Software Development Technology for Homogeneous Computing Environments

Vasily Trishin[1], Nick Lookin[2], and Alexander Filimonov[2]

[1] Institute of Engineering Science Ural Branch of RAS, Yekaterinburg, Russia
[2] Ural Federal University, Yekaterinburg, Russia
nanocomputerlab@gmail.com

Abstract. Homogeneous computing environments (HCE) for many years are of interest to researchers and designers of high-performance computing. The uniformity of the HCE architecture allows it to increase its computing capabilities by a simple increase in the number of processor elements. The possibility of power consumption control making the HCE very attractive for the implementation of the Internet of Things (IoT). The unconventionality of the HCE architecture prevents the use of traditional technologies for its programming. The main reason is that the HCE is a computer system with stored algorithms for solving the target problem. A technology platform for programming HCE, based on three parts (programming, composing and emulator) is discussed. Examples of the developed elements of the proposed technology of HCE programming are given.

Keywords: homogeneous computing environment · mass parallelism · declarative programming · technological software platform · cloud programming

1 Introduction

One of the main trends in the development of software and hardware platforms for real-time systems (RTS) is the scalability of the architectures of embedded computing subsystems, high degree of fault tolerance, minimization of power consumption, ensuring the maximum possible performance with given restrictions on hardware costs. This is most clearly presented, for example, in such a class of RTS as the Internet of things (IoT), when the built-in processor should provide performance of the order of gigaflops with a mass of not more than 10 grams [13]. Specialists note the need for a new level of development of microelectronics to carry out research and development of computer systems with mass parallelism, in particular, systolic architectures that potentially realize the maximum possible performance when processing large amounts of data in real time [14]. The development of systolic processors and systems based on them has revealed a number of problems. One of the most actual problems is the practical lack of software technologies that allow developing, debugging and verifying user programs on the systolic architectures. At present, projects of systolic processors or systems do not contain the same advanced solutions in the field of programming technologies, as processors based on the concept of von Neumann or its development. One of the main reasons for the difficulties in developing software for systolic processors is the unresolved fundamental problem of the mutual mapping of algorithms and architectures. The peculiarities of the topology of processor element arrays not only require transformations of mathematical procedures, but also give the algorithms themselves a distinct “geometric” character. For example, in a number of cases it is required to transform arbitrary algorithmic graphs into so-called lattice graphs [21], which is not always possible. As another reason, it should be noted the explicit declarative character of the description of data processing processes in systolic architectures, for which the fulfillment of the i-th transformation is determined by the readiness of the results of some j-th. This is a typical case of a computer system with mass parallelism and data flow control. Below we consider a possible approach to the development of software technologies for systolic architectures of a certain type.

2 Homogeneous Computing Environment

Homogeneous computing environments are specialized computing systems consisting of regularly connected identical processor elements (PE), each of which is configured to perform an arithmetic or logical function, and to implement an exchange protocol with neighboring elements [18], [20]. Features of the organization of the computational process in HCE make it expedient to use them as special-purpose computers or co-processors for solving a wide, but fixed class of computational tasks. There are some features for these tasks, they are:

- Large arrays of data being processed;
- Non-standard forms of data representation;
- Data processing in hard real-time mode (limits for the time interval or periodicity of for the results of computations);
- Acyclic algorithmic graphs or minimum number of cycles according to the data.

Examples of such tasks are image processing in real time, pattern recognition and classification, the use of modern encryption algorithms in on-fly mode etc.

2.1 Formalism for HCE

Let us have an arbitrary computer system (CS) S = {{PE },{L}} {PE} - PE set, {L} - set of interconnections. This corresponds to the representation of the CS in the form of an undirected graph G_{HCE}, where PE is the vertex, and L is the edge. We introduce the PE numbering, assuming that they form a two-dimensional array.Then

$$S = \{\{PE_{i,j}\}, \{L_{(i\pm\alpha, j\pm\beta)}\}\}, \tag{1}$$

where $\alpha \in$[0,(N-1)],$\beta \in$[0,(M-1)].Representation (1) corresponds to different structures of the CS. We call homogeneous CS, for which all PE$_{i,j}$ are identical, and all connections are regular. Only the regularity of the constraints can lead to the ambiguity of the CS representation. For example, a fully connected graph G$_{HCE}$ is regular by definition [7], at the same time 2D-regular graph can have intersecting edges. The closest single-valued interpretation of HCE are the lattice graphs of the algorithms [21]. In this paper, these graphs are useful for the description of problems. Lattice graphs can also contain intersections of arcs, for example, Hamming graphs [3]. Let us introduce some definitions. The homogeneity of the first type for the graph G$_{HCE}$ is its regularity:

$$\forall i,j,\alpha,\beta; degPE_{i,j} = degPE_{(i\pm\alpha,j\pm\beta)}, \tag{2}$$

where degPE$_{i,j}$- degree of the vertex PE$_{i,j}$ The homogeneity of the second type for the graph G$_{HCE}$ is:

$$\forall i,j,\alpha,\beta; [L_{(PE_{i,j}),(PE_{i,j\pm1})} = L_{(PE_{i,j}),(PE_{i\pm1,j})} = 1] \& [L_{(PE_{i,j}),(PE_{i\pm1,j\pm1})} \neq 1] \tag{3}$$

where L$_{(PE_{i,j}),(PE_{i\pm\alpha,j\pm\beta})}$ - the length of the path (route) from the vertex PE$_{i,j}$ to the vertex $PE_{i\pm\alpha,j\pm\beta}$. A homogeneous graph G$_{HCE}$ is a graph for which simultaneous fulfillment of conditions (2) and (3) is necessary and sufficient. Hence it follows that G$_{HCE}$ is a lattice graph of unit distances, regular, the degrees of all its vertices are equal to 4.

Formalism for PE. Each PE$_{i,j}$, is a finite state machine of the following form (Fig. 1)

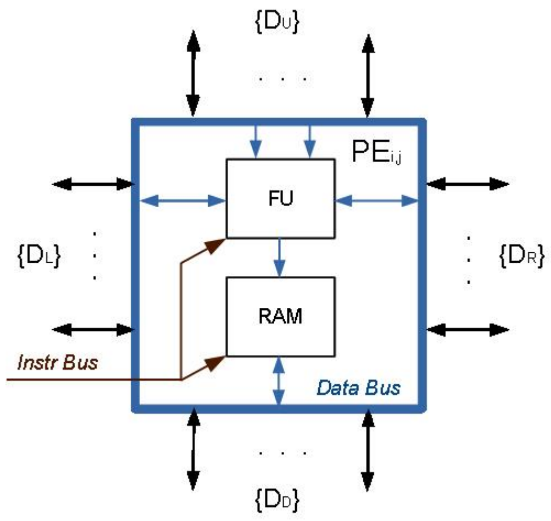

Fig. 1. HCE PE

Notations in the Fig.1: {D_U};{D_D};{D_L};{D_R }; -external data bus; U::= $\langle U_0, U_1, \ldots, U_{N_{U-1}}\rangle$;D::= $\langle D_0, D_1, \ldots, D_{N_{D-1}}\rangle$L::= $\langle L_0, L_1, \ldots, L_{N_{L-1}}\rangle$;R::= $\langle R_0, R_1, \ldots, R_{N_{R-1}}\rangle$.

For HCE considered above:N_{U-1}=N_{D-1}=N_{L-1}=N_{R-1}. Each PE can execute a finite set of microinstructions of two types:

– Microinstruction of data processing (EXE):
  {OUT} := {IN1}(OPCODE){IN2}; IN1,IN2,OUT = $\langle D_k\rangle \vee \langle RAM\rangle$;k = U,L,R,D;
– Microinstruction of transit (TRANSIT):
  {OUT} := {IN};OUT,IN = $\langle D\rangle \vee \langle RAM\rangle$;

These two types of microinstructions are combined into instruction (INSTR), that performs per one cycle: {INSTR} ::= [C1&EXE]$\vee$[C2&TRANSIT], C1, C2 — control signals. The number of instruction types is determined by the operation code. Each PE can realize 3 modes of data processing – EXE mode, TRANSIT mode or their joint execution.

All HCE PE perform INSTR cycle by cycle, operation codes are pre-recordedin the control memory. So the natural metric of any data processing procedure is the time complexity of the computations [22]. There are two kinds of time complexity in HCE case, they are

– autonomous time complexity $L_{t,a}$= n_d , [cycle], where n_d- number of cycles necessary to execute the d-th command (instruction);
– pipeline complexity of computations $L_{t,c}$=1;

The design of the programming languages of HCE should take into account the peculiarities of the temporal organization of data processing, in particular, the organization of pipelines.

Before data processing, code “INSTR” is written to each PE. Thus, HCE is a computer system with a stored algorithm. HCE programming is a declaration $INSTR_{i,j}$ for each PE to implement the algorithm for solving the target task.

Then HCE = {$PE_{i,j}$}; i = 0,1,...,(N-1); j = 0,1,...,(M-1). The original algorithm (or more generally TASK) is TASK = {$INSTR_{p,q}$}; p = 0,1,...,(P-1), P$\leq$(N-1);q = 0,1,...,(Q-1), Q$\leq$(M-1).Both HCE and TASK are determined by their graphs. For HCE graph: G_{HCE} = {{$PE_{i,j}$}};{$L_{i,j}$}}, for algorithm graph: G_{TASK} = {{$INSTR_{p,q}$}};{$(L_{TASK})_{p,q}$}}, for HCE with embedded program: $(G_{HCE})_{instr}$ = {{$PE_{p,q}$}};{$L_{p,q}$}}.It follows from the above that there exists a homomorphism on the graph G_{TASK} on the graph (GHCE)instr, i.e. graph $(G_{HCE})_{instr}$ is subgraph of the graph G_{HCE}.

HCE programming is the declaration procedure of each {$PE_{i,j}$}; i = 0,1,...,(N-1); j = 0,1,...,(M-1) destination on execution of the {$INSTR_{p,q}$}; p = 0,1,...,(P-1), P$\leq$(N-1);q = 0,1,...,(Q-1), Q$\leq$(M-1) with the aim of solving the TASK. All PE are connected to the lattice, which eliminates the need to specify the relationship between the algorithms blocks. In this case, the program should be called the recording of the function blocks of the target task.

3 Features of HCE programming

The programming of the HCE is the process of setting up each PE for the execution of the corresponding processing and transmission operations that are necessary to display the data flow graph of the target task into the PE lattice. The presence of a spatial metric (rectangular matrix of locally connected PEs) causes the "geometric" features of programming, which determines the necessity of constructing the propagation paths of data streams in two-dimensional PE space. Programmers need taking into account the constructive limitations of a particular implementation of the HCE. The result of HCE programming is the special-purpose processor architecture that determines the distribution of operations and the interconnections between PEs in contrast to the programming of the universal processor, where the result is a program for the same architecture. The main feature of the HCE programming is the simultaneous development of a program and architecture that implements it (co-design). As the main components of the HCE programming platform, the subsystem for programming data processing functions (programming subsystem), the architecture layout subsystem (layout subsystem) and the simulation modeling subsystem (simulation subsystem) are proposed in the work. To optimizing the architectures of the HCE, the interconnection between subsystems should be iterative and interactive. These subsystems must be block-modular in order to ensure the possibility of further development of the programming platform. The programming subsystem is designed to display the algorithm in the operational basis of the HCE and the corresponding distribution of the data processing functions between the PE groups. The requirements for the ranges of change and accuracy of the representation of the elements that form the input, output and intermediate data streams should be formed and monitored. For debug the mapping of the algorithms into the operational basis of the HCE, the verification data for the functions of the target task are used (for example, for control the computation accuracy). If the debugging succeeds, the programming system generates a complete data flow graph, which is detailed up to PE's operation and the links between them. The layout subsystem provides the placement of programmed PEs on the HCE and forms links between them. It is designed for laying a complete graph of data streams of the initial algorithm in the HCE lattice. To successfully solve this task, the requirements of the specific implementation of the HCE (CT) must be met. These requirements include:

- Total number of PEs that can be used.
- Geometric parameters of the space where PE can be placed.
- Mutual position of PE groups necessary to execute the d-th command (instruction).

The final solution to the layout problem is the option of laying a complete graph of data flows into HCE, which ensures that all the requirements listed above. This solution is formulated in the technological language of the layout specifications. The simulation subsystem is designed to simulate the implementation of the algorithms on the HCE and provides final control over the fulfillment of the all requirements.

4 Principles of Constructing the Platform for Programming HCE

To take into account the features of the data processing organization in mass-parallel CS, the developers of programming systems use non-traditional forms of representation of problems (tasks). The use of graphic tools allows display the structure of information or control flows adequately, concisely and visually. The emergence and dissemination of graphic and visual methods among Russian [12] and foreign [8] developers of programming platforms for systems with mass parallel computing was natural. Technology of the graph-symbol programming (GSP) [12] is designed to describe the interaction of distributed computing systems through shared memory. It is oriented to the construction of the control flows graph. At the same time, concepts underlying the GSP platform, such as the polymorphism of basic modules (computable functions) and the certification of actors are universal and can be used in the development of the HCE programming subsystem. The GASPARD (Graphical Array Specification for Parallel and Distributed Computing) platform provides a solution to a set of tasks for hardware and software co-design for embedded real-time systems (Real-time / Embedded - RT/E). It is based on the concept of Model-Driven Engineering (MDE) in accordance with the basic requirements of the object management group (OMG) presented in the recommendation [16]. It allows you to develop a hierarchy of interrelated models of software and hardware components of RT/E real-time systems. The GASPARD system was implemented by the Inria development team from 2006 to 2011 as a set of plug-ins for the Eclipse platform and can be considered as a functional prototype of the HCE programming platform. The basic language used to build the model of the computational process in GASPARD is the graphical functional language Array-OL [5]. The use of imperative languages for programming Data Flow driven architecture causes serious problems associated with side racing effects and data availability limitations [11]. That is why programming platforms based on imperative languages are forced to adopt the basic principles of the declarative programming concept [4],[10]. The absence of common memory, variables, pointers, indices, cycles is an additional advantage of declarative languages for programming HCE. It is advisable to use a combination of textual and graphical description of the computational process with the automatic formation of a graph of functional relationships for the verification of the program model. In this case, the process of solving the target can be represented as a hierarchical composition of functional modules (FM), each of which generates a stream of output data. The output streams of the FM lower level of the hierarchy or the input data streams of the target task itself can act as FM parameters. There are some types of FM. They are:

– Initially defined in the programming system (built-in FM);

- Defined by other users (library FM);
- Defined by the user (original FM).

The transition from the static typing of lexical constructions, which is accepted in the Array-OL language to the dynamic one, in which the types of arguments are defined and controlled at the interpretation stage seems promise The violation of the dimension in WL is fixed only if the system fails to perform the specified function element-by-element. In this case, these FM can be determined as polymorphic parametrically. In the language, special procedures or constructions must be provided that ensure the monitoring and agreement of the types of actual parameters of the module. The use of a scalar (or incomplete) function to a multidimensional input data stream is important. It is important for declarative programming systems to be given since it forms the basis for parallelizing computations. For example, in Wolfram Language [19] (hereinafter WL) some of the scalar functions can be applied to multidimensional arguments. In this case, the structures of the streams of operands are automatically reconciled, which allows performing element-by-item arithmetic functions, for example, adding a scalar to a matrix, a vector with a matrix, and so on. The violation of the dimension in WL is fixed only if the system fails to perform the specified function element-by-element (for example, when two different sequences of different lengths are added). A different approach is practiced in SequenceL, in which the mechanisms of implicit parallelization of computations are also actively used [15]. If the dimension of the argument when accessing a function exceeds the value of the dimension specified in its definition, the SequenceL programming system automatically performs a set of matching operations. This is NTD (Normalize-Transpose-Distribute), which relieves the programmer of the need to use additional software to parallelize the computational process. The developers of SequenceL note that the use of a fully automatic approach when matching the dimension of the operands can lead to the appearance of difficult to diagnose errors [6]. The automatic procedure of structural matching, which is performed by selecting the adapters of data stream structures when calling a particular function, is more promising. Structural adapters are standard procedures for direct and inverse space-time transformations of data stream structures that can be defined for each type of HCE architecture. For example, the flow of matrices of dimension m × n that follow with period t can be transformed into a stream of vectors of dimension m that follow with a period t/n or a scalar stream that follow with a period of t/(m × n). For data stream elements, the number of bits, accuracy (error) of representation may be determinate. In addition, format adapters (standard procedures for forward and backward transformations of presentation formats) can be defined. Thus, it seems appropriate to use one of the modern functional or hybrid programming languages as a basic programming language. It allows describing the target task 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 forming a functional connection graph, which is necessary for performing the laying of FE on the HCE.

5 Implementation of the Programming Platform

Figure 2 shows the composition of the components of the HCE programming platform and the scheme of information interaction between them. Platform blocks 1.1-3.2 represent a programming subsystem, block 4 corresponds to a layout subsystem, and block 5 represents a simulation subsystem. The remaining blocks provide an iterative and interactive mode of the interplay between the main platform subsystems. Each of the intermediate stages of the programming

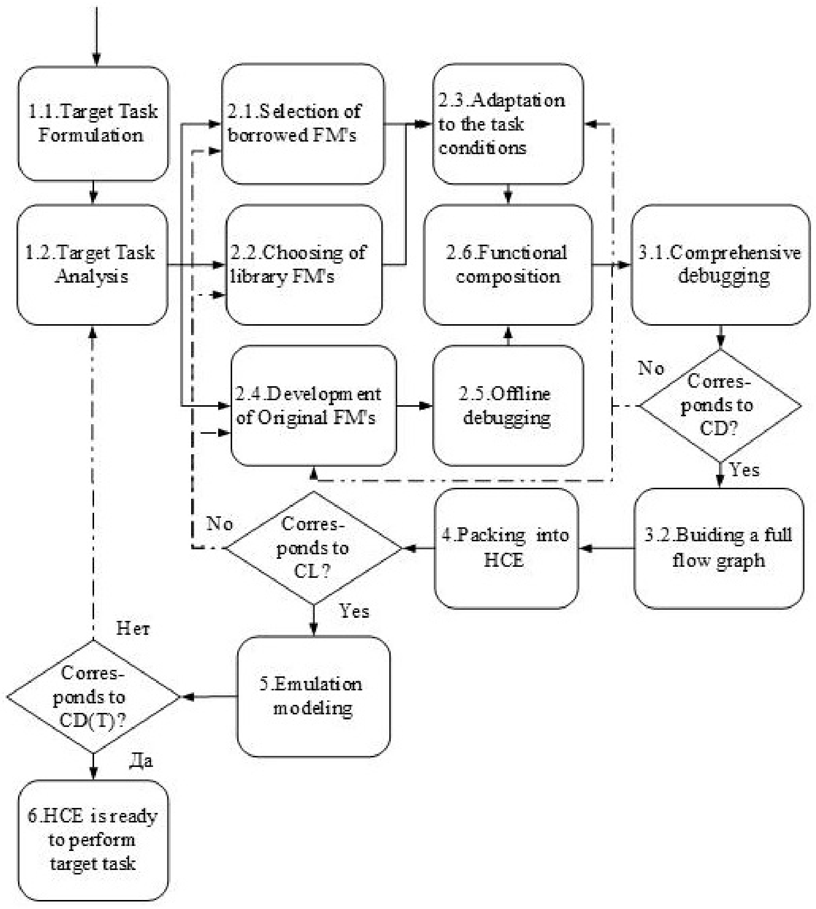
1.1.Target Task Formulation
1.2.Target Task Analysis
2.1.Selection of borrowed FM's
2.2.Choosing of library FM's
2.3.Adaptation to the task conditions
2.4.Development of Original FM's
2.5.Offline debugging
2.6.Functional composition
3.1.Comprehensive debugging
No
Corres-ponds to CD?
Yes
3.2.Buiding a full flow graph
4.Packing into HCE
No
Corres-ponds to CL?
Yes
5.Emulation modeling
Нет
Corres-ponds to CD(T)?
Да
6.HCE is ready to perform target task

Fig. 2. The technological platform for PCE programming

process of the HCE should be finished by the implementation of the control procedure. Successful execution of such a procedure indicates that the next stage can be performed; otherwise, the programming platform makes recommendations to the programmer to improve the re-execution of previous stages. The proposed concept of a technological platform for the development of application (embedded) HCE software largely corresponds to MDE ideology with the necessary additions in terms of developing and testing (verifying) software for real-time systems. The architecture features of the HCE requires consideration of the topology of the connections (grid) and the systolic principle of data processing. The layout of algorithmic graphs into PE's lattice represents s problem of the high combinatorial complexity. For the same algorithm, there are many options for laying its graph into the topology of a two-dimensional PE's cellular array. This requires the directional enumeration in order to optimize the data processing by the criteria of hardware or time complexity of computations. In addition, for mobile systems such as IoT, it is necessary to provide the ability to create and verify software projects remotely using cloud technologies. This is especially important for responsible applications, for example, processing signal flows from the outputs of mobile robot sensors in order to make promptly decisions in hard real-time mode from the command post at a considerable distance from the controlled object. The programming platform for IoT should take into account the specifics of the process programming of HCE and the current requirements that are imposed on the IoT [17] computing components. The features of programming HCE in the IoT structure significantly limit the possibility of using traditional programming systems that are built as local software applications. Cloud platforms, which are becoming more widespread, can significantly reduce the cost of creating software projects due to the ability to interact quickly with libraries of previously created software and hardware modules [23]. Full elimination or minimal use of local applications of the HCE ensures complete independence of the programming process from the characteristics of the local OS. This allows for further development of the programming platform and provides the opportunity to implement new approaches to the collective development of not only programs, but also algorithms [9]. In this case, it is advisable to use additional software "electronic notebook" as a programming interface - (electronic notebook-EN) [2]. In modern programming systems, EN provides some functions such as editing, formatting, text saving, on-line monitoring of syntax (including syntax highlighting) [1], context-defined support [19]. The use of EN as the interface and binding element of the IoT programming platform for IoT allows taking into account all the programming features inherent in this platform. At present, the graphic editor of the PE array has been developed; it is part of the simulation subsystem. By means of editor, the programmer assigns PE's executable instructions and builds the trajectory of the processed data, actually creating a graph $(\mathrm{G}_{HCE})_{instr}$. In Fig.3, as an example, the web-interface of the editor is displayed, it is a fragment of the field of PE, configured to perform tree-like summation of the variables arriving at the four input ports. The scheme of information interaction of the components of the simulation subsystem is shown in Fig.4. In response to the request, the client's browser receives the page of the cloud-based IDE. Graph $(\mathrm{G}_{HCE})_{instr}$ is edit on this page. The PEs are display using the built-in SVG elements, which are copies of the library ones. The images of the symbols of the PE's operating mode are tune using the CSS style sheet.

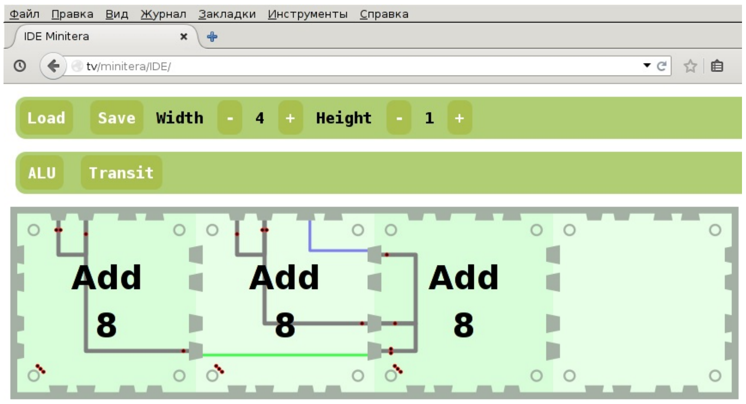

Fig. 3. The simulation subsystem's web-interface window

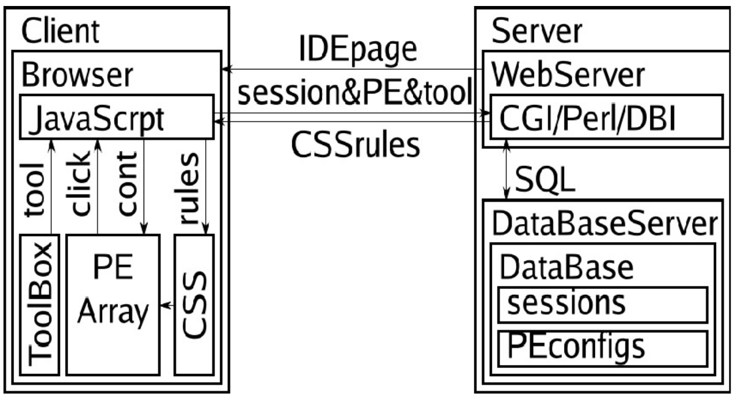

Fig. 4. Flowchart of data interconnection of the simulation subsystem

On the client side, JavaScript code is execute and the information about user's action is sent to the server and receives new rules for the style sheet in response. On the server side, Perl scripts are execute. These scripts access through SQL-queries to the database in which the data about the states of processor elements and user sessions are stored. Today the technology of remote programming of the HCE at the lower level has been developed, that is, using the graphical tools for projects design on the MTera 2 HCE. At this stage, software projects can be implement directly on user computers. The programming system is located on a remote server and its activation and operation is performed through the developed web-interface.

6 Conclusion

Homogeneous computing environments once again attract the attention of researchers and developers of high-performance computing systems in a wide range of applications: from co-processors of supercomputers to functional-oriented processors of real-time systems. Currently, the problem of effective HCE programming is still not solved, and the main reason for this is the lack of an equally developed technology for programming user tasks, both for universal processor architectures. In our paper, the basic principles of construction and functioning of technological platform of HCE programming are considered. The key components of this platform are the functional programming language (the program making subsystem), the information algorithm graph layer on the two-dimensional PE array (layout subsystem), and the emulator (simulation subsystem). The platform is based on the use of cloud services as part of a distributed software development environment. As the first stage of development of the technological platform for HCE programming, an intelligent graphic editor has been developed.

The work was carried out with the partial financing of the Project of the Ural Branch of the Russian Academy of Sciences No. 15-7-1-20 on the subject "Complex study of architectures of 2D processor arrays that carry out fine-grained processing parallelism and fast algorithms for processing speckled images to development VLSI video processors for real-time systems".

References

1. Jupyter and the future of IPython — IPython (2017), https://ipython.org/
2. Project Jupyter | Home (2017), http://jupyter.org/
3. Bailey, R.F., Cameron, P.J.: Base size, metric dimension and other invariants of groups and graphs. Bulletin of the London Mathematical Society 43(2), 209–242 (2011)
4. Böhm, W., et al.: Mapping a Single Assignment Programming Language to Reconfigurable Systems. The Journal of Supercomputing 21(2), 117–130 (2002)
5. Boulet, P.: Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing (2008), https://hal.archives-ouvertes.fr/inria-00261178
6. Cooke, D.E., Rushton, J.N., Nemanich, B., Watson, R.G., Andersen, P.: Normalize, Transpose, and Distribute: An Automatic Approach for Handling Nonscalars. ACM Transactions on Programming Languages and Systems 30(2), 9:1–9:49 (2008)
7. Curtin, B.: Algebraic Characterizations of Graph Regularity Conditions. Designs, Codes and Cryptography 34(23), 241–248 (2005)
8. Devin, F., Boulet, P., Dekeyser, J.L., Marquet, P.: GASPARD: a visual parallel programming environment. In: Proceedings of the International Conference on Parallel Computing in Electrical Engineering, 2002. PARELEC '02. pp. 145–150. IEEE (2002)
9. Egorova, D., Zhidchenko, V.: Visual Parallel Programming as PaaS Cloud Service with Graph-Symbolic Programming Technology. Proceedings of the Institute for System Programming 27(3), 47–56

10. Hammarlund, P., Lisper, B.: Data Parallel Programming: A Survey and a Proposal for a New Model. Tech. rep., Royal Institute of Technology, Department of Teleinformatics, Sweden (1993)
11. Johnston, W.M., Hanna, J.R.P., Millar, R.J.: Advances in Dataflow Programming Languages. ACM Computing Surveys 36(1), 1–34 (2004)
12. Kovartsev, A.N., Zhidchenko V., V., Popova-Kovartseva, D.A., Abolmasov, P.V.: The Basics of Graph-Symbolic Programming Technology. In: OSTIS-2013: Open Semantic Technologies for Intelligent Systems. pp. 195–204. BSUIR, Minsk (2013)
13. Kuhn, A., Conradt, J.: Review of Novel Computing Architectures for Neural Applications (2014), https://www.nst.ei.tum.de/fileadmin/w00bqs/www/publications/as/2014SS-HS-NeuralComputingArchitectures.pdf
14. Master, P., Furtek, F.: A new computing architecture for Big Data and AI applications (2017), http://impactvc.com/a-new-computing-architecture-for-big-data-and-ai-applications/
15. Nemanich, B., Cooke, D., Rushtom, J.N.: SequenceL: Transparency and Multi-core Parallelisms pp. 45–52 (2010)
16. Object Management Group, The: UML Profile for MARTE (2011), http://www.omg.org/spec/MARTE/1.1/
17. OpenFog Consortium Architecture Working Group: OpenFog Architecture Overview (2016), https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf
18. Solinas, M., et al.: The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices. In: 2013 Euromicro Conference on Digital System Design. pp. 272–279 (2013)
19. Wolfram, S.: An Elementary Introduction to the Wolfram Language - Second Edition. Wolfram Media (2017)
20. Алакоз, Г. М. и др.: Вычислительные наноструктуры. Часть 1. Задачи, модели, структуры. Бином. Лаборатория знаний, М. (2009)
21. Воеводин, В. В.: Математические модели и методы в параллельных процессах. Наука, М. (1986)
22. Лукин, Н. А.: Функционально-ориентированные процессоры с однородной архитектурой для реализации алгоритмов бортовых систем управления. In: Доклады пятой международной конференции «Параллельные вычисления и задачи управления». pp. 1177–1185 (2010)
23. Лукин, Н. А., Филимонов, А. Ю., Тришин, В. Н.: Облачная среда программирования однородных вычислительных систем. In: Языки программирования и компиляторы — 2017. pp. 181–184 (2017)