induction method, we also can prove that \( a_n \leq a_{n-N} \). Then, we have \( a_{1+N} \leq a_1' \). As \( a_1' \) is 1 based on the condition (1), \( a_{1+N} \) should be less than or equal to 1. However, \( a_1 < a_{1+N} \) because of condition (1) and (2). Therefore, there is no sequence \( a_n \) that satisfies \( a_n \leq a_{n-N} \) and whose number of terms is less than \( n \).

In conclusion, if \( k \) satisfies \( a_{k-1} = \left\lfloor \frac{n}{2} \right\rfloor \) for \( 1 \leq k \leq n \), then \( \Sigma_{k=1}^n a_k \) is the minimum. \( \square \)

Proposition 1 is established through the proof above. Furthermore, lemma 1 is established. The proof that \( a_{k-1} = \left\lfloor \frac{n}{2} \right\rfloor \) and \( a_k = \left\lfloor \frac{S}{2^{n-k}} \right\rfloor \) represent the same sequence is as follows.

**Proof.**

\[
a_n = \left\lfloor \frac{a_{n+1}}{2} \right\rfloor = \left\lfloor \frac{S}{2} \right\rfloor
\]

\[
a_{n-1} = \left\lfloor \frac{a_n}{2} \right\rfloor = \left\lfloor \frac{S}{2} \right\rfloor = \left\lfloor \frac{S}{2^2} \right\rfloor
\]

\[
\ldots
\]

\[
a_k = \left\lfloor \frac{S}{2^{n+1-k}} \right\rfloor
\]

The first term of \( a_k \) is as follows.
\[ a_1 = \left\lceil \frac{S}{2^n} \right\rceil = 1 \quad (1) \]

\[ \iff 0 < \frac{S}{2^n} \leq 1 \]

\[ \iff 0 < S \leq 2^n \quad (11) \]

Therefore, when the range of \( S \) is limited to \((2^{t-1} < S \leq 2^t)\), \( a_k \) is as follows.

\[ a_k = \left\lceil \frac{S}{2^{t+1-k}} \right\rceil \quad (12) \]

Therefore, if lemma 1 is established, theorem 1 is also established.

\[ \square \]

5 Experimental results

We evaluated the proposed technique in terms of memory resource consumption. We implemented two versions of the sorter to verify the proposed technique. The \textit{RM.CFMS} is a CFMS that implements the proposed memory minimization technique. We designed 23 different types of RM.CFMS and B.CFMS in the FPGA with different maximum numbers of data that the sorter can handle. The FIFOs that make up each sorter consist of FPGA registers. We synthesized sorters using Xilinx Vivado. The target FPGA is xcku025-fvva1156-1-I.

Table I represents the amount of FPGA resources consumed by each sorter. RM.CFMS consumes an average of 5\% fewer flip-flops than B.BFMS. In addition, the LUT-RAMs consumption of RM.CFMS is 14\% less than B.CFMS on average. As shown in the experimental results, the proposed technique reduces the amount of memory resources consumed in implementing the CFMS.

![Table I. Average memory resource requirements](image)

<table>
<thead>
<tr>
<th></th>
<th>Flip-flops</th>
<th>LUT-RAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.CFMS</td>
<td>718</td>
<td>980</td>
</tr>
<tr>
<td>RM.CFMS</td>
<td>684</td>
<td>843</td>
</tr>
</tbody>
</table>

Fig. 5(a) and Fig. 5(b) show the consumption of flip-flops and LUT-RAMs in the FPGA, respectively, as the maximum number of data (denoted as \#Value) increases that B.CFMS and RM.CFMS can handle. Let us divide each experimental sample into three ranges: \(2^6 < \#\text{Value} \leq 2^7\), \(2^7 < \#\text{Value} \leq 2^8\) and \(2^8 < \#\text{Value} \leq 2^9\). For each range, the flip-flop usages of RM.CFMS are 4.1\%, 4.4\% and 5\% less than B.CFMS on average. Furthermore, RM.CFMS consumes an average of 12\%, 11.5\% and 15\% less LUT-RAMs than B.CFMS in each range. Let \( n \) be a natural number. As shown in the two graphs, B.CFMS consumes the same amount of memory resources in the interval where the maximum number of the sorter can handle is more than \(2^{n-1}\) and \(2^n\) or less. This is because the sorter designed according to the baseline design must use a sorter that can handle up to \(2^n\) for sorting data more than \(2^{n-1}\) and \(2^n\) or less. On the other hand, in the case of RM.CFMS, the amount of memory resource consumption changes according to the maximum number of data that the sorter can handle. This is because RM.CFMS...
uses only the minimum amount of memory resources required for the number of
data to be sorted. Therefore, when processing data that is not a power of two,
RM_CFMS consumes fewer memory resources than B_CFMS. Note that B_CFMS
and RM_CFMS consume the same amount of memory resources when the
maximum number of data that the sorter can handle is a power of two. This is
because RM_CFMS is designed to have the same structure as B_CFMS when the
maximum number of data that it can process is a power of two.

(a) Flip-flop consumption of B_CFMS and RM_CFMS

(b) LUT-RAM consumption of B_CFMS and RM_CFMS

Fig. 5. The amount of hardware resources consumed by B_CFMS and
RM_CFMS
6 Conclusion

This paper introduces a memory minimization technique for CFMS. Previous studies have shown that CFMSs consume more memory than they actually need to sort data that is not a power of two. The proposed memory minimization technique minimizes the memory consumption of the CFMS structure according to the maximum number of data that the sorter is designed to handle based on FPGA simulations. Experimental results indicate that a memory minimization technique-based sorter consumes 5% fewer flip-flops and 14% less LUT-RAM on average than a sorter without the proposed technique.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NO. NRF-2015R1A2A1A01002716).