DOI: 10.18721/JCSTCS.13103
УДК 004

EFFICIENCY ANALYSIS OF HIGH-LEVEL SYNTHESIS TOOLS FOR HARDWARE IMPLEMENTATION OF SORTING ALGORITHMS

A.P. Antonov, D.S. Besedin, A.S. Filippov

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

The article is devoted to the research of efficiency of Xilinx's high-level synthesis tools, the Vivado HLS package version 2019.2, for synthesis of a hardware implementation of sorting algorithms. The relevance of creating hardware implementation of sorting algorithms is determined by modern approaches to building high-performance heterogeneous computing systems and modern criteria for the efficiency of such systems – the ratio of performance to power consumption and the ratio of real performance to peak performance. The authors carried out a comparative analysis of the implementation of the selected sorting algorithms on a universal processor and on the basis of the VLSI Xilinx submarine research. The article discusses approaches to optimize the description of algorithms and control the Vivado HLS package to achieve optimal performance of the resulting hardware solutions. The article shows that the main performance gain is provided by parallelizing of the source arrays processing, which is achieved both by the settings of the design tool, the Vivado HLS package, the selected description style, as well as the features of the sorting algorithm selected for hardware implementation.

Keywords: hardware acceleration, sorting algorithms, high-level synthesis, reconfigurable hardware accelerator, FPGA.

Citation: Antonov A.P., Besedin D.S., Filippov A.S. Efficiency analysis of high-level synthesis tools for hardware implementation of sorting algorithms. Computing, Telecommunications and Control, 2020, Vol. 13, No. 1, Pp. 31-41. DOI: 10.18721/JCSTCS.13103

АНАЛИЗ ЭФФЕКТИВНОСТИ СРЕДСТВ ВЫСОКОУРОВНЕВОГО СИНТЕЗА ДЛЯ АППАРАТНОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ СОРТИРОВКИ

А.П. Антонов, Д.С. Беседин, А.С. Филиппов

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Статья посвящена исследованию эффективности средств высокоуровневого синтеза компании Xilinx, пакета Vivado HLS версии 2019.2, для создания аппаратной реализации алгоритмов сортировки. Актуальность создания аппаратной реализации алгоритмов сортировки определяется современными подходами к построению высокопроизводительных гетерогенных вычислительных систем и современными критериями эффективности таких систем: отношению производительности к энергопотреблению и отношению реальной производительности к пиковой производительности. Проведен сравнительный анализ реализации выбранных алгоритмов сортировки на универсальном процессоре и на базе СБИС ПЛ компании Xilinx. Рассмотрены подходы и способы оптимизации описания алгоритмов

и управления пакетом Vivado HLS для достижения оптимальных показателей эффективности полученных аппаратных решений. Показано, что основной выигрыш в производительности дает возможность частичного распараллеливания процесса обработки исходных массивов, что достигается как настройками средства проектирования – пакета Vivado HLS, выбранным стилем описания, так и особенностями алгоритма сортировки, выбранного для аппаратной реализации.

Ключевые слова: аппаратное ускорение, алгоритмы сортировки, высокоуровневый синтез, реконфигурируемый аппаратный вычислитель, СБИС программируемой логики.

Ссылка при цитировании: Антонов А.П., Беседин Д.С., Филиппов А.С. Анализ эффективности средств высокоуровневого синтеза для аппаратной реализации алгоритмов сортировки // Информатика, телекоммуникации и управление. 2020. Т. 13. № 1. С. 31-41. DOI: 10.18721/JCSTCS.13103

Introduction

A modern trend in the development of computing systems is the creation of heterogeneous distributed hardware reconfigurable systems that provide a solution to the problem of hardware adaptation and reconfiguration for the algorithm to solve the main problem [1]. Such approach to the construction of computing systems allows us to create temporarily highly specialized hardware devices using the computing resources available as part of the system, in accordance with the logic of the problem to be solved. It provides a more efficient solution of computationally complex algorithms than universal processors and devices with SIMD architecture [2]. The most important modern performance criteria for high-performance computing systems are energy efficiency, i.e. performance-to-power ratio, and computational efficiency – the ratio of real performance to peak performance [3, 4].

The traditional procedure for developing specialized hardware devices based on the use of hardware description languages, for example, VHDL, Verilog HDL, System Verilog, is laborious and requires significant time both at the stage of device development and at the stage of debugging [5].

A modern approach for developing specialized hardware devices is to use the capabilities of high-level synthesis tools that are provided by leading VLSI manufacturers of programmable logic, such as Xilinx [6] and Intel PSG [7], and companies engaged in the development of electronic device development tools, for example, Mentor Graphics [8].

High-level synthesis tools allow both to synthesize hardware solutions to problems described in high-level programming languages, such as C or C ++, and to verify the correct operation of the algorithm and the synthesized device using a single test described in C or C ++.

The use of high-level synthesis tools to create computing systems is a new approach and there are currently no reliable data on its effectiveness in the implementation of many 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 comparative analysis of the implementation of the same algorithm described in C or C ++ based on a universal processor and its hardware implementation obtained as a result of high-level synthesis. A comparative analysis can be performed according to such an efficiency criterion as performance, or runtime, when solving a task of a given dimension, since the architecture of the universal processor and reconfigurable hardware are different.

Object to research

Sorting algorithms were chosen as the object of this research paper due to their widespread use for solving problems associated with processing large data, their computational complexity, memory requirements and the relevance of the task of accelerating their execution for many applications related to database processing.

A simplified classification of sorting algorithms is shown in Fig. 1. Among the variety of sorting algorithms [9], several typical algorithms were selected for this research: comb sorting, gnome sorting and merge sorting.

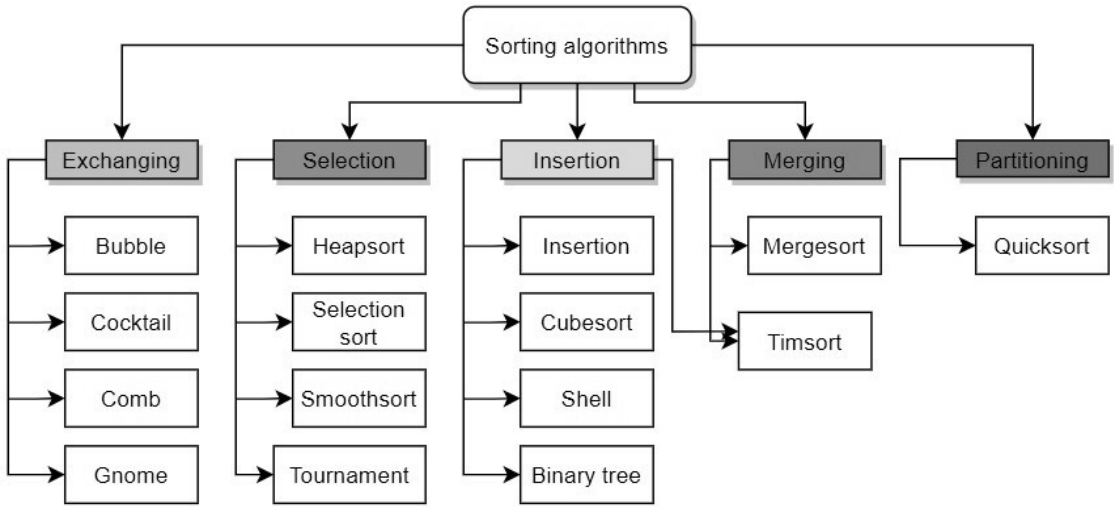

Fig. 1. A simplified classification of sorting algorithms

These algorithms are typical representatives of the classes shown in Fig. 1, and that is the reason to choose them for the research. Moreover, for choosing objects of research such as sorting algorithms, we need to allow for the significant limitation of modern high-level synthesis tools due to the impossibility of implementing recursive algorithms.

Comb sorting and gnome sorting belong to the class of exchange sorting algorithms, they are simple to implement, are considered the slowest when implemented on universal processors, have high O (n^2) computational complexity and do not require additional memory costs, like all representatives of this class (O(1)) [10, 11].

Merge sorting is an algorithm with sorting principle significantly different from exchange sorting algorithms, but it is also simple to implement. This sorting algorithm is faster than exchange algorithms when running on universal processors, since it has less computational complexity O (n log n), but significantly higher memory costs O(n) [12].

Method and research methodology

The research method is simulation of solving sorting problems on computational structures with different architectures and conducting a comparative analysis according to the selected criteria.

As a criterion for a comparative analysis, we selected performance parameters related to each other, that is, the number of operations performed in a given unit of time, and speed, which is the time spent on the task. The selection of these criteria for comparative analysis is justified by the fact that the goal of creating hardware solutions is to increase speed and productivity in solving computationally complex problems and, as a result, increase the computational efficiency of the entire high-performance system.

Here is a list of selected hardware and software tools used in this research, simulation and comparative analysis:

- For the software implementation on a universal processor:
  - IDE – JetBrains CLion;
  - Hardware part – PC based on Intel Core i7-4710HQ 2.50 GHz, with 12 GB RAM, type DDR3.
- For the hardware implementation based on FPGA:
  - IDE – Vivado HLS (High level synthesis) [6];
  - Hardware part – FPGA family Virtex UltraScale by Xilinx [13]: XCVU 125-flvc2104-3-e.

The JetBrains development environment CLion is a cross-platform C and C ++ development environment developed by JetBrains. It allows you to easily compile and run any programs using popular compilers (GCC, Clang, MinGW, Cygwin) and pre-installed libraries. It means the ability to work with the same source code of the program with the addition of standard C libraries operators for calculating the expended time.

For this purpose, the description of the sorting algorithms in the C language was made using the functions of the library time.h, which allows us to estimate the time interval between two control points during program execution, which, thus, when simulating a solution to a problem on a universal processor, provides data on speed and performance for comparative analysis.

The Vivado HLS (High level synthesis) development environment synthesizes the description of the device operation algorithm presented in C or C ++ into a hardware implementation; evaluates the performance and speed of a synthesized device; displays the expected hardware "cost" for its implementation on the basis of the selected element base – the selected FPGA part. This development environment allows optimization of the created device during synthesis, setting up its implementation to use various resources available in the target FPGA part; pipelining and parallelizing hardware implementation according to user-defined criteria.

To assess the performance and speed of a synthesized device, Vivado HLS offers the calculation of the minimum possible period of the clock frequency synchronizing the operation signal of the device, and an estimate of the number of periods of the clock frequency for the complete execution of the algorithm, in other words, the number of clock cycles through which the input of the device that implements the synthesized algorithm can be fed new data. It is possible to calculate the time of one sorting, by multiplying the estimate of the period of the clock frequency by the number of required clock cycles based on these data.

The research methodology includes the following steps:

• The creation of a text code description of an algorithm suitable for both a software implementation based on a universal processor and for the synthesis of a reconfigurable hardware solution. In the created description, the means of controlling the runtime on the basis of a universal processor are used. The description should allow to process arrays of input data of arbitrary size.

• The creation of a test text code description that will be used to verify both the correct operation of the initial description of the algorithm in the C language and the model of the synthesized hardware solution. In the created test description, it is necessary to launch the function of the tested algorithm several times, since this allows you to simulate a continuous data stream characteristic of a hardware implementation. The description should allow you to create arrays of source data of arbitrary size. The source arrays must be initialized random, with a uniform distribution, integers.

• Simulation based on a universal processor:

- Test of the initial description of the algorithm based on a universal processor for a given set of array sizes;

- Software implementation of an algorithm based on a universal processor for a given set of input array sizes. Obtaining a set of characteristics for the execution time of the algorithm.

• Simulation and optimization of hardware implementation of the algorithm:

- Test of the initial description of the algorithm in the framework of a high-level synthesis system on a given set of array sizes;

- Iteratively conducting synthesis-optimization stages for a given set of array sizes and a selected set of control directives for a high-level synthesis system. The goal is to achieve maximum performance for each set of array sizes if there is a limit – the logical capacity selected of the selected FPGA part;

- Hardware and software testing, based on a common test for hardware and software implementations, of each optimal hardware implementation of the algorithm for each set of array sizes.

• Comparative analysis of software and hardware implementations of the same algorithm.

Conducting research

The following sets of array sizes were selected for the research: 128; 1024; 16384; 32768; 65536; 131072. All numbers were of type Integer (signed integer 32 bits).

The description in C language of the merge sorting algorithm used for simulation, both for a software implementation based on a universal processor and for synthesizing a hardware implementation of an algorithm, is shown in Fig. 2. In this code, for clarity, the directives for optimizing hardware implementation and design are omitted, providing an assessment of productivity and performance in software implementation.

```
#define SIZE 65536
#define STAGES 16
void merge_arrays(int in[SIZE], int width, int out[SIZE]) {
    int f1 = 0;
    int f2 = width;
    int i2 = width;
    int i3 = 2*width;
    if(i2 >= SIZE) i2 = SIZE;
    if(i3 >= SIZE) i3 = SIZE;
    merge_arrays: for (int i = 0; i < SIZE; i++) {
        int t1 = in[f1];
        int t2 = (f2 == i3) ? 0 : in[f2];
        if(f2 == i3 || (f1 < i2 && t1 <= t2)) {
            out[i] = t1;
            f1++; }
        else{
            out[i] = t2;
            f2++;}
        if(f1 == i2 && f2 == i3){
            f1 = i3;
            i2 += 2*width;
            i3 += 2*width;
            if(i2 >= SIZE) i2 = SIZE;
            if(i3 >= SIZE) i3 = SIZE;
            f2 = i2;}}}
void merge_sort_parallel(int A[SIZE], int B[SIZE]) {
    int temp[STAGES - 1][SIZE];
    int width = 1;
    merge_arrays(A, width, temp[0]);
    width *= 2;
    int stage;
    for (stage = 1; stage < (STAGES - 1); stage++) {
        merge_arrays(temp[stage - 1], width, temp[stage]);
        width *= 2;}
    merge_arrays(temp[STAGES - 2], width, B); }

```

Fig. 2. Description of merge sort algorithm in C language

During the iterative stages of synthesis optimization, for each given set of array sizes, the following sets of control directives of the Vivado HLS high-level synthesis system were selected:

• Directives for choosing an interface architecture for implementing reading raw data and writing sorted data:

- This allows the synthesizer with certain interface architectures to automatically use BRAM blocks for intermediate storage of an array of numbers;

- This allows you to speed up the steps of reading the source data and writing sorted values in some cases, depending on the features of the algorithm.

• Pipeline directives for both internal and external loops in the description of algorithms:

- Pipelining allows parallelization of both reading the source data, performing individual steps of data processing, and recording sorted data in certain cases, depending on the features of the algorithm.

• Dataflow directives for pipelining at the level of data flows, that is, in relation to the considered implementations of the algorithm, at the level of data processing between cycles:

- Pipelining at the data flow level allows the device to compose an output array during the sorting procedure in some cases, depending on the features of the algorithm. That can make the device more adaptive to the features of the input data, for example, for the case if the array is sorted before the algorithm passes completely.

Research results

As was pointed above, a set of sorting algorithms were considered for hardware implementations synthesized by HLS tool. There are several limitations imposed by the HLS tool on the description of the investigated algorithms. These are:

- Programming language must be C or C ++.
- It must be a non-recursive description.
- Dynamic memory allocation should not be used.

As a result of the research, it was found that for the sorting algorithms of comb and gnome sorting, the optimal hardware implementations, devices obtained as a result of synthesis, have a similar architecture, the features of which are:

- Using dual port RAM memory for input array.
- Using the DataFlow directive, which provides for "forwarding" of the processed data between the internal cycles of the algorithm with the implementation of ping-pong mode.

For the merge sorting algorithm, the hardware solution obtained as a result of synthesis and optimization, a device that implements the specified algorithm, has the following architectural features:

- Dual port RAM memory is used to store the sorted array.
- Implement pipelining of the merge cycle of two arrays.
- During the optimization of this device, the following steps were applied:

- Arrays for storing intermediate data are divided into separate memory blocks, which allows simultaneous merging of different arrays;

- Merge loop in the main sorting function are unrolled for parallel implementation of all loop iterations. In addition to the previous paragraph, this allows you to merge different parts of the entire array of numbers at the same time, that is, to maximize parallelizing of the sorting process.

Table 1 shows the estimates of hardware "cost" for the implementation of optimal hardware devices (optimality criteria were considered above) for the indicated sorting algorithms: Comb, Gnome, Merge.

Table 1

Hardware "cost" of synthesized devices

Array size (number of samples)	Algorithm					
	Comb		Gnome		Merge	
	LCELL, num.	BRAM, num.	LCELL, num.	BRAM, num.	LCELL, num.	BRAM, num.
128	634	0	340	0	6260	12
1024	688	0	346	0	9032	18
16384	741	0	354	0	12812	277
32768	798	0	356	0	13877	812
65536	813	0	351	0	14626	1740
131072	849	0	353	0	15591	3712

Where:

• LCELL – logical blocks (cells) of FPGA part, that contains look-up tables (LUT) used to implement logical functions, and synchronous triggers (FF) used to store data.

• BRAM – built-in memory blocks that are used to store intermediate data when implementing the sorting algorithm. These embedded memory blocks are taken into account while estimating hardware "cost" of the algorithm. The external memory, which is necessary for storing the source and sorted arrays is not taken into account because this hardware "cost" is a constant for all sorting algorithms.

For clarity, the data shown in Table 1 about the logic blocks used to implement each of the algorithms (LCELL) are summarized in one graph, shown in Fig. 3.

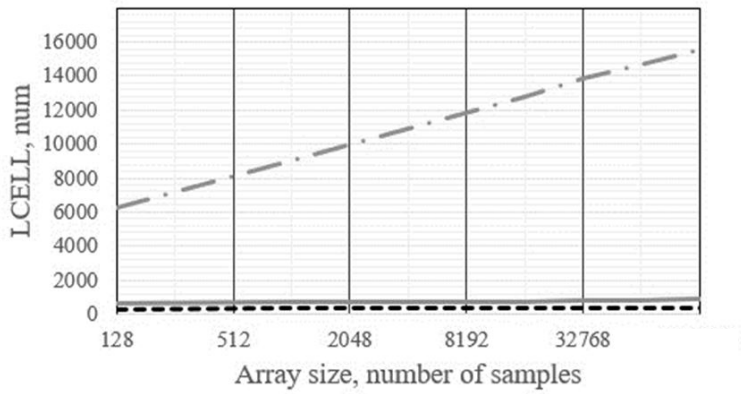

Fig. 3. Hardware "cost" of LCELL synthesized devices
(—) – Comb; (---) – Gnome; (— ·) – Merge

An analysis of the graphs in Fig. 3 shows that the hardware "cost" for implementing sorting algorithms with comb and gnome sorting are significantly lower than the hardware "cost" for implementing the merge sorting algorithm. In this case, there is a directly proportional relationship between the size of the sorted array and the number of logical cells used to implement the merge sort algorithm.

Table 2 shows the performance estimates for the optimal implementation of all synthesized devices obtained in the framework of the Vivado HLS development environment.

Table 2

Performance assessment of sorting devices

Array size (number of samples)	Algorithm					
	Comb		Gnome		Merge	
	Period, ns	Latency, num. of clocks	Period, ns	Latency, num. of clocks	Period, ns	Latency, num. of clocks
128	4.066	35219	6.229	32770	5.176	916
1024	4.066	2141219	6.229	2097154	5.176	10269
16384	4.066	537575459	6.229	536870914	5.176	229417
32768	4.066	2148892707	6.229	2147483650	5.176	491804
65536	4.066	8592752675	6.229	4294967300	5.176	1048623
131072	4.066	34371665987	6.229	8589934600	5.176	2228274

Where:

• Latency is the number of clock cycles of the synchronization signal required to obtain a ready, sorted, array at the device output.

• Period is the minimum possible period of the synchronization signal.

Therefore, you can calculate the time to complete the array sorting, and, therefore, you can determine the performance of the synthesized hardware implementation of the sorting algorithm by multiplying the period and the number of delay ticks.

Table 3 shows the estimates obtained in the framework of the study on the execution time of the selected sorting algorithms on a universal processor and on the basis of synthesized hardware computers.

Table 3

Estimation of sorting time

Array size (number of samples)	Algorithm					
	Comb		Gnome		Merge	
	CPU, s	FPGA, s	CPU, s	FPGA, s	CPU, s	FPGA, s
128	0.000021	0.000157	0.000046	0.0002041	0.00002	0.0000047
1024	0.00102	0.009651	0.00211	0.01306	0.00016	0.0000531
16384	0.3204	2.3895	0.514394	3.3442	0.00252	0.001187
32768	1.3156	9.5581	2.06334	13.3767	0.00496125	0.002544
65536	5.4017	38.2324	8.2765	53.5067	0.0099225	0.005427
131072	22.18	152.782	33.1937	214.0268	0.019845	0.01085

For clarity and simplification of the analysis, the data given in Table 3 are summarized in the graphs presented in Fig. 4 and Fig. 5. All graphs in the figures are presented in a logarithmic scale: base 10 for the ordinate axis, base 2 for the abscissa axis.

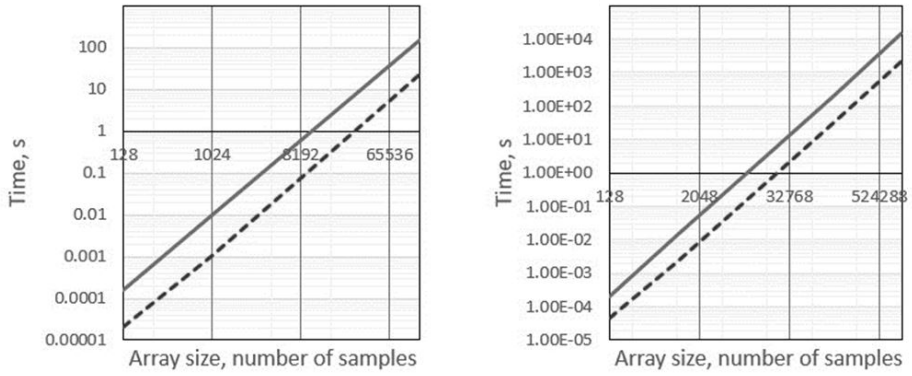

Fig. 4. Dependence of the sorting time with a comb (left) and a gnome (right) on the size of the array (---) – CPU; (—) – FPGA

Conclusions

Analysis of the research results allowed us to draw the conclusions below.

The hardware implementation of the algorithm does not always provide greater performance compared to the execution of the algorithm on a universal processor. So, the graphs in Fig. 4 show that the execution

time of sorting algorithms with a comb and gnome on a universal processor is an order of magnitude shorter than the time achievable with the hardware implementation of these algorithms. These results can be explained:

• These algorithms involve sequential operations that are difficult to parallelize using 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 study has a clock frequency of 2.5 GHz, and the synthesized device, as it is not difficult to calculate from the data given in Table 2, is about 250 MHz. Therefore, in the sequential implementation of the algorithm, the universal processor demonstrates a tenfold gain in the execution time of the algorithm.

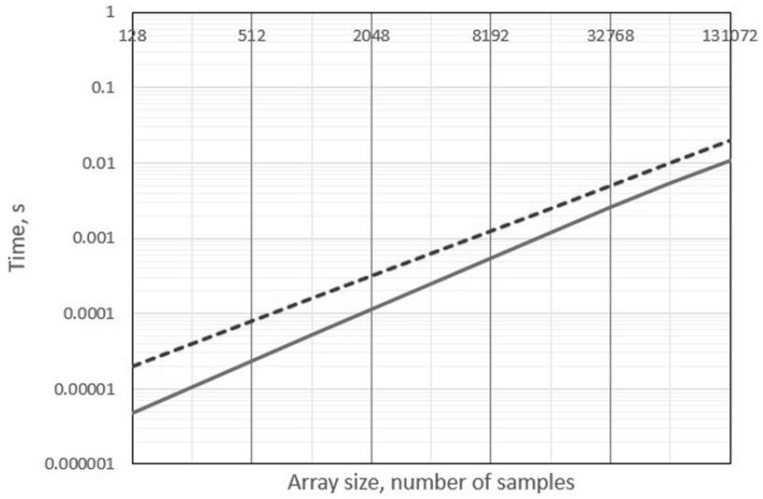

Fig. 5. Dependence of merge sorting time on array size
(---) – CPU; (—) – FPGA

The merge sorting algorithm allows you to parallelize its execution, so the synthesized hardware calculator allows you to simultaneously perform several sorting stages, stream parallelization, and several operations of one sorting stage, pipelining, which provides it with an advantage in speed compared to running the algorithm on a universal processor, see Table 3 and Fig. 5. Moreover, as can be seen from Table 1 and Fig. 3, while the number of samples of the processed arrays are increasing, the hardware expenses are growing as well. The limitation for parallelization is the number of logic elements in modern FPGA circuits.

The further direction of the research is related to the expansion of the number of sorting algorithms covered and the search for such algorithms that will maximize the capabilities of modern FPGAs and high-level synthesis tools, providing a significant performance increase in solving sorting problems in comparison with the best, with the fastest, implementations of sorting algorithms on universal processors.

REFERENCES

1. **Antonov A.P., Zaborovskiy V.S., Kalyayev I.A.** Architecture of reconfigurable heterogeneous distributed supercomputer system for solving problems of intelligent data processing in the era of digital transformation of the economy. *Voprosy Kiberbezopasnosti*, 2019, Vol. 33, No. 5, Pp. 2–11. (rus). DOI:10.21681/2311-3456-2019-5-02-11

2. **Antonov A.P., Zaborovskiy V.S., Kiselev I.O.** The reconfigurable computational modules in network-centric supercomputer systems. *Sistemy vysokoy dostupnosti*, 2018, Vol. 14, No. 3, Pp. 57–62. (rus). DOI:10.18127/j20729472-201803-09

3. **Mantovani F., Calore E.** Performance and power analysis of HPC workloads on heterogeneous multi-node clusters. *Low Power Electron*, 2018, Vol. 2, No. 8, Pp. 1–14. DOI:10.3390/ jlpea8020013

4. **Usman A., Fathy A., Aiiad A., Abdullah A.** Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems. *IEEE Access*, 2018, No. 6, Pp. 23095–23107. DOI:10.1109/ACCESS.2018.2823299

5. **Kobayashi R., Oobata Y., Fujita N., Yamaguchi Y., Boku T.** OpenCL-ready high speed FPGA network for reconfigurable high performance computing. *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2018, Pp. 192–201. DOI:10.1145/3149457.3149479

6. Sreda Vivado HLS. Available: *https://www.xilinx.com/video/hardware/vivado-hls-tool-overview.html* (Accessed: 30.01.2020).

7. Intel HLS compiler. Available: *https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html?wapkw=HLS* (Accessed: 30.01.2020).

8. Catapult HLS. Available: *https://www.mentor.com/hls-lp/catapult-high-level-synthesis/* (Accessed: 30.01.2020).

9. Algoritmy sortirovki. Available: *https://en.wikipedia.org/wiki/Sorting_algorithm* (Accessed: 30.01.2020).

10. Sorting Algorithm – Comb Sort. Available: *https://www.ideserve.co.in/learn/comb-sort* (Accessed: 31.01.2020).

11. Gnomya sortirovka. Available: *https://ru.wikipedia.org/wiki/Gnomya_sortirovka* (Accessed: 30.01.2020). (rus)

12. Sortirovka sliyaniyem. Available: *https://ru.wikipedia.org/wiki/Sortirovka_sliyaniyem* (Accessed: 06.02.2020). (rus)

13. UltraScale and UltraScale + FPGA. Available: *https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html#productTable* (Accessed: 31.01.2020).

Received 16.01.2020.

СПИСОК ЛИТЕРАТУРЫ

1. **Антонов А.П., Заборовский В.С., Каляев И.А.** Архитектура реконфигурируемой гетерогенной распределенной суперкомпьютерной системы для решения задач интеллектуальной обработки данных в эпоху цифровой трансформации экономики // Вопросы кибербезопасности. 2019. Т. 33. № 5. С. 2–11. DOI:10.21681/2311-3456-2019-5-02-11

2. **Антонов А.П., Заборовский В.С., Киселев И.О.** Специализированные реконфигурируемые вычислители в сетецентрических суперкомпьютерных системах // Системы высокой доступности. 2018. Т. 14. № 3. С. 57–62. DOI:10.18127/j20729472-201803-09

3. **Mantovani F., Calore E.** Performance and power analysis of HPC workloads on heterogeneous multi-node clusters // Low Power Electron. 2018. Vol. 2. No. 8. Pp. 1–14. DOI:10.3390/ jlpea8020013

4. **Usman A., Fathy A., Aiiad A., Abdullah A.** Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems // IEEE Access. 2018. No. 6. Pp. 23095–23107. DOI:10.1109/ACCESS.2018.2823299

5. **Kobayashi R., Oobata Y., Fujita N., Yamaguchi Y., Boku T.** OpenCL-ready high speed FPGA network for reconfigurable high performance computing // Proc. of the Internat. Conf. on High Performance Computing in Asia-Pacific Region. 2018. Pp. 192–201. DOI:10.1145/3149457.3149479

6. Среда Vivado HLS // URL: https://www.xilinx.com/video/hardware/vivado-hls-tool-overview.html (Дата обращения: 30.01.2020).

7. Intel HLS compiler // URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html?wapkw=HLS (Дата обращения: 30.01.2020).

8. Catapult HLS // URL: https://www.mentor.com/hls-lp/catapult-high-level-synthesis/ (Дата обращения: 30.01.2020).

9. Алгоритмы сортировки // URL: https://en.wikipedia.org/wiki/Sorting_algorithm (Дата обращения: 30.01.2020).

10. Sorting Algorithm – Comb Sort // URL: https://www.ideserve.co.in/learn/comb-sort (Дата обращения: 31.01.2020).

11. Гномья сортировка // URL: https://ru.wikipedia.org/wiki/Гномья_сортировка (Дата обращения: 30.01.2020).

12. Сортировка слиянием // URL: https://ru.wikipedia.org/wiki/Сортировка_слиянием (Дата обращения: 06.02.2020).

13. UltraScale and UltraScale+ FPGA // URL: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html#productTable (Дата обращения: 31.01.2020).

Статья поступила в редакцию 16.01.2020.

THE AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Antonov Alexander P.
Антонов Александр Петрович
E-mail: antonov@eda-lab.ftk.spbstu.ru

Besedin Denis S.
Беседин Денис Сергеевич
E-mail: 1310nero@mail.ru

Filippov Alexey S.
Филиппов Алексей Семенович
E-mail: alexey.s.filippov@gmail.com