International Journal of Innovative Computing, Information and Control
Volume **15**, Number **1**, February **2019**

ICIC International ©2019 ISSN 1349-4198
pp. 321–335

HARDWARE ACCELERATORS FOR INFORMATION PROCESSING IN HIGH-PERFORMANCE COMPUTING SYSTEMS

Valery Sklyarov[1], Iouliia Skliarova[1], Irbulat Utepbergenov[2] and Ainur Akhmediyarova[2]

[1]Institute of Electronics and Informatics Engineering of Aveiro (IEETA)
Department of Electronics, Telecommunications and Informatics
University of Aveiro
Campus Universitário de Santiago, 3810-193 Aveiro, Portugal
{ skl; iouliia }@ua.pt

[2]Laboratory of Innovative and Smart Technologies
Institute of Information and Computational Technologies
Pushkina str. 125, 050010 Almaty, Kazakhstan
utepbergenov@ipic.kz; akhmediyarova@turan-edu.kz

Received April 2018; revised September 2018

Abstract. *A frequent problem in information processing is the rapid extraction and subsequent analysis of subsets of data that have to be formed from external requests supplying some filtering constraints. The subsets are composed of items that more or less (i.e., not exactly) satisfy the desired requirements and further improvement is needed to find elements that meet the constraints. The paper suggests methods and circuits that accelerate operations on the subsets and data extraction in such a way that: a) the number of items that meet the requirements exactly is determined, and b) all such items are extracted. It is proposed that the subsets are represented in the form of special tables and to process such tables. The problem is divided in two parts that are solved in software and in hardware correspondingly. The first part is the extraction of subsets from large data sets and this is done either in a general-purpose computer or in embedded to programmable system-on-chip processors. The second part, which is the main target of this paper, is the manipulation and analysis of subsets (tables) in hardware accelerators. The results of experiments clearly demonstrate the advantages of the proposed technique over existing methods in terms of both cost and throughput.*

Keywords: High-performance computing systems, Information processing, Filtering, Reconfigurable hardware, Programmable systems-on-chip

1. **Introduction.** Engineering and scientific applications have become more data intensive [1] requiring emerging high-performance interactive frameworks. An example is a *MapReduce* programming model that permits clusters of data sets to be generated and processed with the aid of parallel distributed algorithms. There are two basic procedures for this model that are *map* and *reduce*. The first procedure filters (extracts subsets) and sorts inputs. The second one executes a reduction phase assuring that all outputs of the map operation that share the same feature (key) are shown. Clustering is a division of the original set into subsets of objects that have a high degree of similarity in the same subset (cluster). Objects in different clusters have sharp distinctions [1]. Thus, the extraction of subsets of a set of data that satisfy a given set of characteristics can be seen as a common problem. For example, we might accumulate records about university students and then search for subsets of the records that have a specific set of features,

DOI: 10.24507/ijicic.15.01.321

such as falling within a specific age interval, having average grades, and a particular background. Similar problems arise with search engines such as those used on the Internet where the number of items in the initial collection and in an extracted subset may be large. When these engines are used in real-time control systems that are time-critical, the search must be as fast as possible. To increase the effectiveness of such information processing, attention is often focused on high-performance hardware accelerators [2], for which reconfigurable technology offers many advantages [3]. Examples of the criteria for the search problem are given in [4] that rationally combine flexibility of software and the speed of hardware. Many other practical cases are shown in [5] with an extensive list of the relevant references. It is underlined in [1] that for the majority of such cases using hardware accelerators running in parallel with software is very efficient.

The achievements of publications [1-5], references in [5] and the results of [4] provide the background for the presented work. Significance of data extraction is clearly shown in [1,5] and this technique is applicable to such a very known and efficient programming model as *MapReduce*, as well as to many clustering algorithms. Note that hardware accelerators are currently widely used in data intensive applications. Many examples can be found in [1]. However, parallelism is still limited. For example, the known accelerators cannot be used to extract data satisfying different criteria at the same time. Besides, they execute sequentially parallel sets of operations. Thus, although acceleration is achieved, applying higher parallelism permits additional speed-up to be attained. Thus, the main motivation of this work is an additional speed-up through the use of such type of processing that is done in fully combinational circuits allowing the highest level of parallelism to be reached. The main contribution is the proposing of completely parallel architectures based on combinational circuits with an opportunity to combine fully and partially parallel solutions in order to find a compromise between productivity and resources.

2. **Literature Review.** The importance and complexity of big data [6] force computer engineers to seek new solutions for processing them. Processing speed still continues to rise exponentially while memory bandwidth increases at a much slower rate [7]. Working with big data requires huge storage, extensive processing capability, and high-performance communications for data exchange [7]. It is explicitly stated in [6] that based on previous knowledge we need to introduce new techniques for data manipulation, analysis and mining [6]. Hardware acceleration with the use of Field-Programmable Gate Arrays (FPGAs) and Programmable Systems-on-Chip (PSoC) is very efficient and it is used for data sorting/mining [8,9], data manipulation/extraction [10], processing large graphs [11], etc. Many other comprehensive examples, together with the relevant references, are given in [7].

The majority of the applications referenced above use software at a higher level for manipulating huge volumes of data, and use hardware at a lower level when the number of data items is limited [1]. This is done because even the most advanced FPGAs are not capable of handling big data, but they are able to significantly accelerate lower-level (local) tasks with a limited number of data items that arise in processing big data. For example, some very interesting experiments for Bing search page ranking acceleration were presented by Microsoft Corporation in [12], where the target production latency and the typical average throughput in software-only mode have been normalized to 1.0; it is shown that with a single local FPGA, throughput can be increased by a factor of 2.25. The paper concludes that "with the configurable clouds design reconfigurable logic becomes a first-class resource in the datacenter and over time may even be running more computational work than the datacenter's CPUs". Thus, FPGA-based accelerators may significantly increase throughput for different types of data processing.

The feature that distinguishes this paper from similar works (such as [1]) is that a combination of multiple criteria is applied in a single search (see the previous section). This paper also proposes two new architectures for highly parallel search engines. The first is fast and has a moderate cost in hardware resources. The second provides the minimal possible latency (so it is very fast), but requires larger hardware resources. Thus, depending on requirements, either the first or the second proposal may be chosen. For each of them, two implementations are suggested: one is in hardware only, which is an FPGA-based solution, and the other is a hardware/software combination, which is a programmable system-on-chip solution that combines a high-performance multi-core processor and a programmable logic element.

Current search methods implemented in software [13,14] are based on sequential steps. First, data is extracted that satisfies the first criteria. Then the second criterion is applied to the extracted data, which reduces the number of data in the extracted set. This process is repeated until the last criterion has been applied. We suggest a method that evaluates all the criteria in parallel so it is significantly faster.

The two major innovations of this paper are

1) Proposing two variants of fast circuits that enable data for which at least one of the given constraints is not satisfied to be masked. This provides a solution for one part of the problem being considered.
2) Taking a binary vector with the masks and parallel computation of its Hamming weight. This provides a solution for another part of the problem.

These two major innovations are discussed in detail in Sections 4 and 6, correspondingly. Comparing to the referenced above known results, two distinctive features are provided: a) various criteria can be evaluated in parallel; b) a rational compromise may be found between performance and the used resources.

3. **Problem Description.** In general, the problem may be formulated as follows: a software system extracts a subset in which the requested items are somehow distributed. The subset is further processed in hardware to get the exact items that we want to find. Figure 1 explains the problem and explicitly indicates the main target of this paper. Frequently the extracted subsets are represented in form of tables. An example is given in Figure 2, where the table is composed of N rows $R_0, \ldots, R_{N-1}$ and M columns $F_0, \ldots, F_{M-1}$. Any column F_m, $m = 0, 1, \ldots, M-1$, contains data that can be constrained in some way. For example, the first column (F_0) in Figure 3 contains ages of persons, the second column (F_1) – the first letters of their family names, the third column (F_2) – their length

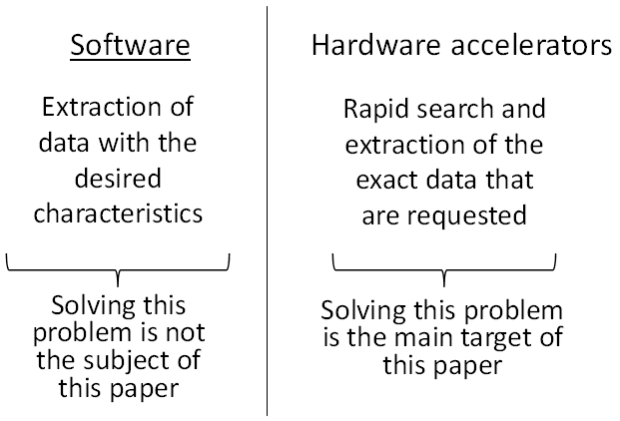

FIGURE 1. Splitting the problem between software and hardware

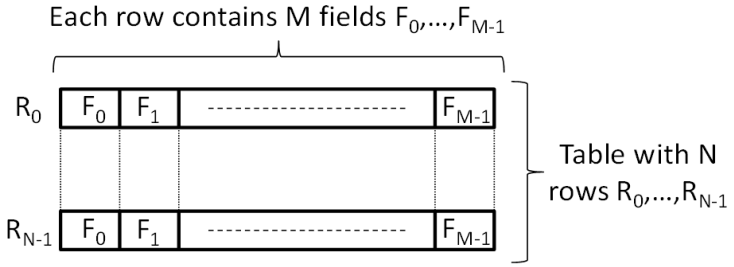

FIGURE 2. Extracted table organization

	F_0	F_1	F_2	F_3	F_4
R_0	26	m	5.8	IEETA	UA
R_1	55	l	5.1	IEETA	UP
R_2	24	s	5.9	IEETA	UL
R_3	29	p	6.2	INESC	UC
R_4	41	b	7.4	IEETA	UP
R_5	37	d	6.3	IEETA	UA
R_6	25	a	7.8	BGUIR	Mi
R_7	51	z	3.9	IEETA	Vi

FIGURE 3. An example of a table

of service in a company in years, the fourth column (F_3) – research affiliation, and the last column (F_4) – academic affiliation. The search problem that is intended to be solved is formulated as follows. For a given table we need to

a) Task 1: Extract the subset of rows that satisfy the predefined constraints; for example, values of the selected fields have to fall within given intervals. The result might be an empty subset.
b) Task 2: Determine how many rows are in the subset to be extracted, i.e., how many rows satisfy the predefined constraints.

The incremental contribution of the work can be highlighted as follows.

a) Task 1 can be solved with different levels of parallelism (see Section 4). When the highest speed is needed, the proposed solution can be fully parallel. This denotes that as soon as input data are changed the extraction of subset with the desired characteristics is done almost immediately (i.e., just with a negligible delay in combinational circuits and sequential operations are not used at all). Note that this method is more resource consuming. However, if the resources need to be minimized we can implement partially parallel solutions for which performance is decreased but it is still better than for the known methods.
b) Task 2 can similarly be solved with different levels of parallelism (see Section 6).

The table is extracted from a large data set by software running in general-purpose computer. The maximum number of rows/columns is constrained by the capabilities of the FPGAs communicating with the computer and normally does not exceed a few thousands. If it is necessary to process a larger number of data (more than a few thousand items), then different architectures can be chosen. For example, handling many tables might be done in parallel by multiple FPGAs or sequentially by a single FPGA. The results that are produced by FPGAs/FPGA are then merged in software. A particular example will be given in the next section. The primary process for Task 1 is to consider

each of the predefined constraints in turn and extract all data items that satisfy each single constraint in parallel. This permits the subset for each individual field constraint to be composed with just the delay of a simple combinational circuit, which is very fast. The number of sequential steps will be equal to the number of fields that are constrained. If required, all constrained fields could be treated in parallel, which would enable the final result to be generated in combinational circuits almost immediately. Obviously, in this case more FPGA resources are required. The primary objective of Task 2 is to obtain, as fast as possible, the number of rows in each extracted subset. This will allow the necessary memory space to be allocated in parallel in software. Note that different subsets are built for different predefined constraints.

It is shown below that the formulated search problem may be solved efficiently in a hardware accelerator, and that this is significantly faster than software programs with similar functionality. The objective of our paper is also to achieve better speedup comparing to the known hardware accelerators [1].

4. **Extracting the Constrained Data.** Figure 4 depicts the basic architecture of the proposed circuit. Multiplexers $\mathrm{MUX}_0, \ldots, \mathrm{MUX}_{N-1}$ select the fields $F_0, \ldots, F_{M-1}$ sequentially. All N rows (see Figure 2) are processed in parallel by applying the constraint to the current field. The blocks (Min, Max) are combinational and each of them has one binary output. If the value of the field falls within the indicated range between the Min and the Max then the output is '1'; otherwise the output is '0'. The remaining part of the circuit operates as follows.

1) The N-bit register Rg contains all ones initially.
2) At each step, one field (selected by the multiplexers) in all the rows is processed. The value in the register Rg is formed as an AND operation between the previous value in the register and the vector $\mathrm{O}_0, \ldots, \mathrm{O}_{N-1}$ on the outputs of the blocks (Min, Max).
3) After the last field has been processed, the vector in the register Rg is a mask for rows that have to be selected, which gives the solution for Task 1.
4) Computing the Hamming Weight (HW) in the register Rg gives the solution of the Task 2 (see Section 6), the number of selected rows.

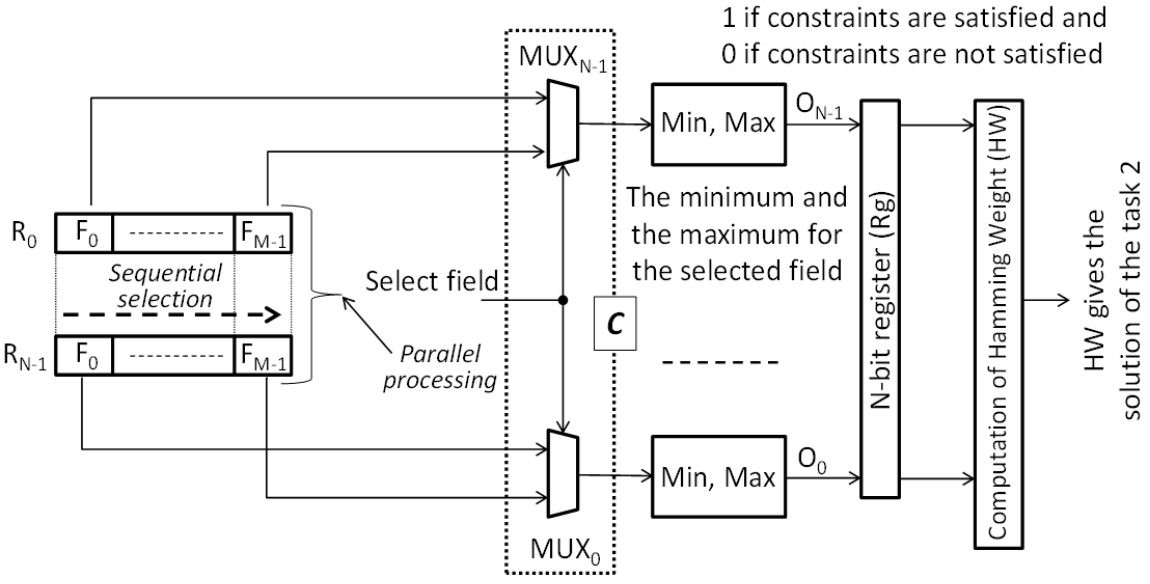

FIGURE 4. The basic architecture

Figure 5 shows an example of extracting data from the table in Figure 3 with constraints described in the lines a), ..., e) (see Figure 5). The result is ready after 5 clock cycles in which the fields $F_0, \ldots, F_4$ are selected sequentially.

The first step is shown in column a) where an AND operation is performed between the vector with 8 values '1' and the vector "10011101" on the outputs of the blocks (Min, Max). Thus, only rows with the values less than 52 and greater than 25 (constraint a) in Figure 5) are selected. The subsequent steps b), c), d), e) construct the final vector "10000100" that identifies the two rows that are shown on the right-hand side of Figure 5. The HW of the vector "10000100" is 2, which is the number of rows extracted. The blocks (Min, Max) are built using the methods [15]. The proposed circuit executes many operations in parallel and the processing time is proportional to M, where M is the number of fields (five for our example).

It should be noted that some operations are still sequential. Let us look at Figure 4. Any individual field F_m in all rows is processed in parallel. However, different fields are still treated sequentially.

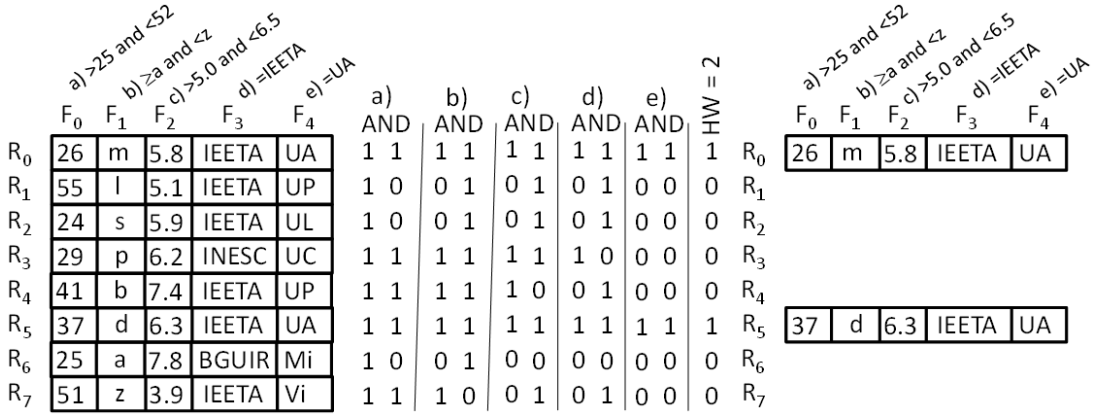

FIGURE 5. An example

Duplicating the block C (see Figure 4) with the multiplexers $\mathrm{MUX}_0, \ldots, \mathrm{MUX}_{N-1}$ for each field $F_0, \ldots, F_{M-1}$ permits a fully combinational circuit to be constructed that does not involve any sequential operations and all fields in all rows are processed in parallel. We found that although such a solution is possible, the hardware resources required will be increased significantly and in practice they are increased by more than a factor of M. In addition, all the rows have to be saved in internal registers, i.e., embedded memory blocks cannot be used. Thus the completely combinational solution can be recommended just for very fast circuits with a limited value of M (normally not exceeding 50). In the subsequent sections the main emphasis will be on the partially sequential (first) architecture.

The main difference between the circuit proposed here (Figure 4) and circuits from [13-15] is avoiding sequential steps. Indeed, either for any field, all rows of the table are handled in parallel or all fields for all rows are handled in parallel. In the last case the result is produced almost immediately after the table and the constraints are changed. The number of sequential steps can be varied from 0 (with more hardware resources) with all fields processed in parallel to M, when all fields are processed sequentially. Generally, a reasonable compromise between the resources needed and the latency can be found. Indeed, to improve performance we can handle more than one (but not all) field in parallel. The next section gives more details and shows with examples the advantages and distinctive features of the proposed architecture compared to other known approaches.

From Figure 4 we can see that solving Task 2 requires the Hamming Weight (HW) for relatively large binary vectors containing several thousand bits to be computed (see the end of Section 3). Section 6 overviews methods that can be applied for such purposes and suggests the best solutions.

5. **The Proposed vs. Known Methods.** Let us compare the proposed method with other methods that can be used for solving the same problem. Clearly, the basic structure shown in Figure 4 can be modeled (implemented) in software. However, parallel processing (see Figure 4), i.e., simultaneous execution of the chosen operation over N rows (for instance, selecting the rows with the values less than 52 and greater than 25 in the field F_0 shown in Figure 5), cannot be done in the general case because the number of parallel threads in multithreaded systems is fixed [16] and cannot be changed flexibly. Thus, only partial parallelism may be applied. On the other hand, the basic structure in Figure 4 is scalable and can be implemented and tested from hardware description language specifications (such as VHDL) in reconfigurable hardware (in FPGA). Parameterization based on language generic constructs makes it possible to implement any reasonable level of parallelism, i.e., such circuits may be used for thousands of rows (see Figure 2) treated at the same time. This is indeed "reasonable parallelism" because such number of rows is sufficient in most practical applications.

Many results of comparison for multi-core CPU (Central Processing Unit), FPGA and GPU (Graphics Processing Unit) can be found in numerous publications. We will review just some of them. A systematic approach that is based on five case study algorithms is proposed and discussed in [17]. Two orders of magnitude speedup over a CPU is observed for GPU and FPGA. An FPGA is superior to a GPU for algorithms requiring large numbers of regular memory accesses (that is our case). In the presence of data dependence, the implementation of a customized data path in an FPGA exceeds GPU performance by up to eight times (that is also our case). Additional helpful comparisons can also be found in [18].

One additional issue that should be addressed is power consumption. Note that power consumption in new ultra-scale reconfigurable microchips [19] is decreased by a factor of 2-5. The results reported in [20] show that FPGAs offer 2.7 to 293 times better energy efficiency compared to CPU and GPU.

There are also many supplementary problems that need to be solved over the extracted data, such as sorting, or some other types of data rearrangements. The majority of current hardware solutions are based on variations of Batcher's even-odd merge and bitonic merge networks [21] that are easily pipelined [22]. The best solutions for such networks are proposed in [4,23]. They allow, in particular, sorting to be executed concurrently with getting inputs through single or multiple ports. Finally, a technique allowing a reasonable compromise between the cost and the latency of the circuit has been developed. The results of experiments, implementations, and rigorous comparisons have demonstrated high efficiency and broad applicability of the proposed methods for a wide range of practical applications. It should be noted that the methods proposed in [4,23] are directly targeted to reconfigurable hardware (FPGA and programmable systems-on-chip) and they cannot be efficiently implemented in software. The technique proposed in this paper can easily be combined with the methods [4,23] making it possible for a large number of additional, very efficient solutions for the considered problems to be achieved. Some examples are listed below:

1) Extracting the maximum/minimum (sorted) subsets from the given set;
2) Extracting subsets with such values that fall within an interval bounded by the given maximum and minimum;

3) Finding the most repeated value or a set of the most repeated values;
4) Computing medians, etc.

The methods proposed in [4,23] that can be combined with the proposed architectures may be used directly, including interactions through a PCI express bus as described in detail in [10].

6. **Counting the Number of Data.** Computing the HW solves the second task. State-of-the-art circuits for such purposes have been exhaustively analyzed in [24]. Three competitive methods [24], [25,26] and [27] were discussed in terms of the cost (i.e., the number of gates) and the latency (i.e., the number of gate levels). It is argued that the cost of the circuits from [24] is the best while the latency for $N \leq 64$ is smaller for the method [27]. For $N > 64$ the method [24] is again considered to be the best. The main difference between the methods [22] and [27] is that computing the HW in [22] is done on Full Adders (FA), whereas in [27] sorting networks are used. In [28] circuits based on counting networks have been proposed and it was proved that they have similar or better characteristics than parallel counters [24]. Figure 6 illustrates why the results of [28] are better. Figure 6(a) presents an example of an HW counter from [24] for $N = 8$ that is mainly composed of FA. Any FA has three inputs (for two one-bit operands and one-bit carry in signal) and two outputs (for sum and carry out signal). Figure 6(b) depicts a counting network from [28] for $N = 8$. The data independent segments of the circuits in Figures 6(a) and 6(b) are composed of such components (e.g., FA in Figure 6(a)) that do not have any mutual input/output dependency, i.e., there is no component whose input signals depend on output signals from any other component within the segment. Hence, all necessary operations can be executed concurrently within a data independent segment and are characterized by a single one-component delay. There is a difference in the delays and complexities of the circuits in Figures 6(a) and 6(b). Delays in the circuit in Figure

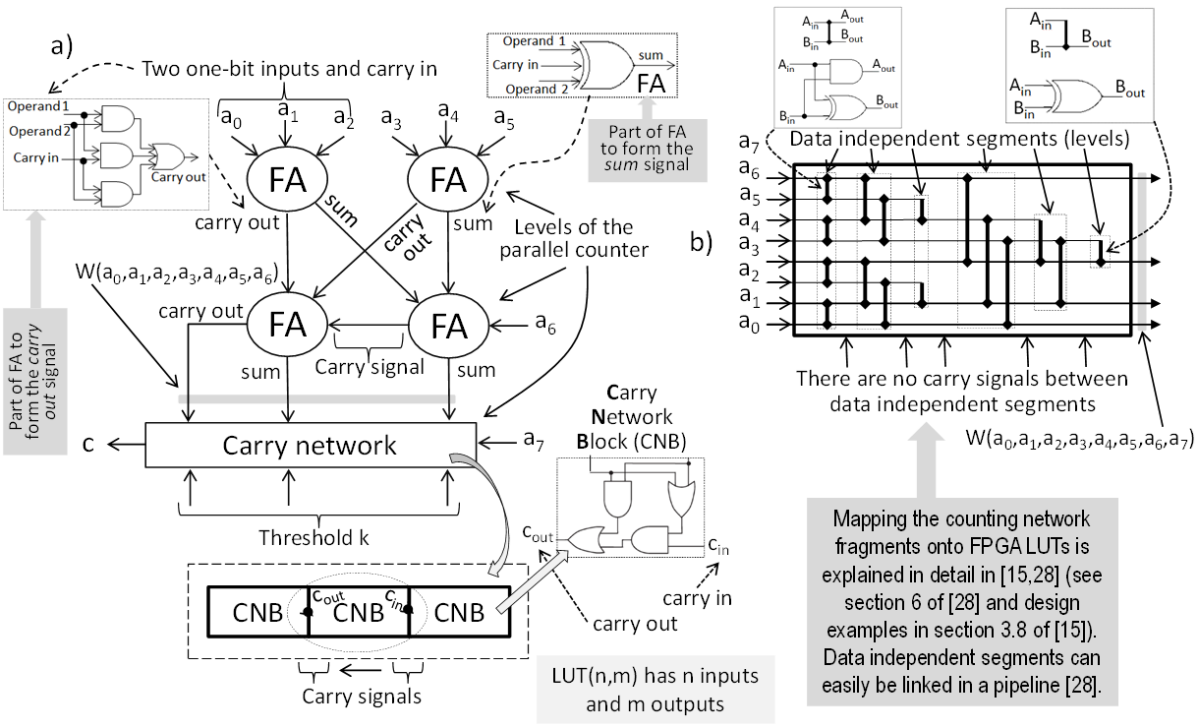

FIGURE 6. 8-input HW counter from [24] (a), and a counting network from [28] (b)

6(a) are larger because there are carry signals between FAs located at the same level. Thus, there is a data dependency inside any level. In contrast to Figure 6(a), there are no carry signals within the levels in Figure 6(b) and, therefore, all components of any level may operate in parallel.

The number of levels is not the same in Figures 6(a) and 6(b). However, the components at any level generally have smaller propagation delays in Figure 6(b) (compare, for example, any component at the top of Figure 6(b) and the circuit that forms a carry out signal in Figure 6(a)). The results strongly depend on the chosen implementation of the circuits in hardware and platform-specific features. Since recent FPGAs contain highly optimized arithmetic blocks, the circuit in Figure 6(a) may give unexpectedly good results. Thus, we decided to compare physical implementations in the most recent 7th series FPGAs of Xilinx (Artix-7 FPGA available on the Nexys-4 prototyping board [29]). We found that for $N = 8$ and $N = 16$ the results for [24,28] are almost the same and they are significantly better than for [25-27]. These results are in conformity with theoretical evaluations of complexity and performance presented below in Table 1 that collects data from [24,28,30], where N is the size of vectors, k is the value of the threshold, $P = \lceil \log_2 N \rceil$, d is a delay of one basic element (such as that is shown on the top of Figure 6(b)), γ_{FA} is the cost of FA relative to a gate, δ_{sum} is the delay of an FA, and δ_{carry} is the delay of carry propagation in the FA (in [24] $\gamma_{FA} = 9$, $\delta_{sum} = \delta_{carry} = 2$ were chosen). It should be also noted that parallel counters [24] are efficient for $N = 2^g$ (g is a positive integer). If N is not equal to 2^g then it is not quite clear from [24] how to build a non-redundant circuit.

There is another distinctive feature for counting networks [28]. In Figure 6(b) registers may easily be inserted between either data independent segments or levels allowing a pipeline to be constructed. The delay of each segment is indeed negligible and it is smaller than for possible pipelined implementations of the circuit in Figure 6(a). All necessary details can be found in [28].

TABLE 1. Cost and latency expressions for HWCs

Design from	Fixed-threshold HW counter		Two-vector HW counter	
	Cost	Latency	Cost	Latency
[25,26]	$2 \times k \times N$	$d \times (N + k - 1)$	$N \times (N + 1)$	$d \times 2 \times N$
[27]	$N \times (P)^2/2$	$d \times P \times (P + 1)/2$	$N \times P^2 + N + P$	$d \times (P + 1)$ $\times (P + 1)/2$
[24]	$(N - P - 1)$ $\times \gamma_{FA} + P$	$d \times \big((P - 1)$ $\times \big(\delta_{sum} + \delta_{carry}\big) + 1\big)$	$2 \times (N - P - 1)$ $\times \gamma_{FA} + 4 \times P$	$(P - 1) \times \big(\delta_{sum}$ $+ \delta_{carry}\big) + 1$
[28]	$\sum_{i=1}^{P} i \times (i + 1)/2$ $\times N/2^i$	$d \times P \times (P + 1)/2$	$2 \sum_{i=1}^{P} i \times (i + 1)/2$ $\times N/2^i$	$d \times (P + 1)$ $\times (P + 1)/2$

The results of Table 1 show that the solutions from [28] are better in terms of cost and latency. It is shown in [30] that embedded to FPGAs DSP (digital signal processing) blocks can efficiently be used for the proposed solutions. It is proved in numerous experiments in [31] that counting networks are the fastest if pipelined solutions are used. However, they consume relatively large resources. From the point of view of the resources, the most economical circuits are LUT-based solutions from [31]. Thus, the following circuits have been chosen:

1) pipelined counting networks [28] for such applications that require the fastest possible computations and multiple vectors (and, thus, multiple tables) to be processed in parallel;
2) LUT-based circuits from [31] for the most economical applications that in this case consume the smallest possible hardware resources;
3) DSP-based solutions [30] for FPGA/PSOC designs that efficiently use available on the respective microchip DSP slices.

7. **Hardware/Software Implementation and Experiments.** We have considered two separate stages. At the first stage a software system extracts a subset in which the necessary items are somehow distributed, i.e., a table similar to Figure 3 is built. Because there is a very large number of data items in an initial set, the extraction of a subset is done in software, which is outside of the scope of the paper. That is why we did not provide any optimization technique. Two methods were used. In the first method data within the given intervals have been randomly generated for each field (F_0, ..., F_{M-1}). In the second method we considered sets of data containing a few millions of items, such as rows shown in Figure 2. Since the first stage is outside of the scope of this paper, any method (even sequential) can be applied. The tables are saved in files that are transferred to a reconfigurable hardware. Clearly fast interfaces like PCI express described in [10] are the most efficient. Since our goal is processing data in subsets, the transfer has also been simplified. For our experiments the files we created were moved to embedded memories applying two methods that are shown in Figures 7 and 8 correspondingly.

In the first method (see Figure 7) a Xilinx Coefficient (COE) file [32] was built initially and it is used in VHDL code to specify the table data that are loaded to embedded FPGA memories. In the second method table data are directly transferred from a PC through an interface available for the Xilinx Software Development Kit. In this case a processor, namely Cortex-A9, for Xilinx all programmable systems-on-chip [33] is involved.

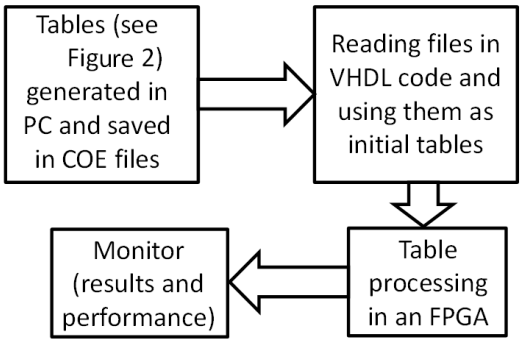

FIGURE 7. Transferring tables in COE files

If only FPGA is used then the structure shown in Figure 4 is implemented. Table rows are saved in embedded memory in such a way that all the fields with the same index can be read in parallel. Thus N fields with the same index are kept in one memory word. The size of each field is η where η is the maximum number of bits used to code data in one field from $F_0, \ldots, F_{M-1}$. In practice this value does not exceed 10 or 11 bits. So, the number of available memory blocks is sufficient, even with relatively cheap FPGAs. The structure shown in Figure 4 is directly implemented in FPGA configurable logic blocks. The complexity of circuits to count the number of data (see Section 4 above) can easily be found [28,30,31]. For example, if the circuits from [31] are chosen then for our experiments ($N = 100$, $M = 20$ and $\eta = 10$) the number of DSP slices is 9 and the maximum delay

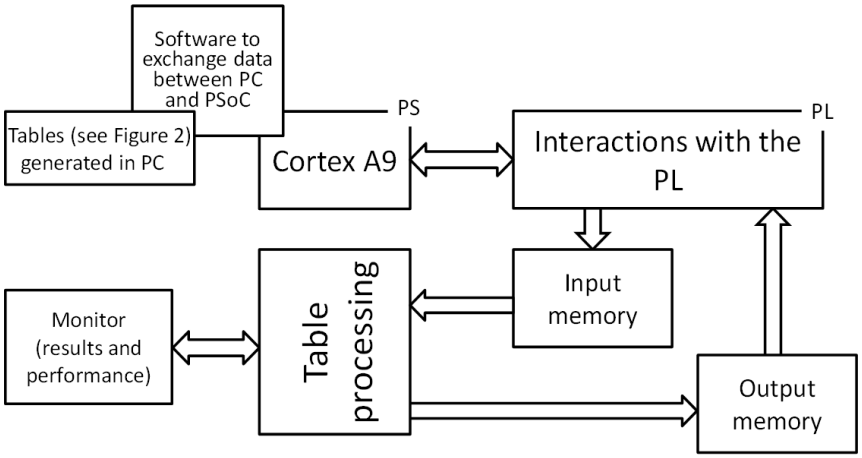

FIGURE 8. Transferring files directly from software of PC

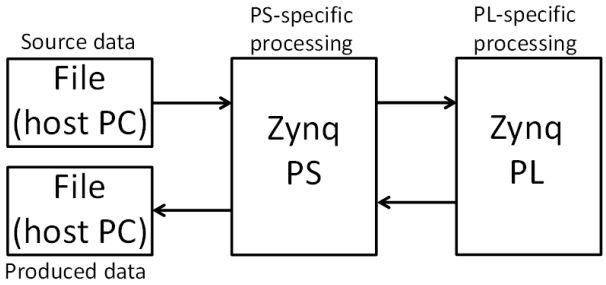

FIGURE 9. Software/hardware implementation

for Hamming weight computation is 7 ns. Counting networks from [28] that are pipelined require more hardware resources but enable the maximum delay to be reduced up to 2 ns.

Figure 9 demonstrates hardware/software solutions that were chosen for the experiments based on PSoC. The two types of implementations that have been evaluated are software only, and software/hardware. In the software only system the tables (see Figures 2, 3) are extracted and processed in software. In the second case the tables are extracted in software and processed in hardware as shown in Figure 9. Source data (i.e., a file with an extracted table) is transmitted from the host PC to hardware that is based on the Zynq PSoC from Xilinx [33]. Two different solutions were studied. The first assumes that the Processing System (PS) executes just supplementary functions, allowing a desired interface with the Programmable Logic (PL) to be established. In the second solution the PS extracts the tables and, hence, the files from the host PC contain subsets with non-extracted tables. Clearly in this case an extraction of subsets with tables (see an example in Figure 3) is slower than in a PC, but once again we would note that such extraction is not the target of this paper. The methods of data transfer and particular solutions for PSoC are described in detail in [34,35] including VHDL and C code for practical tests.

In order to prove the effectiveness of the technique we have developed we compared the proposed method with existing methods and circuits. At the first step we implemented and tested the circuit for extracting the constrained data (see Section 4 and the circuit in Figure 4 without the rightmost block that computes the HW). Both variants of the circuit (partially and fully parallel) have been analyzed. Comparison was done with software running in desktop PC and implementing similar functionality. Table 2 below shows the acceleration that has been achieved in the partially parallel circuit referenced above

TABLE 2. Acceleration achieved in the circuit (Figure 4)

	$N = M = 10$	$N = M = 50$	$N = M = 100$	$N = M = 1000$	$N = M = 2000$
Acceleration	1	6	18	117	173

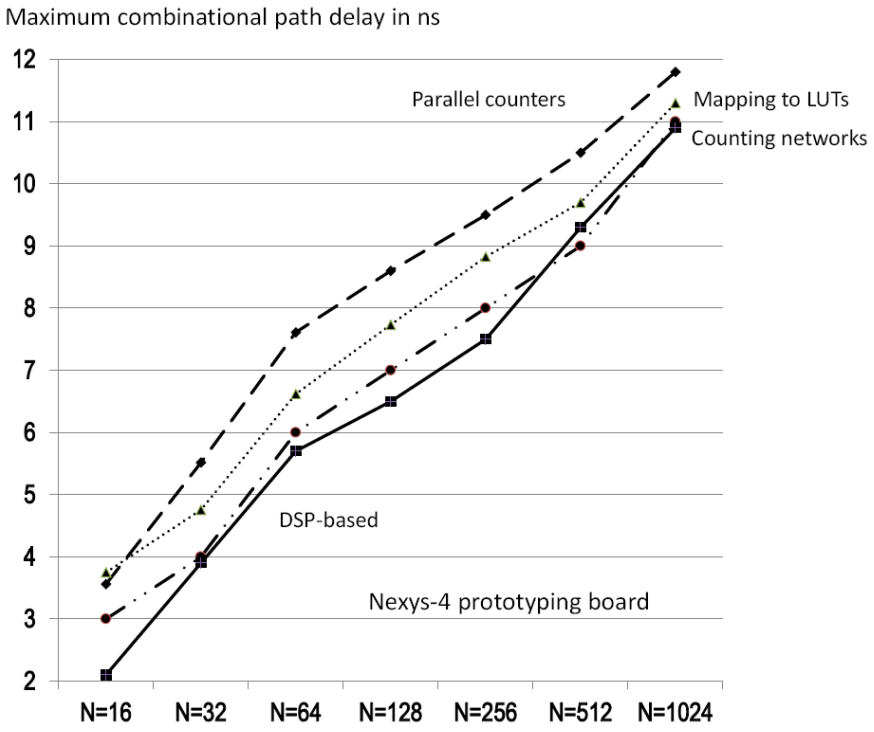

FIGURE 10. Comparison of the maximum combinational path delays

(Nexys-4 prototyping board [29] with Atrix-7 FPGA XC7A100T and clock frequency 100 MHz) compared to a pure software program written in Java and running in a desktop PC (processor i7-4770, 3.4 GHz). Note that the clock frequency of FPGA is 34 times slower than the PC. The table is filled in for five different combinations of the parameters N and M (see Figure 2) and for $\eta = 10$. For small values of N and M the circuit in Figure 4 does not provide any acceleration because the task is simple and the number of operations in software is limited. However, this is not a practical case. For all the remaining combinations acceleration is significant (the greater value of N/M, the higher the acceleration that is achieved). Substantially better acceleration has been reached in the fully parallel circuit referenced above. However, the resources available for the FPGA [29] used allow circuits to be implemented only for the first two columns of Table 2. For the first column ($N = M = 10$) the acceleration is 7 and for the second column ($N = M = 50$) it is 37.

In the second step we implemented and tested the circuit that computes the HW. Figure 10 shows the maximum combinational path delays for the best-known designs.

As we can see, counting networks [28] and circuits based on DSP slices [30] are the fastest. However, counting networks use just logical elements.

Figure 11 shows the number of DSP slices occupied (N_{DSP}) and the number of logical slices required, N_s, for different methods. Mapping to LUTs [31] is the less resource consuming method and counting networks require the largest resources. Thus, if high throughput is more important, then either counting networks [28] or DSP-based solutions should be chosen. If the resources used need to be minimized then the method [31]

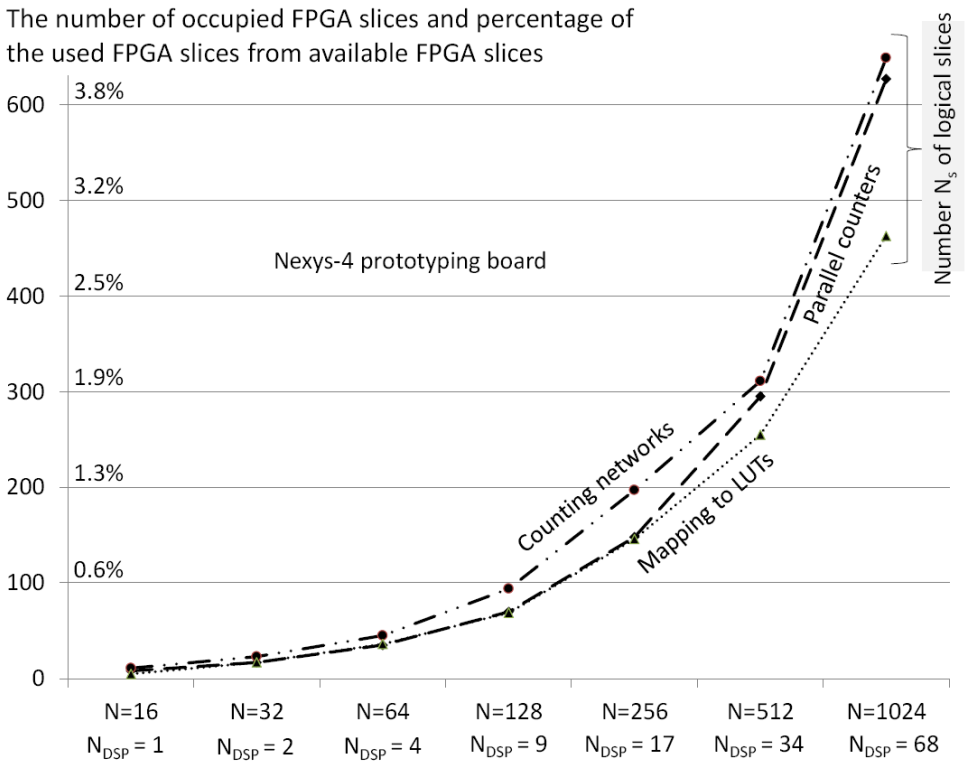

FIGURE 11. The occupied resources from the results of experiments with different designs

is the best option. This confirms the theoretical evaluation presented at the end of Section 6. Generally, Figures 10 and 11 enable the best circuit to be chosen for particular requirements.

8. **Conclusion.** The paper suggests fast hardware accelerators that simplify data manipulation and analysis of tables that contain subsets of items that somehow satisfy a set of desired requirements. The main objective is to determine how many items in the tables exactly satisfy the requirements and finally to extract such items. The problem is solved partially in software and partially in hardware accelerators implemented in reconfigurable logic available in FPGA and PSoC. The basic architecture of the hardware circuit is proposed, the method of data extraction is discussed, and examples are given. The problem considered requires Hamming weight to be computed and the paper suggests the best methods for this based on analysis and comparisons of known alternatives. The results of the hardware/software implementations that have been described and further experiments clearly demonstrate the advantages of the proposed technique.

Acknowledgments. The authors would like to thank Ivor Horton for his very useful comments and suggestions. This research was supported by Portuguese National Funds through FCT – Foundation for Science and Technology, in the context of the project UID/CEC/00127/2013. This work was partially supported by the grant of the Ministry of Education and Science of the Republic of Kazakhstan (project AP05133699 "Research and development of innovative information and telecommunication technologies using modern cyber-technical means for the city's intelligent transport system").

REFERENCES

[1] C. Wang (ed.), *High Performance Computing for Big Data: Methodologies and Applications*, CLR Press by Taylor & Francis Group, 2018.

[2] R. Chen and V. K. Prasanna, Accelerating equi-join on a CPU-FPGA heterogeneous platform, *Proc. of IEEE the 24th Annual Int. Symp. on Field-Programmable Custom Computing Machines*, Washington, DC, USA, pp.212-219, 2016.

[3] B. D. Rouhani, A. Mirhoseini, E. M. Songhori and F. Koushanfar, Automated real-time analysis of streaming big and dense data on reconfigurable platforms, *ACM Trans. Reconfigurable Technology and Systems*, vol.10, no.1, 2016.

[4] V. Sklyarov, I. Skliarova, A. Rjabov and A. Sudnitson, Computing sorted subsets for data processing in communicating software/hardware control systems, *International Journal of Computers Communications & Control*, vol.11, no.1, pp.126-141, 2016.

[5] Y. Gao, S. Huang and A. Parameswaran, Navigating the data lake with datamaran: Automatically extracting structure from log datasets, *Proc. of the 2018 International Conference on Management of Data – SIGMOD'18*, Houston, TX, USA, 2018.

[6] C. L. P. Chen and C. Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on big data, *Information Sciences*, vol.275, pp.314-347, 2014.

[7] B. Parhami, Computer architecture for big data, in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Zomaya (eds.), Springer, 2018.

[8] R. Chen and V. K. Prasanna, Computer generation of high throughput and memory efficient sorting designs on FPGA, *IEEE Trans. Parallel and Distributed Systems*, vol.28, no.11, pp.3100-3113, 2017.

[9] G. Chrysos, P. Dagritzikos, I. Papaefstathiou and A. Dollas, HC-CART: A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system, *ACM Trans. Architecture and Code Optimization*, vol.9, no.4, pp.1-25, 2013.

[10] V. Sklyarov, A. Rjabov, I. Skliarova and A. Sudnitson, High-performance information processing in distributed computing systems, *International Journal of Innovative Computing, Information and Control*, vol.12, no.1, pp.139-160, 2016.

[11] J. Lee, H. Kim, S. Yoo, K. Choi, H. P. Hofstee, G. J. Nam, M. R. Nutter, D. Jamsek and V. Extra, Boosting graph processing near storage with a coherent accelerator, *Proc. of the VLDB Endowment*, vol.10, no.12, pp.1706-1717, 2017.

[12] A. M. Caulfield, E. S. Chung, A. Putnam et al., A cloud-scale acceleration architecture, *Proc. of the 49th IEEE/ACM International Symposium on Microarchitecture – MICRO*, Taipei, Taiwan, pp.1-13, 2016.

[13] D. E. Knuth, *The Art of Computer Programming: Volume 3: Sorting and Searching*, 3rd Edition, Addison-Wesley, Massachusetts, 2011.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stain, *Introduction to Algorithms*, 3rd Edition, MIT Press, 2009.

[15] V. Sklyarov, I. Skliarova, A. Barkalov and L. Titarenko, *Synthesis and Optimization of FPGA-Based Systems*, Springer, Switzerland, 2014.

[16] P. R. Schaumont, *A Practical Introduction to Hardware/Software Codesign*, Springer, New York, 2013.

[17] B. Cope, P. Y. K. Cheung, W. Luk and L. Howes, Performance comparison of graphics processors to reconfigurable logic: A case study, *IEEE Trans. Computers*, vol.59, no.4, pp.433-448, 2010.

[18] S. Chey, J. Liz, J. W. Sheaffery, K. Skadrony and J. Lach, Accelerating compute-intensive applications with GPUs and FPGAs, *Proc. of Symposium on Application Specific Processors – SASP'08*, USA, pp.101-107, 2008.

[19] M. Santarini, Xilinx 16nm UltraScale+ devices yield 2-5X performance/watt advantage, *XCell Journal*, no.90, pp.8-15, 2015.

[20] S. Kestur, J. D. Davisz and O. Williams, BLAS comparison on FPGA, CPU and GPU, *Proc. of IEEE Computer Society Annual Symposium on VLSI*, Lixouri, Greece, pp.288-293, 2010.

[21] S. W. Aj-Haj Baddar and K. E. Batcher, *Designing Sorting Networks – A New Paradigm*, Springer, 2011.

[22] R. Mueller, J. Teubner and G. Alonso, Sorting networks on FPGAs, *The Int. Journal on Very Large Data Bases*, vol.21, no.1, pp.1-23, 2012.

[23] V. Sklyarov and I. Skliarova, Fast regular circuits for network-based parallel data processing, *Advances in Electrical and Computer Engineering*, vol.13, no.4, pp.47-50, 2013.

[24] B. Parhami, Efficient Hamming weight comparators for binary vectors based on accumulative and up/down parallel counters, *IEEE Trans. Circuits and Systems – II: Express Briefs*, vol.56, no.2, pp.167-171, 2009.

[25] V. Pedroni, Compact Hamming-comparator-based rank order filter for digital VLSI and FPGA implementations, *Proc. of IEEE Int. Symp. on Circuits and Systems*, Vancouver, Canada, vol.2, pp.585-588, 2004.

[26] V. A. Pedroni, Compact fixed-threshold and two-vector Hamming comparators, *Electronics Letters*, vol.39, no.24, pp.1705-1706, 2003.

[27] S. J. Piestrak, Efficient Hamming weight comparators of binary vectors, *Electronics Letters*, vol.43, no.11, pp.611-612, 2007.

[28] V. Sklyarov and I. Skliarova, Design and implementation of counting networks, *Computing Journal*, vol.97, no.6, pp.557-577, 2015.

[29] Digilent, Inc., *Nexys 4^{TM} FPGA Board Reference Manual*, https://reference.digilentinc.com/lib/exe/fetch.php?media=nexys:nexys4:nexys4_rm.pdf, 2016.

[30] V. Sklyarov and I. Skliarova, Multi-core DSP-based vector set bits counters/comparators, *Journal of Signal Processing Systems*, vol.80, no.3, pp.309-322, 2015.

[31] V. Sklyarov and I. Skliarova, Digital Hamming weight and distance analyzers for binary vectors and matrices, *International Journal of Innovative Computing, Information and Control*, vol.9, no.12, pp.4825-4849, 2013.

[32] Xilinx, Inc., *COE File Syntax*, www.xilinx.com.

[33] Xilinx, Inc., *Zynq-7000 All Programmable SoC Data Sheet: Overview*, https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, 2017.

[34] J. Silva, V. Sklyarov and I. Skliarova, Comparison of on-chip communications in Zynq-7000 all programmable systems-on-chip, *IEEE Embedded Systems Letters*, vol.7, no.1, pp.31-34, 2015.

[35] V. Sklyarov, I. Skliarova, J. Silva, A. Rjabov, A. Sudnitson and C. Cardoso, *Hardware/Software Co-design for Programmable Systems-on-Chip*, TUT Press, 2014.