Processing Sorted Subsets in a Multi-level Reconfigurable Computing System

Artjom Rjabov[1], Valery Sklyarov[2], Iouliia Skliarova[2], Alexander Sudnitson[1]
[1]*Department of Computer Engineering, Tallinn University of Technology, Tallinn, Estonia*
[2]*Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, Aveiro, Portugal*
artjom.rjabov@gmail.com

***Abstract*—The paper suggests a technique for extracting and filtering sorted subsets in a three-level computing system with such sub-systems as general-purpose computer (level 1), ARM Cortex-A9 (level 2), and reconfigurable logic (level 3). The last two levels are implemented in Zynq-7000 device available on the prototyping board ZC706. Communications between the levels 1 and 2-3 are organized through PCI express bus and interactions between components of levels 2 and 3 - through on-chip AXI interfaces. We studied two levels of software programs (running in PC and ARM), high-performance hardware accelerators implemented in Zynq-7000 programmable logic, and architecture enabling interactions and exchange of data between different levels. The selected for analysis sorting problem has high computational complexity and is widely required in data processing (data mining and statistical data manipulation, in particular). The results of experiments demonstrate that the elaborated architecture is efficient and permits fast solutions to be found. Proposals for potential further improvements are also given.**

***Index Terms*—Computing sorted subsets, communicating software/hardware systems, sorting networks, filtering, programmable systems-on-chip.**

I. INTRODUCTION

Many practical applications require acquisition, analysis, and filtering of large data sets. Let us consider some examples. In [1] a data mining problem is explained with analogy to a shopping card. A basket is the set of items purchased at one time. A frequent item is an item that often occurs in a database. A frequent set of items often occur together in the same basket. A researcher can request a particular support value and find the items which appear together in a basket either a maximum or a minimum number of times within the database [1]. Similar problems appear to determine frequent queries at the Internet, customer transactions, credit card purchases, etc. requiring processing very large volumes of data in the span of a day [1]. Fast extraction of the most frequent or the less frequent items from large sets permits data mining algorithms to be accelerated and may be used in many known methods from this scope, *e.g.* [2]–[4]. Another example can be taken from the area of control. Applying the technique [5] in real-time applications requires knowledge acquisition from the controlled systems. For example, signals from sensors may be filtered and analysed to prevent error conditions [5]. To provide more exact and reliable conclusion, combination of different values need to be extracted, ordered, and analysed. Similar tasks appear in monitoring thermal radiation from volcanic products [6], filtering and integrating information from a variety of different sources in medical applications [7] and so on.

Since many systems have hard real-time constraints, performance is important and hardware accelerators may provide significant assistance for software products (such as [5]). Similar problems appear in so-called straight selection sorting (in such applications where we need to find the task with the shortest deadline in scheduling algorithms [8]).

The paper suggests a new method to design high-performance accelerators based on all programmable systems-on-chip (APSoC) from the Xilinx Zynq-7000 family [9] communicating with a general-purpose computer through PCI express bus. APSoCs are recently developed field-configurable devices integrating the most advanced programmable logic (PL) and a widely used processing system (PS): the dual-core ARM® Cortex™ MPCore™. The available interfaces between the PS and PL are supported by ready-to-use intellectual property (IP) cores. These, combined with numerous architectural and technological advances, have enabled APSoCs to open a new era in the development of highly optimized computational systems [10].

The remainder of the paper is organized in four sections. Section II describes the problem and suggests an architecture of a 3-level system. Section III considers different modes of functionality of hardware accelerators. Section IV reports the results of experiments and compares them with alternative computations in general-purpose software. The conclusion is given in Section V.

II. PROBLEM DEFINITION AND SYSTEM ARCHITECTURE

Let A be a set of data items that can be of any predefined type common for general-purpose languages (*e.g.* integer). We consider here such computations that permit:

Manuscript received November 22, 2014; accepted January 16, 2015.
This research was supported by the European Union through the European Regional Development Fund, the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research, the Estonian Science Foundation Grant No. 9251, and Portuguese National Funds through FCT - Foundation for Science and Technology, in the context of the project PEst-OE/EEI/UI0127/2014.

– Extract subsets of A containing L_{max} (L_{min}) items with the maximum (the minimum) values;

– Extract subsets containing filtered values of A that fall within the given upper (u) and lower (l) bounds.

The set A can be very large and we would like to execute the computations indicated above as fast as possible.

The proposed system architecture combines the following three levels (Fig. 1):

1. Software of a host computer (such as PC) developed in a general-purpose programming language (*e.g.* C/C++ or Java). Since such software has a number of known constraints (such as the maximum number of parallel threads, and architecture-specific limitations) we would like to develop a more flexible and parallel acceleration system taking advantages of field-programmable technology.
2. APSoC PL enabling broad parallelism to be provided and eliminating architectural constraints (i.e. the most appropriate accelerator architecture can be proposed and realized).
3. APSoC software that permits interactions between different levels to be simplified and optimized with the aid of available efficient IP cores.

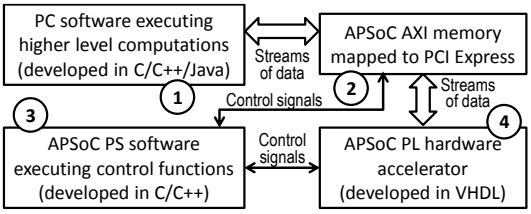

Fig. 1. Elaborated architecture.

In the proposed designs software (in the host PC and in the PS of APSoC) is running under Linux operating system. The following functionality (Fig. 1) is provided:

– As soon as some acceleration is needed, the program (in the host PC, see block 1) copies data from the set A through PCI express bus to DDR memory (see block 2) communicating with the APSoC (see blocks 3, 4) and controlled by the APSoC (ZC706 prototyping system [11] of Xilinx will be used in further experiments).

– As soon as data are transferred to the DDR, an interrupt (see block 1) is generated and handled in the APSoC PS (block 3). C/C++ function, that handles the interrupt in the PS, requests the acceleration operation in the PL and supplies necessary data (such as the number and the size of items in the received set A: see blocks 3 and 4) through AXI (Advanced eXtensible Interface [9]) GP (general-purpose [9]) port. Basic functionality of the function that handles interrupts is similar to [12].

– The PL accelerator (see block 4) executes highly parallel operations over the set A and copies the extracted subset to the same DDR memory.

– As soon as all items that form the result are transferred to the DDR, the PL generates an interrupt to the PS (see blocks 3 and 4) which is handled in the PS software.

– Interrupt handler in software of the PS sets a special flag indicating that the requested acceleration operation has been completed (see blocks 1 and 3). The flag is tested in the PC software and as soon as it is set, the resulting data are copied to the PC (see blocks 1 and 2).

Configuration of the APSoC, specifying the requested acceleration operation such as finding the minimum/maximum subsets or filtering using bounds (and consequently enabling the required operation to be chosen), is done before the execution time. It is also possible to choose operations during run-time providing necessary details from the PC to the PS and further to the PL. Data exchange between different sub-systems (PC, DDR memory, PS and PL) is initiated as follows (Fig. 2):

1. PC/PS (memory): a) software of the host PC executes C library function `memcpy` which copies data from the set A (kept in the host PC memory) to the DDR memory through the following blocks: Xilinx IP core for working with PCI express [13] (see the block AXI memory mapped to PCI express), AXI interconnect and PS memory controller (Fig. 2); b) software of the host PC generates an interrupt (through additional `memcpy` function) indicating completion of data transfer and handled in the PS (see PCI Control Unit and interrupt IRQ in Fig. 2).
2. Memory, PS/PL: a) software of the APSoC PS transfers control signals to the PL through an AXI GP master port using `Xil_Out32` function of Xilinx [14] (see GP Control Unit in Fig. 2); b) software of the APSoC PS sends a request to the PL (once again through an AXI GP master port) to execute the chosen operation.
3. The PL carries out the indicated operation getting blocks of data from the DDR memory and transferring the results to the DDR memory through AXI high-performance (HP) ports (see HP Control Unit and interrupt IRQ in Fig. 2).
4. When the results are ready and copied by the PL to the DDR, the PL generates an interrupt handled by the PS.
5. Interrupt handler in the PS sets the flag for the host PC (see PCI Control Unit in Fig. 2).
6. The PC transfers the resulting subset using `memcpy` function and the Xilinx IP core for working with PCI express.

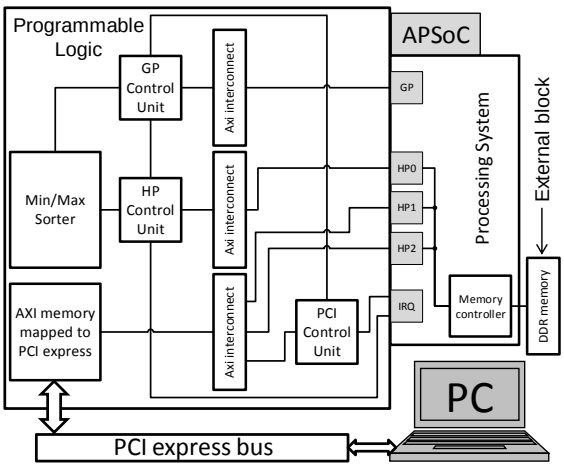

Fig. 2. Interactions between different system components.

III. FUNCTIONALITY OF THE HARDWARE ACCELERATOR

Let N be the number of elements in the given set A. We consider such tasks for which $L_{max} << N$ and $L_{min} << N$ which are more common for practical applications. Accelerating circuits implement partial sort that is done in highly parallel networks [15]. Since N may be large, it

cannot be processed completely in hardware due to the lack of sufficient resources.

We suggest solving the problem iteratively using hardware architecture shown in Fig. 3. Data are incrementally received in blocks containing up to K items and then processed by the sorting circuits [15] which iteratively execute many parallel operations and can be applied for significantly larger number of data items within the same hardware than other known sorting networks. The proposed method enables sorted subsets to be incrementally constructed as follows:

1. At initialization step (preceding the execution step) the maximum and the minimum subsets are filled in with the minimum and the maximum values as it is shown in Fig. 3.
2. Blocks with data items are sequentially supplied to the inputs of the sorting circuits located between the circuits which compute the maximum and the minimum subsets. All data items from one block are supplied in parallel. A new block arrives only when all data items from the previous block are processed (i.e. items, which satisfy criteria of the bottom block go down and items, which satisfy criteria of the upper block go up as it is shown in Fig. 3).
3. As soon as all the blocks (in which the set A is decomposed) are processed, the maximum and the minimum subsets are ready and they are transferred to the DDR memory.

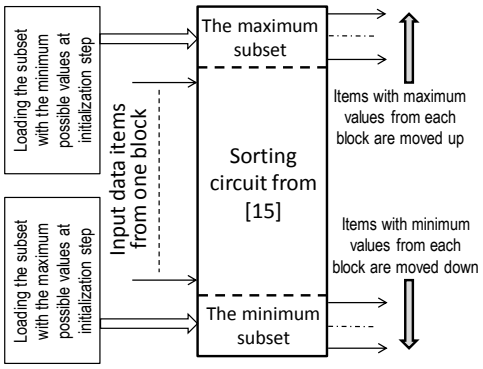

Fig. 3. Basic structure of the hardware accelerator.

It is easy to show that the circuit in Fig. 3 permits very large sets A to be processed. The sizes of the minimum/maximum subsets and the size of the blocks may vary (the details are given in the next section).

If data items need to be filtered then the circuit shown in Fig. 4 is invoked. Now we would like to use the circuit in Fig. 3 only for such data items that are within the bounds l and u. The circuit in Fig. 4 enables data items to be filtered in real-time (i.e. during data exchange between the PL and the DDR memory). The block "*l* and/or *u*" admits only those data items from AXI HP port that fall within the given bounds l/u. If and only if the item I_k is admitted, the address counter is incremented and the write enable (WE) signal is asserted allowing the value I_k to be written to the input register with the number chosen by the address counter. Data items from the input registers are inputs of the circuit shown in Fig. 3.

The filtered values may be: a) sent back to the DDR memory; b) sorted using the projects from [12]; c) used to extract the minimum/maximum subsets (Fig. 3).

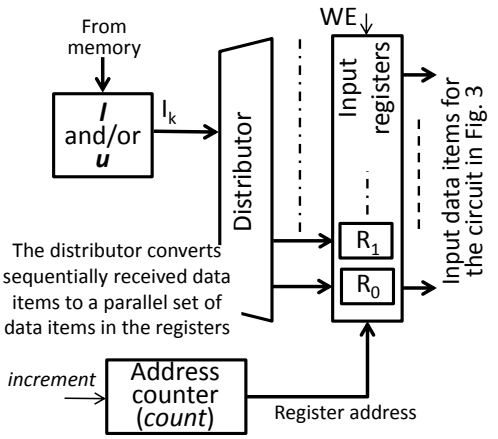

Fig. 4. Filtering circuit.

Note that additional circuits (such as problem-targeted control finite state machines) are needed for executing operations in Fig. 3 and Fig. 4. They are implemented similarly to [12], [15], [16].

IV. EXPERIMENTS AND COMPARISONS

Experiments were done with the Xilinx ZC706 prototyping system [11] containing the Zynq-7000 XC7Z045 APSoC device with PCI express endpoint connectivity "Gen1 4-lane (x4)". The PS is the dual-core ARM Cortex-A9 and the PL is Kintex-7 FPGA from the Xilinx 7th series. In all the experiments data from the set A are generated in the host PC randomly and the results are analyzed and verified also in the host PC. The size of data was chosen to be 256 KB. In the experiments the host PC generates 32-bit integer values and 64K (65,536) of 32-bit words are processed. The values of L_{min}/L_{max} varied from 128 to 1024 bytes (i.e. from 32 to 256 32-bit words).

A similar task was also solved in software only of the host PC where data from the set A were preliminary sorted and then the maximum and minimum subsets for different values of L_{min}/L_{max} were extracted. The bound values (L_{min}/L_{max}) in the host PC almost do not influence the results because the time is mainly consumed by the sort function. The following simple Java code was used:

```
long time=System.nanoTime();
   Arrays.sort(A);
long time_end=System.nanoTime();
```

where A is an array representing the set A. The array is generated randomly as:

```
for(int i = 0; i < A.length; i++)
      A[i] = rand.nextInt(Integer.MAX_VALUE);
```

The time is measured as (**double**) (time_end - time)/1000000. μs. Sorting 64 K of 32-bit data in PC with i7-4770 CPU 3.4 GHz and 8 GB of RAM requires approximately 18,000 μs. Similar results were also obtained for C/C++ programs running in the same PC. Transferring 256 KB from PC to DDR memory requires approximately 1,800 μs. Table I indicates the time consumed in the APSoC for extracting subsets with different number of data items and the resulting acceleration (taking into account the indicated above communication overhead of 1,800 μs).

TABLE I. THE CONSUMED TIME BY THE DEVELOPED HARDWARE ACCELERATOR AND ACCELERATION ACHIEVED COMPARED TO GENERAL-PURPOSE SOFTWARE RUNNING IN PC WITH MULTICORE I7 PROCESSOR AND 8 GB OF RAM.

L_{min} and L_{max}	The consumed time in μs	Acceleration
128 and 128	2,908	3.8 (6.2)
256 and 256	4,041	3.1 (4.5)
384 and 384	5,090	2.6 (3.5)
512 and 512	6,201	2.2 (2.9)
640 and 640	7,284	2.0 (2.5)
768 and 768	8,348	1.8 (2.2)
896 and 896	9,477	1.6 (1.9)
1024 and 1024	10,544	1.5 (1.7)

The column "L_{min} and L_{max}" includes two values because the analysed circuit permits the maximum subset with L_{max} elements and the minimum subset with L_{min} elements to be extracted at the same time.

The second column indicates the consumed time for all necessary operations in the PS and PL of the APSoC.

The column "Acceleration" also shows (see the values in parentheses) acceleration without taking into account communication overheads (i.e. without the mentioned above value 1,800 μs). Analysis of this case permits to estimate potential acceleration when data are transferred only once and then used for different computations in the APSoC.

We also evaluated potentialities for further accelerations and came to the following conclusions.

Software in the host PC is running in a high-performance multicore processor operating at significantly higher clock frequency than APSoC. To achieve additional acceleration in generally slower reconfigurable logic: a) high-level parallelism has to be used enabling hundreds of operations needed in software programs to be executed at the same time; b) the depth of combinational circuits implemented in the PL cannot be large because deep circuits involve extensive combinational path delays. The chosen sorted network [15] is not deep and operates at significantly higher clock frequency than alternative known circuits (see experiments in [15]). Higher-level parallelism can be achieved by joining data exchange and data processing operations. Let us look at Fig. 3, Fig. 4. Data are received from AXI HP ports sequentially and the maximum size of data items from one port is 64 bit. Such data need to be unrolled with the aid of the distributor circuit shown in Fig. 4. Thus, while a current block of data is being processed in the circuit in Fig. 3, the next block can be received and unrolled.

From Table I we can see that the larger are the values L_{min}/L_{max}, the smaller is the achieved acceleration. However, for the majority of practical applications very large values of L_{min}/L_{max} are not needed. Additional experiments demonstrated that the larger are the blocks of data handled in the PL the higher is the acceleration. We found that the size of such blocks for the ZC706 system could be increased from the considered 256 32-bit words to 2,048 32-bit words. Thus, the results may be additionally improved.

There are 4 AXI HP and one AXI accelerator coherency ports in Zynq devices [9]. Using many ports in parallel can be seen as another opportunity to increase throughput of hardware accelerators in the PL.

V. CONCLUSIONS

The paper suggests novel solutions for extracting subsets with the desired properties from large data sets and evaluates capabilities of a 3-level computing system that combines general-purpose software, application-specific software and reconfigurable hardware. We elaborated architecture of such a system and evaluated different implementations of the proposed solutions making a number of experiments with the Xilinx ZC706 prototyping system which interacts with a general-purpose computer through PCI express bus. We found that the considered PC-PS-PL computing system is faster by a factor ranging from 1.5 to 3.8 comparing to general-purpose software running in i7-based multicore PC.

REFERENCES

[1] Z. K. Baker, V. K. Prasanna, "An architecture for efficient hardware data mining using reconfigurable computing systems", *Proc. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, USA, 2006, pp. 67–75. [Online]. Available: http://dx.doi.org/10.1109/FCCM.2006.22

[2] S. Sun, "Analysis and acceleration of data mining algorithms on high performance reconfigurable computing platforms", *Ph.D. Dissertation*, Iowa State University, 2011. [Online]. Available: http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1421&context=etd

[3] X. Wu, V. Kumar, J. R. Quinlan, *et al*., "Top 10 algorithms in data mining", *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0114-2

[4] M. F. M. Firdhous, "Automating legal research through data mining", *Int. Journal of Advanced Computer Science and Applications*, vol. 1, no. 6, pp. 9–16, 2010.

[5] D. Zmaranda, H. Silaghi, G. Gabor, C. Vancea, "Issues on applying knowledge-based techniques in real-time control systems", *Int. Journal of Computers, Communications and Control*, vol. 8, no. 1, pp. 166–175, 2013. [Online]. Available: http://dx.doi.org/10.15837/ijccc.2013.1.181

[6] L. Field, T. Barnie, J. Blundy, R. A. Brooker, D. Keir, E. Lewi, K. Saunders, "Integrated field, satellite and petrological observations of the November 2010 eruption of Erta Ale", *Bulletin of Volcanology*, vol. 74, no. 10, pp. 2251–2271, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00445-012-0660-7

[7] W. Zhang, K. Thurow, R. Stoll, "A knowledge-based telemonitoring platform for application in remote healthcare", *Int. Journal of Computers, Communications and Control*, vol. 9, no. 5, pp. 644–654, 2014. [Online]. Available: http://dx.doi.org/10.15837/ijccc.2014.5.661

[8] D. Verber, "Hardware implementation of an earliest deadline first task scheduling algorithm", *Informacije MIDEM*, vol. 41, no. 4, pp. 257–263, 2011.

[9] Xilinx, Inc., "Zynq-7000 All Programmable SoC Technical Reference Manual", 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[10] Xilinx, Inc., *XCell Journal*, vol. 83, second quarter, 2013.

[11] Xilinx, Inc., "ZC706 evaluation board for the Zynq-7000 XC7Z045 all programmable SoC. user guide", 2013. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf

[12] V. Sklyarov, I. Skliarova, J. Silva, A. Rjabov, A. Sudnitson, C. Cardoso, *Hardware/Software Co-Design for Programmable Systems-on-Chip*, Tallinn: TUT Press, 306 p., 2014.

[13] Xilinx, Inc., "AXI Bridge for PCI Express v2.5", 2014. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_pcie/v2_5/pg055-axi-bridge-pcie.pdf

[14] Xilinx, Inc., "OS and Libraries Document Collection", 2014. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/oslib_rm.pdf

[15] V. Sklyarov, I. Skliarova, "High-performance implementation of regular and easily scalable sorting networks on an FPGA", *Microprocessors and Microsystems*, vol. 38, no. 5, pp. 470–484, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2014.03.003

[16] V. Sklyarov, I. Skliarova, A. Rjabov, A. Sudnitson, "Fast matrix covering in all programmable systems-on-chip", *Elektronika ir Elektrotechnika*, vol. 20, no. 5, pp. 150–153, 2014. [Online]. Available: http://dx.doi.org/10.5755/j01.eee.20.5.7116