Research of the Efficiency of High-level Synthesis Tool for FPGA Based Hardware Implementation of Some Basic Algorithms for the Big Data Analysis and Management Tasks

Alexander Antonov, Denis Besedin, Alexey Filippov
Peter the Great St. Petersburg Polytechnic University
St. Petersburg, Russia
antonov@eda-lab.ftk.spbstu.ru, 1310nero@mail.ru, filippov@eda-lab.ftk.spbstu.ru

Abstract - **The article is devoted to a research of an efficiency of high-level synthesis approach, based on Xilinx's high-level synthesis tool - Vivado, for a hardware implementation of sorting algorithms, which are one of the key algorithms for Big Data analysis, Data Mining, Data and Management. Performance and hardware costs are the measures of the efficiency in the provided research. The research methods are simulation and comparative analysis. Efficiency of software implementation of the selected sorting algorithms, based on a universal processor, is compared with efficiency of hardware implementation of the same sorting algorithms, obtained by high-level synthesis procedure with help of Xilinx's high-level synthesis tool. The article discusses approaches to optimize the description of the sorting algorithms and assignments in boundaries of high-level synthesis procedure to achieve optimal efficiency of the final hardware solutions. The article shows that the main efficiency gain is determinate by the internal features of the sorting algorithm, selected for hardware implementation; the ability to parallelize the processing of the source arrays, which is achieved both by the settings of the Vivado synthesis tool and description style used for source code. Article highlights research results and provide a direction for the future research works.**

I. Introduction

The modern trend in development of state-of-art computing systems is implementation of the Distributed Reconfigurable Heterogeneous High Performance Computing (DRH HPC) systems [1].

Due to rapidly increasing requirements for high performance computing systems in accordance with such criteria as:

- Performance, measured in FLoating-point Operations Per Second (FLOPS);
- Energy Efficiency (FLOPS/W);
- Performance Efficiency (Real performance FLOPS/Peak performance FLOPS);
- Area Efficiency (Real FLOPS/square),

DRH HPC need to have adaptation and hardware reconfiguration capabilities for effective implementation nearly any computing intensive algorithm [2], [3]. Modern Heterogeneous High Performance Computing consist on: multiprocessing units (MPU); single instruction multiple date (SIMD) accelerators, commonly known as General Purpose Graphic Processing Units (GP GPUS), and hardware reconfigurable accelerators, often pointed as Reconfigurable Computing Technology (RCT). The core of the state of art DRH HPC systems is Reconfigurable Computing Technology (RCT).

Reconfigurable computing technology uses Field-Programmable Gate Array (FPGA) [4], [5]. FPGA is an integrated Circuit (IC) that can change its internal structure in accordance with the solving task. Modern FPGA consists of programmable logic cells (LCELL) that can perform any logic/memory functions and programmable matrix (interconnection matrix) that can connect all logic cells together to implement complex functions. Binary file, often called configuration file, is a file to program or configure logic cells and interconnection matrix in the FPGA. Configuration file sets up the logic cells and the interconnection matrix such, that FPGA can implement the task being solved. State-of-art FPGA contains not only logic cells and interconnection matrix but also Digital Signal Processing (DSP) blocks; embedded memory blocks (BRAM); High Bandwidth Memory (HBM) blocks, based on embedded Double-Data Rate (DDR) memory; hardware implemented controllers and transceivers for external: DDR memory, PCIe interface, 100G Ethernet ports. Nearly all FPGA, which are in the market, could be configured on the fly. To configure FPGA on the fly means that FPGA configuration for solving a new task can be downloaded into FPGA during execution of the current task. Some modern FPGAs support a partial configuration and reconfiguration. The FPGA partial reconfiguration means that a part of FPGA can be configured for solving new task while the rest of FPGA continues to solve current task [6]. Finally, FPGA can be configured and partially reconfigured through PCIe and Ethernet interfaces.

From the system point of view, the DRH HPC systems allow, by using the available heterogeneous computational resources, particularly hardware reconfigurable FPGA based accelerators, temporarily, and, that is very important for the

overall system performance, on the fly, create highly specialized computational "pipes" for solving the particular tasks. The computational pipe can, in the simplest case, consist of just hardware reconfigurable FPGA based accelerators or, for solving a complex task, include SIMD accelerators, MPUs and hardware reconfigurable FPGA based accelerators working together by solving the particular task [7]. Proposed approach helps to satisfy the most important modern performance criteria for high-performance computing systems: Energy Efficiency and Performance Efficiency [8].

To implement an algorithm on hardware reconfigurable FPGA based accelerator it is necessary to prepare configuration file for the algorithm that will be downloaded to FPGA during runtime, i.e. by which FPGA will be configured or partially reconfigured for solving the particular task.

The traditional procedure for developing implementation for reconfigurable hardware devices is based on the using of Hardware Description Languages (HDL), for example, such as VHDL, Verilog HDL, System Verilog. This procedure is very time-consuming and requires hard work both at the stage of development and at the stage of debugging [9], [10].

A modern approach is to use the capabilities of high-level synthesis tools that are provided by leading FPGA manufacturers of programmable logic, such as Xilinx [11] and Intel PSG [12], and companies engaged in the development of electronic device development tools, for example, Mentor Graphics [13].

High-level synthesis tools allow not only to synthesize hardware solutions to algorithm described in high-level programming languages, such as C or C ++, but also to verify the correct operation of the synthesized algorithm, prepared for configuring FPGA, by applying the common (for software and hardware testing) test described in C or C ++.

Methodology of using the high-level synthesis tools to create reconfigurable hardware parts of heterogeneous computing systems is a rather new methodology, just like the high-level synthesis tools, and, at present, there are no reliable evaluations for the efficiency of using such methodology (and tools) for implementation of data processing algorithms with high computational complexity and significant memory requirements.

In order to analyze the efficiency of using high-level synthesis tools, it is necessary to carry out a simulation and a comparative analysis of software implementation, based on a universal processors, and hardware implementation , based on reconfigurable FPGA base accelerators, of the same algorithm, described in C or C ++.

Since the architectures of the universal processor and reconfigurable hardware are different, a comparative analysis can be performed according to efficiency criterion based on performance of the particular implementation.

The target of the research is to find the sorting algorithms (at least one) that, in accordance with their internal features, can help to achieve significant performance increase in solving sorting problems when those implemented in hardware, by using modern HLS tools, in comparison with the software implementations of the same algorithms on the universal processors.

II. Research of the efficiency of sorting algorithms implementations

A. Objects for the research

Objects for the current research are sorting algorithms. Such a choice is determined by the facts that:

- Wide using of the sorting algorithms for solving problems associated with Big Data analysis, Data Mining, Data Storage and Management;
- The relevance increasing performance of the sorting algorithms for speeding up many applications related to Big Data analysis, Data Mining and Data Management;
- The computational complexity and memory requirements of sorting algorithms.

To systematize the research, a simplified, based on a sorting method used, presented on Fig. 1, classification of the sorting algorithms is used.

This classification helps to choose at least one algorithm from every sorting method for further research [14].

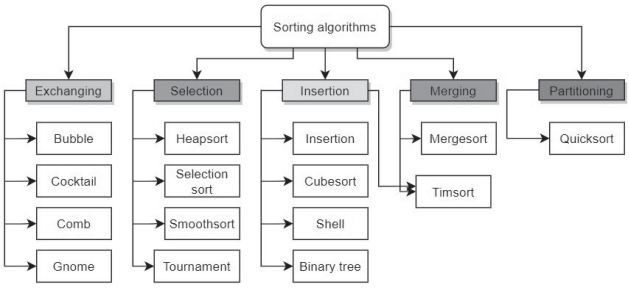

Fig. 1. Simplified classification of sorting algorithms

Previous hands-on experiences in the hardware implementation, based on HDL descriptions, of the sorting algorithms determined our first choice of a set of the algorithms for the current research.

The selected algorithms are: Gnome sort [15], Heap sort [16]; Shell sort [17]; Merge sort [18]; Quick sort[19].

The chosen algorithms complexity in the worst case are: Gnome – $O(n^2)$, Heap – O(n log n), Shell – $O(n \log^2 n)$, Merge – O(n log n), Quick – $O(n^2)$ [20]. The estimations of memory consumption for the algorithms are: Gnome – O(1), Heap – O(n), Shell – O(1), Merge – O(n), Quick – O(log n) [21].

The chosen algorithms are the typical representatives of the each sorting method indicated on Fig. 1.

The source codes of the selected sorting algorithms are in C language. Library functions of time.h were included in source codes for performance evaluation purposes.

B. The research methods

The used research methods are:

- Simulation of solving the sorting problems on computational structures with different architectures (on

an universal processor and on the reconfigurable FPGA based hardware);

- Comparative analysis according to the selected measures of the efficiency: performance, to compare software and hardware implementations, and hardware cost, to compare different hardware implementations with nearly the same performance.

The hardware cost is a number of logic elements (LCELL) and embedded memory blocks (BRAM) used in the FPGA to implement a particular sorting task.

The performance is a time, in nanosecond, for solving the particular sorting task.

The selection of these measures for comparative analysis is justified by the fact that the goals of creating reconfigurable FPGA based hardware solutions are to increase for the entire DRH HPC system:

- Power Efficiency (FLOPS/W);
- Performance Efficiency (Real FLOPS/Peak FLOPS).

The Simulation procedure and comparative analysis use the following tools:

For the software implementation:

- Integrated Development Environment (IDE) – JetBrains CLion [21].
- Multi-core system – Intel Core i7-4710HQ, 2.50GHz with 12G Bytes DDR3 RAM.

For the reconfigurable hardware implementation based on FPGA:

- IDE – Vivado HLS [3].
- FPGA – XCVU125-flvc2104-3 [7].

The selection of JetBrains CLion development environment is justified by the fact that it is a cross-platform C and C ++ development environment, which allows you to easily compile and run any programs using popular compilers such as: GCC, Clang, MinGW, Cygwin, and pre-installed libraries. Those for the purposes of the current research, means the opportunity for a wide range of users to repeat our results.

To evaluate performance of the software implementation of the particular sorting algorithm, working on an array of randomly generated values, the time interval between two control points during the program execution is estimated.

The Vivado HLS (High-level synthesis) Integrated Development Environment is able to:

- Synthesizes and implements on reconfigurable, FPGA based hardware, the sort algorithm described either by C or C ++ language. The tool is able to automatically:
- Evaluate all necessary data for estimation the performance of the synthesized implementation targeted to the particular FPGA;
- Estimate the expected hardware costs for the implementation.

This development environment allows us to optimize the implementation by assigning to use various resources available in the target FPGA and by pipelining and parallelizing hardware implementation according to user-defined criteria.

To evaluate the performance of a synthesized hardware solution, Vivado HLS calculates the minimum possible period of the clock frequency synchronizing the operation of the device. Than the tool estimates the number of the clock periods necessary to complete execution of the algorithm. That is, in other words, the tool calculates the number of clock cycles through which the input of the device that implements the synthesized algorithm can be fed by new data. Based on these data, performance, it term of time for sort algorithm execution, is calculated by multiplying the estimate of the minimum possible period of the clock frequency on the number of required clock cycles.

C. Conducting the research

The research procedure includes the following steps:

1) Creation of a source code of an algorithm suitable for both a software implementation based on a universal processor and for the synthesis for a reconfigurable FPGA based hardware solution. The source code should allow to process input, unsorted, arrays having different size.

2) Creation of a testbench that will be used both to verify the correct operation of the algorithm described on the C language, and to verify the behavior of the synthesized and implemented hardware solution. During the testing procedure, it is necessary to launch the tested sorting algorithm several times, since this allows simulating a continuous data stream characteristic of the hardware implementation. The code should allow creating source arrays of different size. The source arrays must be initialized by random integers with a uniform distribution.

3) Simulation software implementation based on a universal processor:

- Testing the source code of the algorithm for a given set of array sizes.
- Simulation and performance estimation for a given set of array sizes.

4) Simulation and optimization of reconfigurable FPGA based hardware implementation of the algorithm:

- Testing the source code of the algorithm in the framework of a high-level synthesis tool for a given set of array sizes.
- Iterative carrying out the stages of synthesis and optimization for a given set of array sizes, by applying a selected set of control directives. The goal is to achieve maximum performance for each set of array sizes having in mind that there are restrictions on available logical capacity for the particular FPGA.
- Co-simulation each hardware implementation of the algorithm for each set of array sizes, based on the

testbenches, used for testing the software implementation of the same sorting algorithm.

5) Comparative analysis of software and hardware implementations of the same sorting algorithm.

For the current research, the following sets of array sizes are selected: 100; 1000; 10000; 100000; 1000000. All numbers are of the type Integer (signed integer 32 bits). It allows simplifying comparative analysis by using native data type for the software implementation. It is necessary to pay attention on the fact that if data size is need to be reduced, for example by using any size which is less than 32 bits, than hardware implementation will have additional positive gap in performance.

During the synthesis and optimization, for each given set of array sizes, the following sets of control directives of the Vivado HLS were applied:

- Set of directive for choosing an interface architecture for implementing reading raw data and writing sorted data. These sets allow the synthesizer automatically use BRAM blocks for intermediate storage of raw and intermediate arrays. The target is to speed up the steps of reading the source data and writing sorted values.
- Set of pipeline directives for both internal and external loops used in the description of the sorting algorithm. Pipelining, depending on the internal features of the particular sorting algorithm, allows parallelization of reading the raw data, performing certain stages of data processing, and writing the sorted data.
- Set of directives for unrolling loops. The set allows increasing performance of the hardware implementation by executing a pointed number of sorting processes in parallel. When applying it is necessary to consider that, this directive requires many additional logic resources of FPGA.
- Set of dataflow directives for pipelining the hardware implementation at the level of data flows, i.e., for the sorting algorithms, at the level of data processing between cycles. Pipelining at the data flow level allows, depending on the internal features of the sorting algorithm, to compose an output array during the reading input array and the sorting procedure. This can make the implementation more adaptive to the features of the input data, for example, for the case if the array is sorted before the algorithm passes completely.

III. Results of the research

The estimation of the hardware cost for the initial, without any additional optimization directives, invocation of the high-level synthesis and implementation procedure is in Table I. The hardware cost is estimated in:

- LCELL - logical blocks of FPGA, consist of logical functions and synchronous flip-flops (FF), used to implement the particular sorting algorithm.
- BRAM - built-in memory blocks of FPGA, used to store intermediate data when implementing the particular sorting algorithm. This estimates is for internal memory blocks only. The hardware cost do not take into account external to FPGA memory blocks used for keeping raw and sorted arrays.

TABLE I. HARDWARE COSTS ESTIMATION WITHOUT OPTIMIZATION

Algorithm	Array size	LCELL	BRAM	Interface
Gnome sort	1.0E+2	290	0	ap_memory
	1.0E+3	290	0	
	1.0E+4	290	0	
	1.0E+5	290	0	
	1.0E+6	283	0	
Merge sort	1.0E+2	2496	2	ap_memory
	1.0E+3	2595	18	
	1.0E+4	2514	231	
	1.0E+5	2602	2834	
	1.0E+6	2648	33632	
Heap sort	1.0E+2	870	0	ap_memory
	1.0E+3	870	0	
	1.0E+4	870	0	
	1.0E+5	870	0	
	1.0E+6	870	0	
Quick sort	1.0E+2	1186	4	ap_memory
	1.0E+3	1211	4	
	1.0E+4	1230	38	
	1.0E+5	1264	358	
	1.0E+6	1299	3373	

The performance estimation for software implementation and for the initial, without any additional optimization directives, hardware implementation for the selected sorting algorithms is shown in Table II.

TABLE II. PERFORMANCE ESTIMATION WITHOUT OPTIMIZATION

Sort type	Array size	CPU time, s	Hardware implementation without any optimization		
			Clock period, ns	II	Time, s
Gnome sort	1.0E+2	4.60E-05	6.229	3.20E+04	2.84E-04
	1.0E+3	2.11E-03	6.229	2.05E+06	1.82E-02
	1.0E+4	5.14E-01	6.229	5.24E+08	4.65E+00
	1.0E+5	3.32E+01	6.229	3.35E+10	2.98E+02
	1.0E+6	2.14E+03	6.229	2.14E+12	1.90E+04
Merge sort	1.0E+2	5.00E-06	3.750	14E+2	5.33E-06
	1.0E+3	7.00E-05	3.767	20E+3	7.55E-05
	1.0E+4	1.10E-03	3.988	28E+4	1.12E-03
	1.0E+5	1.20E-02	4.106	82E+5	3.37E-02
	1.0E+6	1.40E-01	4.084	10E+7	4.08E-01
Heap sort	1.0E+2	1.48E-05	3.755	45E+2	1.70E-05
	1.0E+3	1.86E-04	3.755	68E+3	2.55E-04
	1.0E+4	2.33E-03	3.755	91E+4	3.42E-03
	1.0E+5	2.92E-02	3.755	11E+6	4.30E-02
	1.0E+6	2.73E-01	3.755	16E+7	5.87E-01
Quick sort	1.0E+2	7.37E-06	4.156	42E+7	1.73E+00
	1.0E+3	1.05E-04	4.156	40E+11	1.67E+04
	1.0E+4	1.20E-03	4.156	40E+15	1.66E+08
	1.0E+5	1.78E-02	4.156	> 4E+20	>4.00E+10
	1.0E+6	2.10E-01	4.156	> 4E+20	>4.00E+10

The performance of the hardware implementation in Table II is estimated in:

- Clock period – the minimum possible period of the clock frequency;
- II – initiation interval, the number of the clock periods necessary to complete execution of the particular sorting algorithm;
- Time – calculated by multiplying the estimates of the minimum possible period of the clock frequency on the number of required clock cycles.

Some conclusions that can be drawn from the analysis of the results of the initial, without optimization, stage of the research are below.

Shell sort shows very low performance and high hardware cost even for small arrays. According to this, the algorithm was rejected for next stages of the research. Results for the sorting algorithm are excluded from Table I and Table II.

We need to consider a limitation of modern high-level synthesis tools [11], [12], [13] associated with the inability to implement recursive algorithms.

In accordance with the pointed limitation the typical, recursive, form of Quick sort algorithm is not suitable for hardware implementation by using modern HLS tools. Results for implementation of non-recursive form of Quick sort algorithm are in Table I and Table II . Comparative analysis of the performance estimation shows that the algorithm demands a huge number of the clock periods necessary to complete its execution. According to this, there is no sense to optimize non-recursive form of Quick sort algorithm and the algorithm was rejected for further research.

Table III summarizes performance and hardware cost estimations, which were achieved after optimization of the high-level synthesis and implementation procedures.

TABLE III. EFFICIENCY OF OPTIMIZED IMPLEMENTATIONS

Sort type	Array size	CPU time spent, s	Hardware implementation after optimization		
			Time spent, s	LCELL	BRAM
Gnome sort	1.0E+2	4.60E-05	2.04E-04	290	0
	1.0E+3	2.11E-03	1.31E-02	290	0
	1.0E+4	5.14E-01	3.34E+00	290	0
	1.0E+5	3.32E+01	2.14E+02	290	0
	1.0E+6	2.14E+03	1.37E+04	283	0
Merge sort	1.0E+2	5.00E-06	5.33E-07	5.19E+03	1.20E+01
	1.0E+3	7.00E-05	5.19E-06	7.45E+03	1.80E+01
	1.0E+4	1.10E-03	5.18E-05	1.05E+04	2.47E+02
	1.0E+5	1.20E-02	5.18E-04	1.29E+04	2.52E+03
	1.0E+6	1.40E-01	1.55E-02	1.53E+04	3.37E+04
Heap sort	1.0E+2	1.48E-05	9.42E-06	1.10E+05	0
	1.0E+3	1.86E-04	1.41E-04	1.10E+05	0
	1.0E+4	2.33E-03	1.89E-03	1.10E+05	0
	1.0E+5	2.92E-02	2.38E-02	1.10E+05	0
	1.0E+6	2.73E-01	3.25E-01	1.10E+05	0

To optimize of the high-level synthesis and implementation procedures for Gnome sort algorithm the following assignments in Vivado HLS were used:

- AP_MEMORY interface with AP_SOURCE RAM2P.

To optimize of the high-level synthesis and implementation procedures for Merge algorithm the following assignments in Vivado HLS were used:

- AP_MEMORY interface with AP_SOURCE RAM2P.
- PIPELINE for merging arrays.
- DATAFLOW for the top function.
- ARRAY_PARTITION COMPLETE for an internal arrays.
- UNROLL COMPLETE for merging cycle.

To optimize of the high-level synthesis and implementation procedures for Heap sort algorithm the following assignments in Vivado HLS were used:

- AP_MEMORY interface with AP_SOURCE RAM2P.
- ARRAY_PARTITION COMPLETE for an input array.
- UNROLL COMPLETE for all cycles.

To simplify comparative performance analysis some figures based on Table III are below.

Fig. 2 visualizes the comparative analysis of the software-based implementation and the optimized hardware implementation of the Gnome sort algorithm. Results for Gnome sort are shown on a logarithmic scale for convenience. The Fig. 2 shows that Gnome sort algorithm implemented in hardware by using modern HLS tools works slower than the software implementation: by one order of magnitude slower for any arrays size.

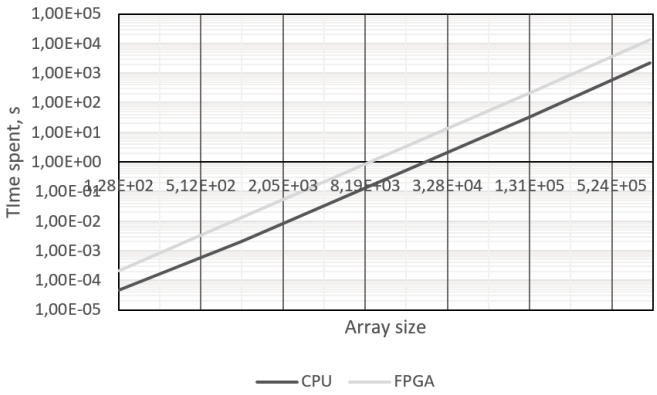

Fig. 2. Comparative performance analysis: Gnome sort

Fig. 3 visualizes the comparative analysis of the software-based implementation and the optimized hardware implementation of the Merge sort algorithm. The Fig. 3 shows that Merge sort algorithm implemented in hardware by using modern HLS tools works faster than the software implementation. It should be noted that the positive performance gap depends on the size of the array: the larger the array, the greater the positive gap. For the largest array in the current research, the gap is nearly one order of magnitude.

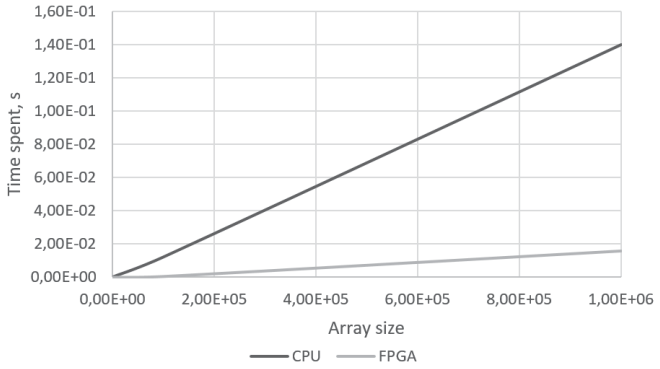

Fig. 3. Comparative performance analysis: Merge sort

Fig. 4 visualizes the comparative analysis of the software-based implementation and the optimized hardware implementation of the Heap sort algorithm. The Fig. 4 shows that Heap sort algorithm implemented in hardware by using modern HLS tools works slower than the software implementation for large arrays. It should be noted that the negative performance gap depends on the size of the array: the larger the array, the greater the negative gap. For the largest array in the current research, the negative gap is nearly two times.

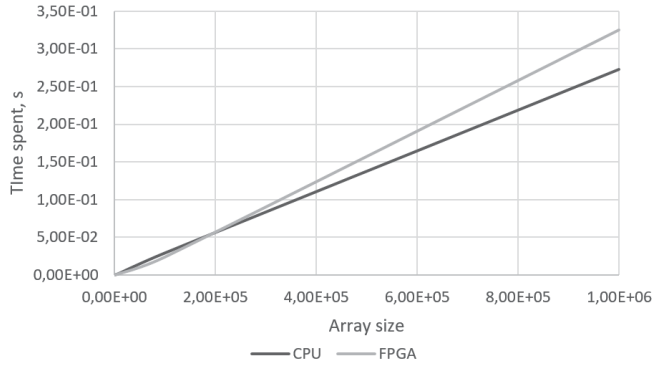

Fig. 4. Comparative performance analysis: Heap sort

Taking into account the sorting algorithm that gave in hardware implementation a performance gain relative to the software implementation (Merge sort and Heap sort), it is necessary to pay an attention to the fact, illustrated in Table III, that Merge sort algorithm, which gives a greater positive gain in the performance, also has a large hardware cost for implementation.

IV. Conclusion

The hardware implementation of the sorting algorithm achieved by utilizing of the modern HLS tool does not always provide higher performance than the execution of the same algorithm on a universal processor. Performance gap (positive or negative) depends on the internal features of the particular sorting algorithm, for example, whether it allows parallelization of reading, processing, writing the data, and on optimization directives applied.

There are some reasons, why the hardware implementations of the particular algorithms optimized by modern HLS tools are still slower, than the software implementation of the same algorithms:

- The algorithms assume a sequential execution of operations that is difficult to parallelize by using even state of art high-level synthesis tools.
- The clock speed of the universal processor is about an order of magnitude higher than the clock frequency for hardware implementation: the processor used for the current research has a clock frequency of 2.5 GHz, and the synthesized device, as it could be calculated from the data given in Table III, is about 250 MHz.

There are algorithms that cannot be implemented in hardware by using modern HLS tools, for example, recursive Quick sort algorithm, in accordance with the HLS tools restrictions.

The research shows that the non-recursive form of Quick sort algorithm implemented in hardware by HLS tool demonstrates a very low performance comparing to software implementation the same algorithm. The reasons are the same: internal features of the particular sorting algorithm and inability of the modern HLS tool to parallelize and pipeline the algorithm.

There are sorting algorithms, for example Merge sort algorithm found during current research, which have positive gap in performance when comparing with software implementation of the same algorithm. Such algorithms, in accordance with their internal features, allow to perform several sorting stages concurrently and to do pipelining. These are the reasons for the advantages in performance in comparison with the software implementation.

The current research gave a solid background for understanding capabilities of sorting algorithms and tools, which should be taken in account while doing further research.

The direction and scope of the further research is related with expansion of the number of sorting algorithms covered, taking an additional attention to the sorting methods which gave the better results.

The first target to the further research is Tim sort [22] algorithm. The algorithm combines, as pointed on Fig. 1, Merge sort and Insertion sort, which expected to make hardware implementation even faster than Merge sort, with reduced hardware costs.

The ultimate target for the further research should be to find a set of sorting algorithms which, implemented in hardware, by using modern HLS tools, will have the positive performance gap in comparison with the fastest software implementations of any sorting algorithms. This set should be ranked according to the hardware costs required to implement the algorithms.

V. Acknowledgment

The research was supported by international scientific-educational center “Embedded Microelectronic System” of Peter the Great St. Petersburg Polytechnic University.

REFERENCES

[1] A. Antonov, V. Zaborovskij, I. Kalyaev, "The architecture of a reconfigurable heterogeneous distributed supercomputer system for solving the problems of intelligent data processing in the era of digital transformation of the economy", *Cybersecurity Issues*, Aug. 2019, vol. 33, # 5, pp. 2-11. DOI:10.21681/2311-3456-2019-5-02-11.

[2] F. Mantovani, E. Calore, "Performance and Power Analysis of HPC Workloads on Heterogeneous Multi-Node Clusters", *Low Power Electronics and Application*. May 2018, vol. 8, #2, pp. 13-27. https://doi.org/10.3390/jlpea8020013

[3] M. Usman Ashraf, F. Alburaei Eassa, A. Ahmad Albeshri, A. Algarni, "Performance and Power Efficient Massive Parallel Computational model for HPC Heterogeneous Exascale Systems", *IEEE Access,* April 2018, vol. 6, pp. 23095-23107. DOI: 10.1109/ACCESS.2018.2823299.

[4] Xilinx official website, UltraScale and UltraScale+ FPGA, Web: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html#productTable

[5] Intel official website, Intel PSG and FPGA, Web: https://www.intel.com/content/www/us/en/products/programmable.html

[6] R. Kobayashi, Y. Oobata, N. Fujita, Y. Yamaguchi, T. Boku, "OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing", *In Proc. International Conference on High Performance Computing in Asia-Pacific*, HPC Asia, Jan. 2018, pp. 192-201. DOI:10.1145/3149457.3149479.

[7] A. Antonov, V. Zaborovskij, I. Kisilev, "Specialized reconfigurable computers in network-centric supercomputer systems", *High availability systems*, May 2018, vol.14, #3, pp. 57-62. DOI:10.18127/j20729472-201803-09

[8] J. Dongarra, S. Gottlieb, W. Kramer, "Race to Exascale", *Computing in Science and Engineering*, Feb. 2019, vol.21, #1, pp. 4-5. https://doi.org/10.1109/MCSE.2018.2882574

[9] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, J. Dongarra, "Investigating power capping toward energy-efficient scientific applications", *Concurrency and Computation Practice and Experience*, March 2018, pp. 1-14. DOI: 10.1002/cpe.4485

[10] V. Le Fèvre, T. Herault, Y. Robert, A. Bouteiller, A. Hori, J G. Bosilca, J. Dongarra, "Comparing the Performance of Rigid, Moldable and Grid-Shaped Applications on Failure-Prone HPC Platforms", *Parallel Computing*, July 2019, vol. 85, pp. 1-12. https://doi.org/10.1016/j.parco.2019.02.002

[11] Xilinx official website, IDE Vivado HLS, Web: https://www.xilinx.com/video/hardware/vivado-hls-tool-overview.html

[12] Intel official website, Intel HLS compiler, Web: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html?wapkw=HLS

[13] Mentor official website, Catapult HLS, Web: https://www.mentor.com/hls-lp/catapult-high-level-synthesis/

[14] W. Sherenaz, A. Baddar, K. Batcher, *Designing Sorting Networks. A New Paradigm*. Springer, 2011. DOI: 10.1007/978-1-4614-1851-1.

[15] Sorting Algorithm, Gnome sort, Web: http://rosettacode.org/wiki/Sorting_algorithms/Gnome_sort

[16] Sorting Algorithm, Heap sort, Web: http://rosettacode.org/wiki/Sorting_algorithms/Heapsort

[17] Sorting Algorithm, Shell sort, Web: http://rosettacode.org/wiki/Sorting_algorithms/Shell_sort

[18] Sorting Algorithm, Merge sort, Web: http://rosettacode.org/wiki/Sorting_algorithms/Merge_sort

[19] Sorting Algorithm, Quick sort, Web: http://rosettacode.org/wiki/Sorting_algorithms/Quicksort

[20] Sorting And Searching Algorithms, Time and Space Complexities, Web: https://www.hackerearth.com/ru/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/

[21] Software IDE, Clio Web: https://www.jetbrains.com/clion/

[22] Sorting Algorithm, Tim Sort Web: https://dev.to/s_awdesh/timsort-fastest-sorting-algorithm-for-real-world-problems--2jhd