ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2019 Управление, вычислительная техника и информатика № 48

УДК 004.272
DOI: 10.17223/19988605/48/8

A.Y. Filimonov, V.N. Trishin

MODEL-ORIENTED CO-DESIGN OF HOMOGENEOUS COMPUTING SYSTEMS

The paper was carried out with the partial funding of the scientific project of UB RAS No. 15-7-1-20 "Complex study of the architectures of 2D processor arrays that perform fine-grained processing parallelism and fast algorithms for processing speckled images to create VLSI video processors for real-time systems."

This article proposes an approach to the construction of a technological platform for designing function-oriented processors (FOP) built on the basis of homogeneous computing environments (OVS). A feature of the proposed approach is the use of co-design - a simultaneous and interconnected design of hardware (HW) and software (SW) FOP components. The paper presents proposals for the implementation of this approach, based on the use of the principle of model-based design, which allows you to make the co-design process iterative and effectively distribute data processing procedures between HW and SW - components of FOP. Based on the consideration of the experience in implementing such solutions, the paper presents the composition of functional subsystems of technological platforms, a diagram of information interaction between these subsystems based on intensive using of web-technologies at all stages of the design and operation of FOP.

Keywords: function-oriented processors; homogeneous computing systems; model-oriented design; parallel programming.

The current stage of the information systems development is characterized by an exponential growing amount of IoT (Internet of Things) connections and the volume of data they transfer increasing. According to forecasts, by 2020, the number of "smart things" connected to the Internet will be more than 20 times greater than human population [1], while the total amount of stored and processed data will reach $40 \cdot 10^{10}$ B [2]. The sharp increase in the amount of data generated by various IoT sensors led to the emergence of the concept of "fog computing", the main idea of which is to move the data preprocessing processes as close as possible to the points where these data were generated [3]. All these changes have a significant impact on the development of software and hardware platforms for real-time systems (RTEs), making such solutions urgent that provide scalability of the architectures of embedded computing subsystems, high degree of fault tolerance, minimization of power consumption, providing the maximum possible performance with given limitations on hardware costs [4]. This is most clearly manifested, for example, in such a class of RTE, when the embedded processor should provide a performance of the order of gigaflops with a mass of not more than 10 grams [5]. Experts note the need to conduct research and development of computing systems with mass parallelism, in particular, systolic architectures, which potentially realize the maximum possible performance when processing large amounts of data in real time at a new level of development of microelectronics [6]. Recently there are more and more examples of successful implementation of homogeneous components of computer systems [7]. The manufacturability of homogeneous computing systems (HCS) opens up wide prospects for the use of functionally-oriented processors (FOPs) based on them in various mobile systems, where high computational performance, low power consumption and resistance to failures are required. The programmability of the HCS architecture determines the variability in the implementation of hardware (HW) and software (SW) types of FOP, which requires the implementation of optimization procedures at all stages of its construction. Procedures for simultaneous and mutually conditioned design - co-design - make it possible to design of HW and SW HCS-FOP in such a way that the SW takes into account the available HW capabilities and vice versa, making changes to the HW project that would maximize the implementation of requirements for SW. Model-Driven Engineering (MDE) approach makes the design process of HCS-FOP iterative,

allows efficient distribution of data processing procedures between HW and SW, and the step-by-step verification of design solutions ensures the progressive nature of the design process and the achievement of predictable performance characteristics of the HCS-FOP, which is especially important when using such kind of devices in the IoT sphere [8]. In this paper considered an approach to the creation of a technological platform that is capable of ensuring the fulfillment of all stages of design, production, and operation of an HCS- FOP.

1. Main features of the HCS design process

A homogeneous computing system is a computer system consisting of identically connected identical processor elements (PEs), each of them is programmatically configured to perform an arithmetic or logical function, as well as to implement a data exchange with neighboring elements [9]. Programming HCS is a process of mutually coordinated adjustment of each of the PE array to perform corresponding processing and data transfer operations, which are necessary for mapping the data flow graph of the target task into the PE grid. By this process, the data stream graph of the programmable function must be transformed into a lattice graph [10] in a two-dimensional PE space, taking into account the design limitations of a particular implementation of the HCS [9]. Since the result of programming the HCS is the architecture of the computer, so the main feature of the design process for HCS is the need for its simultaneous execution with the programming process. The process of designing the HCS is designed to ensure the construction of a computer system that satisfies the set of algorithmic, constructive and operational requirements presented to it [11]. For HCS, which have to be operated in straightly limited conditions, it is very important to observe operational and design requirements, such as device's overall dimensions and its power consumption [9].

In addition to the requirements listed above, HCS that are designed to pre-process IoT sensors (Pre-processing level, level 3 of the fog computing architecture [12]) must meet the requirements for the components of the OpenFog Consortium Architecture [3]. The advantage of using HCS FOP at this level of fog computing architecture is that such systems are able to work indefinitely for a long time, implementing the principle of controlled degradation, which in this case can be provided by remote programming of HCS directly during operation [9]. In order to take advantage of this, it should be possible to interact with the design platform, which significantly limits the possibility of using local applications for its implementation [8].The platform for designing computer systems needed to fulfill similar requirements and constraints are presented should provide the ability to predict (monitor) the degree to which these requirements are met during the design process and should have appropriate mechanisms to manage this process. The conducted researches show that application of the model-oriented approach in the design of embedded systems allows to significantly improve the design quality [13]. The main idea of this approach is to apply the procedures for sequential transformation of the model of the projected system in such a way that at each subsequent stage this model begins to correspond to a larger list of requirements for the system being designed than the model of the previous stage [14]. Thanks to the automatic or automated generation of such models, the use of multi-level verification becomes possible, which allows to identify and eliminate errors in the early design stages [15]. That is why it seems particularly expedient to apply the model-oriented approach to the co-design of the HCS fop, which in this case involves the construction of the following models:

– computational model that allows you to determine the requirements for the accuracy of data representation and the error in solving the target task;

– architectural model, which allows to determine the degree of fulfillment of design requirements;

– complex model, which allows to determine the degree of implementation of the entire set of requirements for HCS.

In case of using the model-oriented approach, the technological cycle of co-design of HCS FOP begins with the construction of a computational model of the task (task) and represents a sequence of stages of transformation of this model up to the stage of creating a complete set of design documentation necessary for production and programming of the computer system.

2. Construction of HCS FOP computational model

The traditional approach to constructing computational models based on the use of block diagrams and programs in imperative programming languages allows one to adequately reflect the processes typical for computers with von Neumann architecture, exactly because this approach was especially created to describe computational processes that are controlled by the command flow (Control Flow).As the consequence, using of this approach to describe processes in computing systems that have an unconventional architecture, in particular systems that are controlled by Data Flow, causes significant problems [16]. The use of graphical tools helps to overcome some of those problems, because it allows to adequately, concisely and visually display the structure and interaction of information or control flows, as evidenced by the using of graphic and visual methods among domestic [17] and foreign [18] developers of programming platforms for systems with mass parallelism calculations. Considering capability of using such solutions for constructing the computational model of HCS, however, it should be kept in mind that the technology of graph-symbol programming (GSP) [17] is designed to describe the interaction of distributed computing systems through shared memory areas, and therefore, first of all, is directed on constructing the of control flows graph, but not data flow graph. At the same time, some of principles and concepts underlying the GSP platform, for example, such as the polymorphism of the base modules (computable functions) and the certification of "actors", appear to be quite universal and can be used productively in the creation of the HCS programming subsystem. Much closer to the HCS programming platform is the GASPARD (Graphical Array Specification for Parallel and Distributed Computing) platform, which provides a solution to a set of tasks that arise during the simultaneous design of real-time embedded (Real-time / Embedded-RT / E) hardware and software. The GASPARD platform is based on the concept of Model-Driven Engineering (MDE), which, in accordance with the basic requirements of the object management group (OMG) presented in Recommendation [19], allows to create a hierarchies of interrelated models of software and hardware components of RT / E systems real time. The basic language used to build the model of the computational process in GASPARD is the graphical functional language Array-OL [20]. The functional paradigm of the language in this case can be explained by the fact that the use of imperative languages for programming systems with the architecture controlled by data flows causes serious problems associated with side effects such as data racings and data availability limitations [21]. That is why to ensure the expressiveness and adequacy of describing the computational process in streaming computing systems with massive parallelism, programming platforms based on imperative languages are compelled to adopt the basic principles of the declarative programming concept [22, 23]. In this case, the process of solving the target task (TT) can be represented in the form of a hierarchical composition of functional modules (FM), each of which generates a stream of output data, taking as parameters the output data streams of the lower hierarchy level or the target task input data streams. If the FM will be determined parametrically polymorphic, special procedures or constructions must be presented in the system that provide control and agreement of the types of actual parameters of the FM. The process of applying a scalar (or incomplete) function to a multidimensional flow of input data, in declarative programming systems, have special importance since this process forms the basis for parallelizing computations.

For example, in Wolfram Language (WL) [24], some of the scalar functions can be applied to multidimensional arguments. A somewhat different approach presented in SequenceL language, where the mechanisms of implicit parallelization of computations are also actively applied [25]. The developers of SequenceL, however, recognize that the use of a fully automatic approach when matching the dimension of the operands can lead to errors localization difficulties [26]. More promising for implementation in the system of programming of HCS seems to be an automated procedure for structural matching, which can be performed by selecting the appropriate data stream structures when calling a specific function in the composition.

Thus, for constructing a computational model of the FOS HCS, it seems expedient to use one of the modern functional or hybrid programming languages that allows describing the TT in the form of a hierarchical composition of functional modules. Additional advantages in this case are the use of symbolic

programming languages (LISP, WL) because it allows the formation of a functional connection graph, which is necessary for performing the function stacking in the HCS matrix.

3. Realization of technological cycle of co-design of HCS

As a basic component of the HCS co-design platform, a subsystem for programming data processing functions (a programming subsystem), an architecture layout subsystem (a layout subsystem), and a simulation subsystem (an emulation subsystem) are proposed in the work. The programming subsystem provides the transformation of a computational model into an architectural model, the layout subsystem transforms the architectural model into a complex one, the emulation subsystem provides a comprehensive verification of selected for implementation the software and hardware solutions of the FOS HCS. To ensure the optimization and adaptation of the emerging HCS architecture, the process of information interaction between subsystems have to be iterative and interactive one. The subsystems themselves are built according to the block-modular approach in order to ensure the possibility of further development of the programming platform. The operation of the HCS co-design subsystem begins after the computation model of the target task is defined, represented by the graph of the function blocks connected by the data streams. The programming subsystem is designed to map the TT algorithm to the operational basis of the HCS and the corresponding distribution of data processing functions between the PE groups. To provide debugging for this process control data of the task function (CDF) have to be used, for example, to provide control the accuracy of calculations. As an interface tool of the programming subsystem, in this case it is advisable to use the additional software "electronic notebook" - (EN) [27]. In modern programming systems, EN provides functions such as editing, formatting, text saving, on-line monitoring of syntax structures (including syntax-highlighting) [28] and context-sensitive support for the programming process [24]. The use of EN as the interface and binding element of the programming platform for IOT becomes even more useful since it allows to take into account all the programming features inherent in this platform. After debugging process completion, the programming system generates a complete data flow graph, which determine needed operations for every PE. The layout subsystem is designed for laying a complete graph of data flows of the TT in the HCS lattice and ensures the placement of programmed PEs on the HCS matrix and creation of links between them. In fulfilling this task, the construction limitations inherent in the specific implementation of the HCS (CL) should be ensured. In general, these requirements may limit:

- the total number of PEs that can be used to solve the TT;
- geometric parameters of the region where these PEs can be located;
- the mutual position of PE groups that are involved in the implementation of interrelated functions.

The final solution of the layout problem is the one of possible mappings decisions a complete graph of data streams in the PE matrix, which ensures that all CLs are fulfilled in condition of reaching required value of the predicted decision time of the TT. This solution will be formulated in the technological language of the styling specifications, which is the final result of the processes of programming and design of the HCS. The simulation subsystem is designed to simulate the implementation of the prepared decision of the TT on the HCS and provides final control over the implementation of the requirements for the implementation of this TT.

Figure 1 presents the composition of the components of the HCS co-design platform and the scheme of information interaction between them. Blocks 1.1-3.2 form the programming subsystem, block 4 corresponds to the layout subsystem, block 5 represents the simulation subsystem, and the remaining blocks provide an iterative and interactive mode of interaction between the main platform subsystems. This is especially important for responsible applications, for example, processing signal flows from mobile sensor outputs robot for the purpose of prompt decision making in the regime of hard real time from the command post at a considerable distance from the control direct object

An example of the implementation of the web interface of the simulation subsystem is shown in Fig. 2. In response to the request, the client's browser receives a page of the cloud-based integrated development environment (IDE), in which the PE link graph is edited. The PEs themselves are displayed using built-in SVG elements, which are a copy of the librarian one.

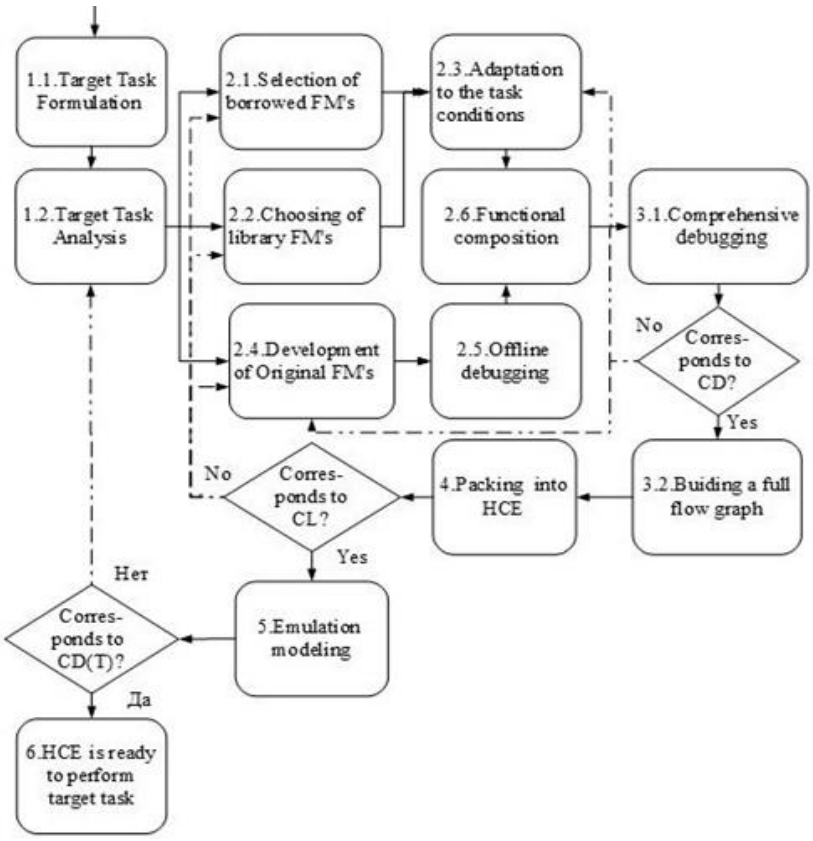

Fig. 1. Scheme of information interaction of the HCS co-design platform components

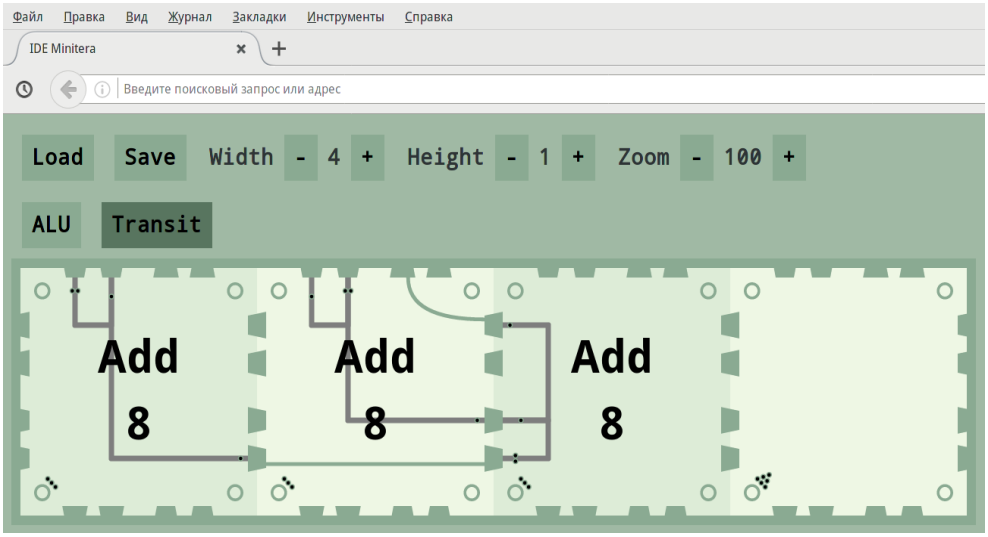

Fig. 2. Implementing the web interface of the simulation subsystem

The visibility of the symbols of the operating mode of the PE is adjusted using the CSS style sheet. On the client side, a JavaScript code is executed that passes data about the user's action to the server and receives new rules for the style sheet in response. On the server side, Perl scripts are accessed through SQL queries to the database, which stores data about the states of processor elements and user sessions. Since the modern concept of managing network components involves the use of the NETCONF or RESTCONF protocols [29] to manage device configurations, as a promising direction for the further development of the proposed approach should be the integration into the HCS co-design platform of the communication model in the YANG language [30].

Conclusion

Currently, the problem of effective HCS programming still not solved, and the main reason for this is the lack of developed and mature programming and design technologies for this computing architectures. In this paper the basic principles of construction of a technological design platform which allows to unite in a single process and automate all stages of co-design of HCS are considered. The key components of this platform are the functional programming subsystem, the information algorithm stacker on the two-dimensional PE array (the layout subsystem), and the emulator (simulation subsystem). The platform assumes the use of cloud services at all stages of development and development of HCS programs. As the first stage of the work on the creation of the technological platform for HCS programming, an intelligent graphical editor has been developed that allows verifying the description of the information flow graph and automating the procedures for laying software projects in the cellular space of the HCS.

REFERENCES

1. CompTIA. (2015) *Sizing Up the Internet of Things*. [Online] Available from: https://www.comptia.org/resources/sizing-up-the-internet-of-things (Accessed: 22nd May 2018).
2. Audin, G. (2015) *The Network Impact of Big Data*. [Online] Available from: https://www.nojitter.com/post/240170228/the-network-impact-of-big-data (Accessed: 22nd May 2018).
3. OpenFog Consortium Architecture Working Group. (2016) *White Paper.OpenFog Architecture Overview*. [Online] Available from: https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture- Overview- WP-2-2016.pdf (Accessed: 22nd May 2018).
4. Chiang, M. & Zhang, T. (2016) Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*. 3(6). pp. 854–864. DOI: 10.1109/JIOT.2016.2584538
5. Kuhn, A. (2014) *Review of Novel Computing Architectures for Neural Applications*. [Online] Available from: https://pdfs.semanticscholar.org/fab9/f4e8834900c0f66d10d7d0e8b786915c7271.pdf (Accessed: 22nd May 2018).
6. Master, P. (n.d.) *A new computing architecture for Big Data and AI applications*. [Online] Available from: http:// impactvc.com/a-new-computing-architecture-for-big-data-and-ai-applications (Accessed: 22nd May 2018).
7. Jouppi, N.J. et al. (2017) In - Datacenter Performance Analysis of a Tensor Processing Unit. *Proc. of the 44th Annual International Symposium on Computer Architecture, ISCA*. pp. 1–12. DOI: 10.1145/3140659.3080246
8. Lukin, N.A, Filimonov, A.Yu. & Trishin, V.N. (2017) Cloud computing environment of homogeneous computing systems. In: Dubrov, D.V. *Programming languages and compilers – 2017*. Rostov-on-Don: [s.n.]. pp. 181–184. (In Russian).
9. Lukin, N.A. (2014) Functionally-oriented processors are key components of built-in supercomputers for real-time systems. *Izvestiya Yuzhnogo federal'nogo universiteta. Tekhnicheskie nauki – Izvestiya SFEDU. Engineering Sciences*. 12. pp. 52–64. (In Russian).
10. Voevodin, V.V. (1986) *Mathematical Models and Methods in Parallel Processes*. Moscow: Nauka.
11. Lukin, N.A. (2010) [Functionally oriented processors with a homogeneous architecture for the implementation of algorithms of onboard control systems]. *Parallel Computing and Control Problems*. Proc. of the 5th International Conference. Moscow. pp. 1177–1184. (In Russian).
12. Atlam, H.F., Walters, R.J. & Wills, G.B. (2018) Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing*. 2(2). DOI: 10.3390/bdcc2020010.
13. Gamatié, A., Le Beux, S., Piel, É., Ben Atitallah, R., Etien, A., Marquet, P. & Dekeyser, J.-L. (2011) A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embed. Comput.Syst*. 10(4). Article 3936. DOI: 10.1145/2043662.2043663
14. Adler, R., Schaefer, I., Schuele, T. & Vecchie, E. (2007) From Model-Based Design to Formal Verification of Adaptive Embedded Systems. *ICFEM 2007, LNCS 4789*. pp. 76–95. DOI: 10.1007/978-3-540-76650-6_6
15. Rovers, K.C., van de Burgwal, M.D., Kuper, J., Kokkeler, A.B.J. & Smit, G.J.M. (2011) Multi-domain transformational design flow for embedded systems. *International Conference on Embedded Computer Systems (SAMOS 2011)*. pp. 93–101. DOI: 10.1109/SAMOS.2011.6045449
16. Lukin, N.A. & Filimonov, A.Y. (2017) Software technologies for homogeneous computing environment. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 40. pp. 61–70. (In Russian). DOI: 10.17223/19988605/40/7
17. Kovartsev, A.N. & Zhidchenko, V.V. (2011) *Methods and means of visual parallel programming. Automation of programming*. Samara: Samara State Aerospace University.
18. Devin, F., Boulet, P., Dekeyser, J-L. & Marquet, P. (2002) GASPARD a visual parallel programming environment. *Parallel Computing in Electrical Engineering, PARELEC '02.* Proc. International Conference. DOI: 10.1109/PCEE.2002.1115225

19. Object Management Group, Inc. (n.d.) *UML Profilefor MARTETM: Modeling and Analysis of Real-time Embedded Systems*. [Online] Available from: http ://www.omg.org/spec/MARTE/1.1 (Accessed: 22nd May 2018).
20. Boulet, P. (2008) Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing. *Research Report RR-6467, INRIA.*
21. Johnston, W.M., Hanna, P. & Millar, R.J. (2004) Advances in dataflow programming languages. *ACM Computing Surveys (CSUR)*. 36(1). pp. 1–34. DOI: 10.1145/1013208.1013209
22. Böhm, A.P.W., Hammes, J., Draper, B.A., Chawathe, M., Ross, C., Rinker, R. & Najjar, W. (2002) Mapping a Single Assignment Programming Language to Reconfigurable Systems. *Supercomputing*. 21. pp. 117–130. DOI: 10.1023/A:1013623303037
23. Hammarlund, P. & Lisper, B. (1993) Data Parallel Programming: A Survey and a Proposal for a New Model. *Technical Report 93/8-SE, Department of Teleinformatics Royal Institute of Technology, Sweden*.
24. Wolfram, S. (2017) *An Elementary Introduction to the Wolfram Language*. 2nd ed. Wolfram Media.
25. Nemanich, B., Cooke, D. & Rushton, N. (2010) Sequencel: transparency and multicore parallelisms. *DAMP '10 Proceedings of the 5th ACM SIGPLAN workshop on Declarative Aspects of Multicore Programming*.
26. Cooke, D., Rushton, N., Nemanich, B. & Watson, R.G. (2008) Normalize, transpose, and distribute: An automatic approach for handling nonscalars. *ACM Transactions on Programming Languages and Systems*. 30(2). DOI: 10.1145/1330017.1330020
27. Project Jupyter. (n.d.) *Jupyter Notebook Documentation Release 5.1*. [Online] Available from: https://media.readthedocs.org/pdf/jupyter-notebook/latest/jupyternotebook.pdf. (Accessed: 22nd May 2018).
28. The IPython Development Team. (n.d.) *IPython Documentation Release 6.2.0*. [Online] Available from: http://ipython.readthedocs.io/ en/latest/whatsnew/development.html. (Accessed: 22nd May 2018).
29. Xu, H., Wang, C., Liu, W. & Chen, H. (2012) NETCONF-based Integrated Management for Internet of Things using RESTful Web Services. *International Journal of Future Generation Communication and Networking*. 5(3). pp. 73–82.
30. Scheffler, T. & Bonness, O. (2017) Manage Resource-constrained IoT Devices Through Dynamically Generated and Deployed YANG Models. *Proc. of the Applied Networking Research Workshop*. 1. pp. 42–47. DOI: 10.1145/3106328.3106331

Received: December 6, 2018

Filimonov A.Y., Trishin V.N. (2019) MODEL-ORIENTED CO-DESIGN OF HOMOGENEOUS COMPUTING SYSTEMS Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie vychislitelnaja tehnika i informatika [Tomsk State University Journal of Control and Computer Science]. 48. pp. 67–73

DOI: 10.17223/19988605/48/8

Филимонов А.Ю., Тришин В.Н. МОДЕЛЬНО-ОРИЕНТИРОВАННЫЙ КО-ДИЗАЙН ОДНОРОДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕД. *Вестник Томского государственного университета. Управление, вычислительная техника и информатика.* 2019. № 48. С. 67–73

В работе предложен подход к построению технологической платформы проектирования функционально-ориентированных процессоров (ФОП), построенных на основе однородных вычислительных сред (ОВС). Особенностью предлагаемого подхода является использование ко-дизайна – одновременного и взаимосвязанного проектирования аппаратного (HW) и программного (SW) обеспечения таких ФОП. В работе приведены предложения по реализации данного подхода, основанные на использовании принципа модельно-ориентированного проектирования, который позволяет сделать процесс ко-дизайна итеративным и эффективно распределить процедуры обработки данных между HW- и SW-компонентами ФОП. На основе рассмотрения опыта реализации подобных решений в работе представлен состав функциональных подсистем технологических платформ, приведена схема информационного взаимодействия между этими подсистемами, предполагающая интенсивное применение web-технологий на всех стадиях проектирования и эксплуатации ФОП.

Ключевые слова: однородные вычислительные системы; модельно-ориентированный дизайн; параллельное программирование.

FILIMONOV Alexander Yurievich (Associate Professor of the Department of Information Technologies and Automation in the Institute of Radioelectronics and Information Technologies at Ural Federal University, Yekaterinburg, Russian Federation).
E-mail: a.filimonov@urfu.ru

TRISHIN Vasily Nikolaevich (Junior Researcher of the Institute of Engineering Science, Ural Branch of the Russian Academy of Sciences, Yekaterinburg, Russian Federation).
E-mail: trishinvn@yandex.ru