Unbalanced FIFO Sorting for FPGA-Based Systems*

Rui Marcelino
UAlg/ISE
Faro, Portugal
rmarcel@ualg.pt

Horácio C. Neto
UTL/IST/INESC-ID
Lisboa, Portugal
hcn@inesc-id.pt

João M. P. Cardoso
Univ. do Porto/FEUP/DEI
Porto, Portugal
jmpc@acm.org

Abstract—**Sorting is an important operation in a myriad of applications. It can contribute substantially to the overall execution time of an application. Dedicated sorting architectures can be used to accelerate applications and/or to reduce energy consumption. In this paper, we propose an efficient sorting unit aiming at accelerating the sort operation in FPGA-based embedded systems. The proposed sorting unit, named Unbalanced FIFO Merge Sorting Unit, is based on a FIFO merger implementation and is easily scalable to handle different data-set sizes. We show results of the proposed sorting unit when isolated and when integrated in a software/hardware solution. When using a Xilinx Virtex-5 SX50T FPGA device, the logic resources for a 32 K-word machine is lower than 1%, and the block RAM usage is about 22%. When compared to a quicksort pure software implementation, our Sorting Unit provides speed-ups from 1.2× to 50× and about 20× when isolated and when integrated in a software/hardware solution, respectively.**

I. Introduction

Sorting is one of the fundamental operations in computing. We can see uses of sorting in many applications: signal processing systems [1], encoding [2], etc. In some data acquisition systems, with huge quantities of data to analyze, the extraction of some features is performed using the sort operation [3]. Most times, the performance of the sorting algorithm is crucial for the overall system performance, as is the case in database management systems [4]. The importance and features of sorting also make it a desirable benchmark, e.g., for evaluating the energy efficiency of a wide range of computer systems from clusters to handhelds [5].

Our goal is to speedup sorting by coupling dedicated hardware sorting units to a general purpose processor (GPP). As target we consider FPGA-based embedded systems, being the simple view shown in Fig. 1 an example of such a system. The data to be sorted are stored in a memory connected to the system bus. This memory can be located inside or outside the FPGA (e.g., in an SDRAM device). In this paper, we improve our previous approaches to hardware sorting units [6] by presenting an unbalanced FIFO sorting unit that takes advantage of the FPGA hardware resources and is suited to embedded systems where data to be sorted are located in a single-port memory.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. Section 3 describes the basics of the sorting architecture, and Section 4 presents the hardware/software solution. Experimental results are presented in Section 5 and we conclude the paper in Section 6.

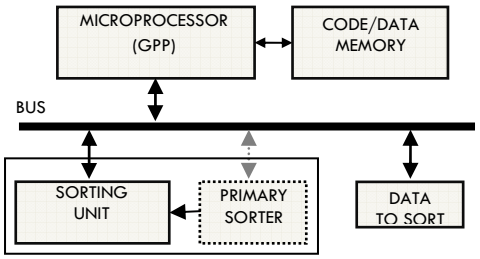

Fig. 1. Block diagram of the target system.

II. Related Work

Concerning application-specific architectures, two different approaches have been considered for accelerating sorting operations, one focusing on variations of the sorting networks [7][8][9], and the other exploring systolic linear arrays [10][11]. Although those approaches may achieve high-performance sorting operations, both rely on the simultaneous availability of data to feed the sorting unit. For most cases, this hampers their practical use with current technology or at least in systems where data to be sorted are located in the same limited port memory device.

Ratnayke and Amer [12] proposed an FPGA implementation variation of the counting sort algorithm [13]. The occurrence histogram is implemented in on-chip FPGA memories (BRAMs). This approach is efficient for relative small data-sets. Sorting large data-sets with this approach would require large memories. Note that it is with large data-sets where the execution time for sorting might be more critical.

Due to the importance of sorting operations, parallel sorting solutions have also been proposed [14][15][16]. They make use of parallel sorting algorithms, suitable for multiprocessor implementation and are efforts orthogonal to the one proposed in this paper.

III. Sorting Unit Architecture

The new FIFO-based Merge Sorting Unit uses an unbalanced FIFO merger (different FIFO lengths), as illustrated in the block diagram show in Fig. 2. The unit uses two FIFOs (A and B). The *FIFO_A* is the input FIFO where the data are written in the sorting unit and has length *L*. The *FIFO_B* has length *M* and is responsible to accumulate sorted data. The sorting unit is able to sort *M* values, which is the size considered for the sorting unit.

This sorting unit requires that the data written to *FIFO_A* have been previously sorted. This primary computation (re-

* This work was partially supported by FCT, the Portuguese Science Foundation, under the research grant PTDC/EEA-ELC/70272/2006.

ferred herein as *primary sorter*) can be done by the host microprocessor (e.g., using quicksort) or by a second hardware sorting unit coupled to the FIFO-based Merge Sorting Unit. These two options are illustrated and evaluated in this paper.

The first time the merge operation is performed (first iteration of the sorting unit) data are simply moved from *FIFO_A* to *FIFO_B*. On the other iterations, each step sorts and merges the data from the two FIFOs into *FIFO_B*. In each step, the outputs of the two FIFOs are compared and one of the outputs is guided to *FIFO_B* using the multiplexer. This process repeats until the *FIFO_B* is full or the *FIFO_A* is empty.

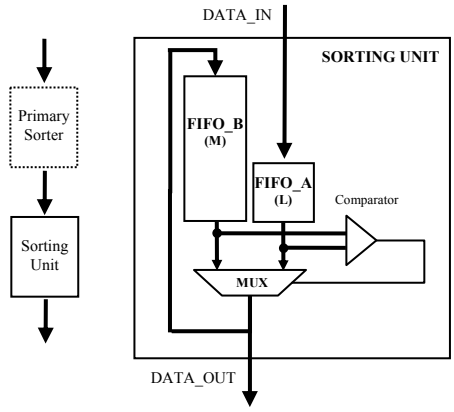

Fig. 2. Block diagram of the Unbalanced FIFO-based Merge Sorting Unit.

Given the size of *FIFO_A*, *L*, and the size of *FIFO_B*, *M*, where,

$$M = k \times L, \quad k \; is \; integer \geq 1 \qquad (1)$$

the execution time required, by the unit to sort *M* elements, considering k L-size chunks of data previously sorted, is proportional to:

$$T_{FIFO}(M,L) \propto L \sum_{j=1}^{M/L} j = \frac{M}{2L}\left(\frac{M}{L}+1\right) \qquad (2)$$

When $L=M$ we have a time complexity of O(m), being m the number of elements to sort. When L decreases compared to M, the time complexity asymptotically tends to O(m^2). The space complexity always stands at O(m). Fig. 3 presents the execution time trend required to sort using the FIFO-based Merge Sorting Unit for different ratios of L/M, according to equation (2). It shows the performance degradation that occurs when reducing the size of the *FIFO_A* with respect to a given size of *FIFO_B* (i.e., when increasing the value of k in (1)). The curve shape illustrates that the latency of the sorting unit is very sensible to the value of k. At a first sight, one may conclude that the best sorting unit would be the balanced one (i.e., L≡M). However, this might imply large data chunks to be sorted by the *primary sorter* and always requires hardware resources for two FIFOs of the same size, which, with respect to the unbalanced solution, achieves smaller sorting units given the FPGA resources available.

The sorting unit previously described needs a *primary sorter* for each set of L data elements written to the unit. We can represent the total sorting time of the Unbalanced FIFO-based Sorting Unit (UFSU) as:

$$T_{UFSU}(M,L) = T_{com.}(M) + T_{FIFO}(M,L) + \left(\frac{M}{L}\right) T_{Primary}(L) \qquad (3)$$

$T_{Com.}(M)$ represents the total time to communicate M elements between memory and the sorting unit. This value remains constant for different L values. $T_{FIFO}(M,L)$ represents the time spent by the sorting unit to sort M elements, being L the size of the input *FIFO_A*. Finally, $T_{Primary}(L)$ represents the time needed by the *primary sorter* to sort L data elements before writing them into the sorting unit. As referred, this *primary sorter* can be done in software or using a dedicated hardware engine.

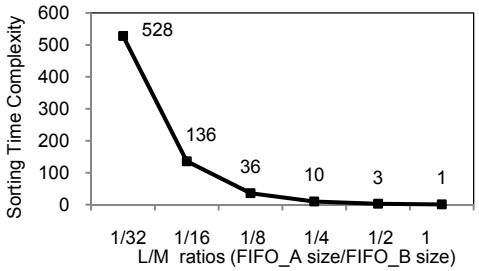

Fig. 3. Latencies of the FIFO sorting unit for different sizes of the *FIFO_A*, *L* related to *M*.

IV. Hardware-Software Solution

When the size N of data to be sorted is larger than the size of the sorting unit, M, a sorting algorithm based on the splitting of data in chunks and on sorting operations on those chunks must be used.

Our current implementation uses a hardware/software solution. For the top-level sorting we use the odd-even merge algorithm proposed by Batcher [7][17], where the comparator-swap of M elements is done by our hardware sorting unit. The data are split into blocks of M or less elements. In the first pass, N/M merges are performed through the data. The sorted blocks of data are merged in a sequential way, doing several passes through the data until the whole data elements become sorted. Note that the *primary sorter* operation required by our sorting unit is only needed in this first pass of this hardware/software solution.

The total number of merges required is given by:

$$Merges(N,M) = \frac{N}{2M} log\left(\frac{2N}{M}\right) log\left(\frac{N}{M}\right) + \frac{2N}{M} - 1 \qquad (4)$$

Since this approach uses a sequential sorting system, all the components in the chain have an impact on the global system performance. We have also to consider the software overhead due to the Odd-Even merger implementation, $SW_{Overhead}$ (proportional to *Merges(N,M)*). The global sorting latency can be represented by:

$$T(N,M,k) = \left(\frac{N}{M} + 2 \cdot Merges(N,M)\right) \cdot T_{com.}(M) + \left(\frac{N}{M}\right) \cdot k \cdot T_{Primary}(L \equiv M/k) + Merges(N,M) \cdot T_{FIFO}(M, L \equiv M/k) + SW_{Overhead} \qquad (5)$$

V. Experimental Results

The sorting hardware units have been specified in parameterized behavioral RTL-VHDL code. A Xilinx Virtex 5 SXT FPGA (XC5VSX50T-1ff1136) has been used to implement the overall system. The sorting times presented in this paper consider data sets of 32-bit integers, randomly generated using uniform distribution.

The Embedded Development Kit (EDK) and ISE, release 10.1iSP3, from Xilinx, were used for system development, RTL synthesis and placement and routing, and device configuration. For prototyping and test, we use a system with a Xilinx 32-bit MicroBlaze *softcore* processor, as GPP. The MicroBlaze was configured with default parameters, i.e., without the optional datapath units, and without caches. The software implementations were compiled using the gcc compiler included in the EDK (mb-gcc), with the –O2 option.

For the particular system presented herein, the sorting unit is instantiated as a Processor Local Bus (PLB) custom core and the data to be sorted are stored in BRAMs connected to the PBL bus. Data transfer between the BRAMs and the sorting units is done through the GPP. We use as reference a pure software sorting implementation of the quicksort algorithm with data and instructions in on-chip memories thus resulting in a best-case scenario.

In these experiments, both the sorting unit and the *softcore* processor were running at the same clock frequency (100 MHz for the validations done using the FPGA board). The sorting unit implemented in the board uses for *FIFO_A: (a)* 128 words (*L*), when using a hardware *primary sorter*, and (b) 128 and 1 k words (*L*), when using a software *primary sorter*. All implementations use 32 k words (*M*) for FIFO_B.

The FIFOs were implemented using the Xilinx CORE Generator [18]. The sorting unit merges one data element in each two clock cycles (one cycle for reading and the other cycle for writing to FIFO). This implementation does not use pipelining.

TABLE I summarizes the FPGA resources used for the sorting unit. As can be seen, the logic resources are less than 1% of the FPGA resources, and for a 32 k×32-bit sorting unit only 22% of the BRAMs are used. The maximum clock frequency reported by the Xilinx ISE tool is 166 MHz, which exceeds the current maximum PLB frequency that can be achieved (150 MHz in the Xilinx Virtex 5 devices).

TABLE I. Hardware results for the FIFO-based Merge Sorting Unit, able to sort 32 k×32-bit elements, considering a Xilinx Virtex 5 (XC5VSX50T-1) FPGA.

Number of Slices Registers	559 of 32,640	1%
Number of Slices LUTs	487 of 32,640	1%
Number of RAMB36	30 of 132	22%
Maximum Frequency	166 MHz	

For the hardware *primary sorter* we analyze Insertion Sorting Units [6] with two sizes: 128× and 1k× 32-bit. TABLE II shows the resources used and maximum operating clock frequencies for these two units considering two Virtex 5 FPGAs. This unit requires around 33 registers and 112 LUTs per 32-bit word. Based on these characteristics, large insertion sorting units will be difficult to justify.

A comparison between the performance of the Unbalanced FPGA-based Merge Sorting Unit (UFSU), with *primary sorter* in software and hardware, with a best-case pure software quicksort (using on-chip memories for data and instructions) is presented in Fig 4. As expected, the speedups are higher when using a hardware unit for *primary sorter*. Specifically, with an insertion sorting unit as *primary sorter*, the primary sorting time is completely overlapped by the communication time.

TABLE II . Hardware results for the primary Insertion Sorting Unit, considering Xilinx Virtex 5 FPGAs.

FPGA device	XC5VSX50T-1		XC5VLX330T-1	
Size of Sorting Unit (#elements)	128 ×32-bit		1 k ×32-bit	
Number of Slices Registers	4,223	12%	33,791	16%
Number of Slices LUTs	16,420	50%	114,708	55%
Maximum Frequency	265 MHz		271 MHz	

Due to the overhead of the quicksort in the *primary sorter*, the best performance is achieved when using the lowest value L=128 (Fig 4(a)). With the increase of M the speedup of the sorting unit over quicksort pure software solution increases until a certain value of M (which depends on the L value). After that point, it starts decreasing (see column 32 k in Fig 4(a) for L=128). This happens when the heavy impact of the sorting unit based on the quadratic behavior given by equation (2) starts to be noticed. With the *primary sorter* using the hardware insertion sorting unit (Fig 4(b)), the performance increases with the increase of L. However, the performance always decreases with the increase of the M/L ratio, as intuitively expected.

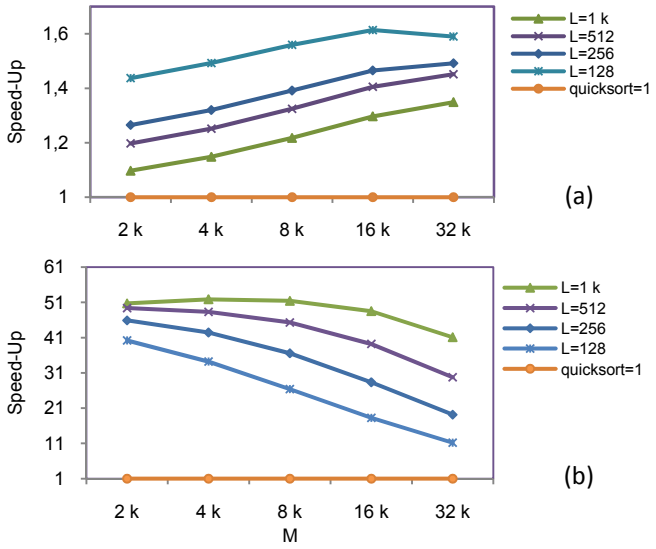

Fig 4. Speed-up between the sorting unit (UFSU) and software quicksort when sorting a set of 32 k 32-bit words. Two options are considered for different sizes of *L*: (a) Software *primary sorter*; (b) Hardware *primary sorter*. Values for *L* greater than 128 have been estimated.

Fig. 5 presents the results when using data sets larger than the size of the sorting unit. In this case, we use the hardware/software implementation referred in this paper, which is based on a software Odd-Even Merger algorithm. The data to be sorted is placed in external memory DDR2 present in the

Virtex-5 board used. The data sets used range from 64k-words to 16M-words.

A FIFO-based Merge Sorting Unit with size (M) of 32 k words is used for the sorting unit. We consider L sizes of 128 and 1k words. The results presented in Fig. 5 include: (i) two types of software/hardware solutions using a software or a hardware *primary sorter*, respectively; (ii) software quicksort; (iii) software odd-even sorting algorithm using quicksort to sort chunks of 32 k words.

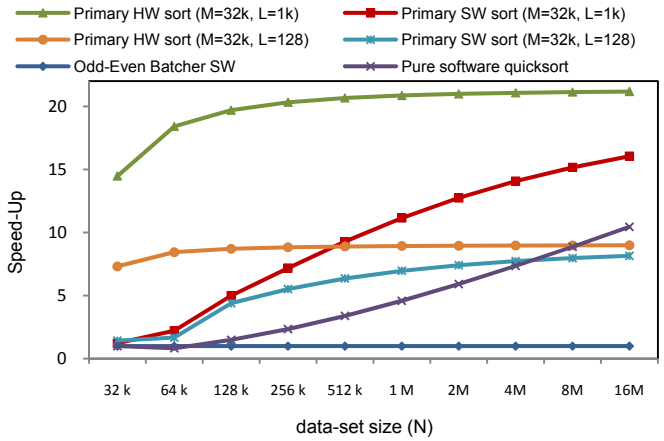

Fig. 5. Evaluation of the hardware/software sorting system for 32-bit integer data-sets with sizes from 64 k to 1 *M*. Comparison between the sorting unit, with software and hardware *primary sorters* versus software quicksort.

We obtain a high speed-up even when using the software *primary sorter*. Note that in the software/hardware solution, the *primary sorter* is only needed in the first pass through the overall data. In this case, and as a consequence of the software algorithm used to merge chunks of data sorted by the hardware sorting unit, the speed-ups increase with the increase of L. It is interesting to see that a solution with a *primary sorter* using quicksort (as is the case of the one using L=1k) may surpass, for values of N greater than a certain value, the performance of solutions using an insertion sorting unit for *primary sorter* (as the one using L=128). For sorting units with a given size (M,L), the quicksort and the insertion sorting unit versions for the *primary sorter* tend asymptotically, when N increases, to the same execution time, as is clear in Fig. 5 for L=128. This was expected because the quicksort overhead to sort the chunks of data tends to be unnoticeable for large values of N.

VI. Conclusions

This paper presented a novel software/hardware approach to accelerate sorting. We specifically address embedded systems with sequential access to the main storage where data elements to be sorted are placed. The overall system consists of two sorting units and a software sorting algorithm that uses the hardware sorting units to sort data chunks. The core of the approach uses an unbalanced FIFO-based merge sorting unit with two FIFOs. The modularity of the architecture presented allows it to be easily adapted for different data types and sizes.

The system has been implemented in a state-of-art FPGA and was evaluated by coupling the sorting units to the peripheral on-chip bus in an embedded system based on a *softcore* microprocessor (Xilinx MicroBlaze was used). The proposed sorting unit requires an acceptable number of logic resources and memory elements and provides speed-ups of 20× over a quicksort pure software solution running in the *softcore* microprocessor of the system. The results presented demonstrated that the proposed sorting unit is suitable for speeding-up the sorting of large data-sets.

Future work includes studies about a pipelined FIFO-based merge sorting unit, and a sorting architecture using parallel hardware sorting units.

References

[1] Sekanina Lukáš, "Evolutionary Design Space Exploration for Median Circuits," In: *Lecture Notes in Computer Science, Vol. 2004*, No. 3005, DE, pp. 240-249, 2004.

[2] E. Jamro, M. Wielgosz, and K. Wiatr, "FPGA Implementation of the Dynamic Huffman Encoder," In *Proc. Workshop of Programmable Devices and Embedded Systems,* pp. 60-65, February 2006.

[3] C. C. W. Robson and C. Bohm, "A high speed data acquisition collector for merging and sorting data," *Nuclear Science Symposium Conference Record, 2008*. NSS '08. IEEE, 2008.

[4] G. Goetz, "Implementing sorting in database systems," *ACM Comput. Surv.*, vol. 38, Issue 3, paper 10, 2006.

[5] R. Suzanne, A. S. Mehul, R. Parthasarathy, and K. Christos, "JouleSort: a balanced energy-efficiency benchmark," in *ACM SIGMOD Int'l Conf. on Management of Data*. Beijing, China, 2007.

[6] R. Marcelino, H. Neto, and J. M. P. Cardoso, "Sorting Units for FPGA-Based Embedded Systems," *IFIP Int'l Federation for Information Processing, Vol. 271*; Distributed Embedded Systems: Design, Middleware and Resources; Bernd Kleinjohann, Lisa Kleinjohann, Wayne Wolf. , eds., pp. 11–22., 2008.

[7] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the spring joint computer conference,* April 30-May 2, 1968. Atlantic City, New Jersey: ACM, 1968.

[8] J. Martinez, R. Cumplido, and C. Feregrino, "An FPGA-based parallel sorting architecture for the Burrows Wheeler transform," *Int'l Conf. on Reconfigurable Computing and FPGAs (ReConFig'05)*, 2005.

[9] Y. Zhang and S. Q. Zheng, "An Efficient Parallel VLSI Sorting Architecture," *VLSI Design*, vol. 11, pp. 137-147, 2000.

[10] L. Chi-Sheng and L. Bin-Da, "Design of a pipelined and expandable sorting architecture with simple control scheme," *IEEE Int'l Symposium on Circuits and Systems (ISCAS'02)*, 2002.

[11] B. Parhami and K. Ding-Ming, "Data-driven control scheme for linear arrays: application to a stable insertion sorter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 23-28, 1999.

[12] K. Ratnayake and A. Amer, "An FPGA Architecture of Stable-Sorting on a Large Data Volume: Application to Video Signals," *41st Annual Conf. on Information Sciences and Systems (CISS '07)*, 2007.

[13] H. H. Seward, "Information sorting in the application of electronic digital computers to business operation," MS thesis, Massachusetts Institute of Technology (MIT), 1954.

[14] M. Edahiro, "Parallelizing fundamental algorithms such as sorting on multi-core processors for EDA acceleration," *Asia and South Pacific Design Automation Conference (ASP-DAC'2009)*, 2009.

[15] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani, "AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors," *16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT '07)*, 2007.

[16] E. Herruzo, G. Ruiz, J. I. Benavides, and O. Plata, "A New Parallel Sorting Algorithm based on Odd-Even Mergesort," *15th EUROMICRO Int'l Conf. on Parallel, Distributed and Network-Based Processing (PDP '07)*, 2007.

[17] S. Robert, *Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, and Searching*: Addison-Wesley Longman Publishing Co., Inc., 1997.

[18] Xilinx Inc. "Virtex-5 FPGA User Guide. UG190 (v4.5)," Jan. 9, 2009.