DOI: 10.14529/jsfi230202

High-Level Synthesis Toolchain "Theseus" for Multichip Reconfigurable Computer Systems

Aleksey I. Dordopulo[1], *Ilya I. Levin*[2], *Vyacheslav A. Gudkov*[1,2], *Andrey A. Gulenok*[1]

In the paper we consider the high-level synthesis toolchain for transformation of programs written in C (the standard ISO/IEC 9899:1999) into configuration files of field programmable gate arrays (FPGAs) used in multichip reconfigurable computer systems. Unlike most academic (DWARV, BAMBU, LEGUP) and commercial (CatapultC, Vivado HLS, Vivado Vitis) high-level synthesis tools, "Theseus" uses the original methodology of transformation (porting) sequential calculations into a parallel-pipeline configuration of FPGA hardware. For a sequential program, an information graph is created and transformed into the maximally parallel structure, which is then ported to a specified configuration of the reconfigurable computer system using formal methods of reduction of performance and hardware costs without marking the source text with auxiliary parallelization directives. The distinctive feature of the approach is a significantly smaller number of analyzed variants in comparison to parallelizing compilers. Due to this, it is possible to reduce the porting time of sequential programs in the synthesis of solutions for reconfigurable computer systems with a set of FPGA chips interconnected by a spatial communication system. In the paper we show the results of porting a number of application tasks to the architecture of various reconfigurable computer systems using the proposed "Theseus" toolchain.

Keywords: high-level synthesis, HLS, program translation, C language, performance reduction, reconfigurable computer system, programming of multiprocessor computer systems.

Introduction

Reconfigurable computer systems (RCS) containing field programmable gate arrays (FPGAs) provide adaptation of the system's architecture to the task's structure and reduction of the additional charges for organization of calculations. This provides a significant gain in task solution time in comparison to multiprocessor systems [1] even with a 10-fold difference in operating frequencies. RCSs that contain many FPGA chips connected by a switching system [2, 3], significantly exceed the cluster computer systems in real performance of applications and power effectiveness on many real-life tasks, but their programming and debugging require extensive and deep knowledge of the FPGA circuitry and architecture from programmers. Simplification of RCS programming is possible with the development of high-level synthesis software [4, 5], which converts sequential programs, written in high-level languages, into FPGA configuration files.

In this paper we consider the description of methods, used for automatic transformation of a sequential program by "Theseus" tools for high-level synthesis of multichip RCS configuration files. Section 1 of this paper provides a brief overview of high-level synthesis tools and assessment of their applicability for multichip RCSs. Section 2 describes the structure of the Theseus toolchain and intercomponent communication during synthesis of the multichip solution from the initial program. Section 3 presents methods of transformations of sequential programs for automatic adaptation of tasks to the available RCS hardware resource for each component of the toolchain. Section 4 presents "Theseus" porting results for a number of tasks compared with Vivado HLS results. In conclusion, we analyse the obtained results.

[1]Supercomputers and Neurocomputers Research Center, Taganrog, Russian Federation
[2]Southern Federal University, Taganrog, Russian Federation

1. Overview of High-Level Synthesis Tools

Currently, high-level synthesis tools or HLS compilers [4, 5] are widely used for programming FPGAs. Such tools convert a high-level program into configuration files of special-purpose hardware, using HDL hardware description languages. Depending on the language of the input program, HLS compilers refer to translators of either problem-oriented(for a certain problem area) or general-purpose languages.

The classification of high-level synthesis tools according to these criteria is shown in Fig. 1. Here, ▷ means that currently an HLS compiler is developed and supported, ‖ means that a project is suspended and there is no information concerning future development plans, □ means that a project is finished and not supported anymore.

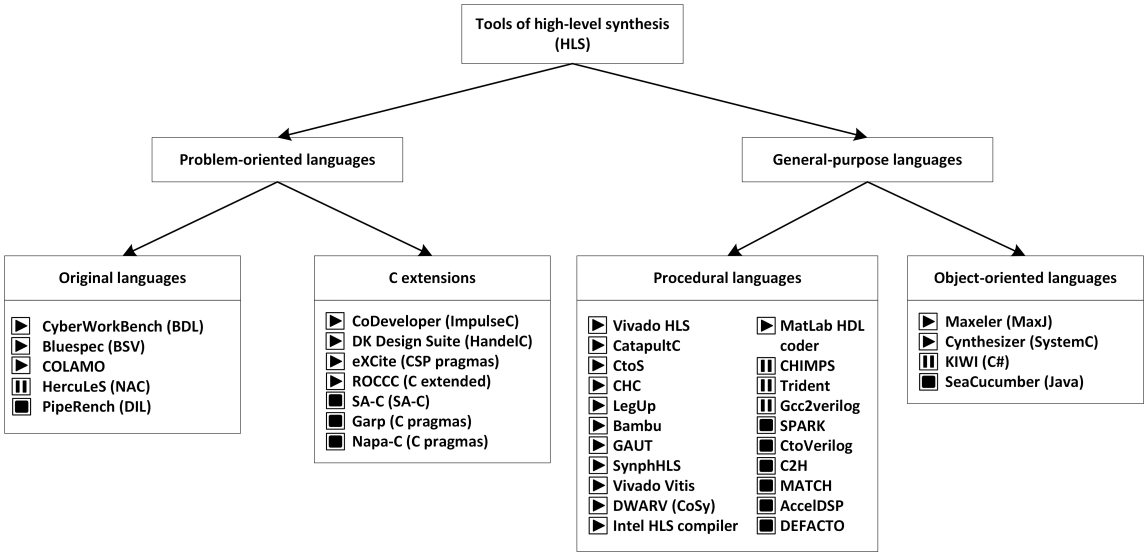

Figure 1. Classification of high-level synthesis tools

The most well-known academic (DWARV [6], BAMBU [7], LEGUP [8]) and commercial (CatapultC, Vivado HLS [9], Vivado Vitis [10]) HLS compilers convert written in C (or its dialect) program into a VHDL program of digital devices. Xilinx Vivado HLS and Vitis computer-aided design systems have become generally accepted standards of high-level synthesis tools.

Vivado HLS [9] is a tool for fast project design. It contains a number of optimization tools, typical for both compilers and digital circuit design systems [11]. Xilinx Vitis [10] is a development environment that combines graphical tools, compilers, analyzers and debuggers to speed up the execution of code fragments of sequential programs implemented in the FPGA architecture. Vitis is focused on Xilinx FPGA-based accelerators for server and cloud applications and/or Alveo accelerators for embedded devices.

The use of HLS compilers does not guarantee [11] an effective implementation of calculations in RCS for any program written in the C language. For a synthesized IP core, the gain in the speed of calculations compared to a general-purpose microprocessor is provided by the properties of the FPGA architecture. Computer-aided design of an IP core simplifies the porting of calculations to the FPGA architecture, and the programmer must scale IP cores according to the available hardware resource. There are no computer-aided tools of scaling IP cores and organizing data flows (at least within one chip), and this task is assigned to the user. For multichip RCS [2, 3], where many FPGAs are connected by a spatial switching system, the complexity of scaling and matching of IP cores for an efficient solution is rising significantly [12].

Unlike the HLS tools mentioned in [4, 5], the developed toolchain "Theseus" converts a sequential C program into the most parallel form, which is adapted to a specified hardware computing resource and translated into FPGA configuration files of multichip RCS. The toolchain converts the input program without users **#pragma** directives or other manual code marking and provides automatic synchronization of data and control signals for synthesizing multichip solutions. The basis for automatic adaptation of an application to the architecture and configuration of a specified RCS is the original porting methodology, which provides the search for an efficient solution for a priori unknown hardware resource of the computer system.

2. Structure of the "Theseus" Toolchain

The "Theseus" toolchain (Fig. 2) consists of four programs called Angel, Centaur, Procrustes and Sinis, each of which performs a functionally completed transformation:

- Angel converts the input C program into the maximally parallel structure, and then translates it with an implicit description of parallelism into the syntax of the COLAMO programming language;
- Centaur converts the maximally parallel structure into a resource-independent parallel-pipeline COLAMO-program;
- Procrustes automatically, with no users code, selects the parameters of the structure for its efficient implementation in the RCS architecture;
- if the hardware resource is insufficient for structural implementation of the main fragment of the task, then Sinis reduces the parallelism of the structure, increasing the task solution time, but making it possible to execute on the available hardware resource.

The input sequential C program (ISO/IEC 9899:1999) is translated by Angel into the maximally parallel structure, which is then converted into a scalable form and ported by Procrustes to the available RCS hardware resource. The result of the "Theseus" toolchain is a program written in the high-level language COLAMO with the parameters that match the limitations of the resource and real performance rate. The COLAMO [2] translator and the multichip solution synthesizer Fire!Constructor [2] translate the COLAMO program into configuration files for FPGAs of the multichip RCS. Then, the Xilinx Vivado synthesizer generates bitstream configuration files (* .bit) for each chip.

3. Transformation of a Sequential Program by the "Theseus" Toolchain to Available RCS Resource

3.1. Angel: Transformation of an Input Sequential Program into Maximally Parallel Structure

The sequential written in C program is transformed by the Angel translator into the maximally parallel form – the task information graph, which is then transformed into the COLAMO program with an implicit description of parallelism. The task information graph (TIG) is a finite oriented acyclic graph whose vertices correspond to operations on data, and arcs reflect the data dependencies between them. All vertices of the TIG are distributed in layers and iterations [2]. Layers, like the layers of the algorithm graph [13], contain data-independent vertices (subgraphs), and iterations describe the data dependence among vertices (subgraphs) of different layers. Unlike other graph forms used for the representation of calculations [13], layers and

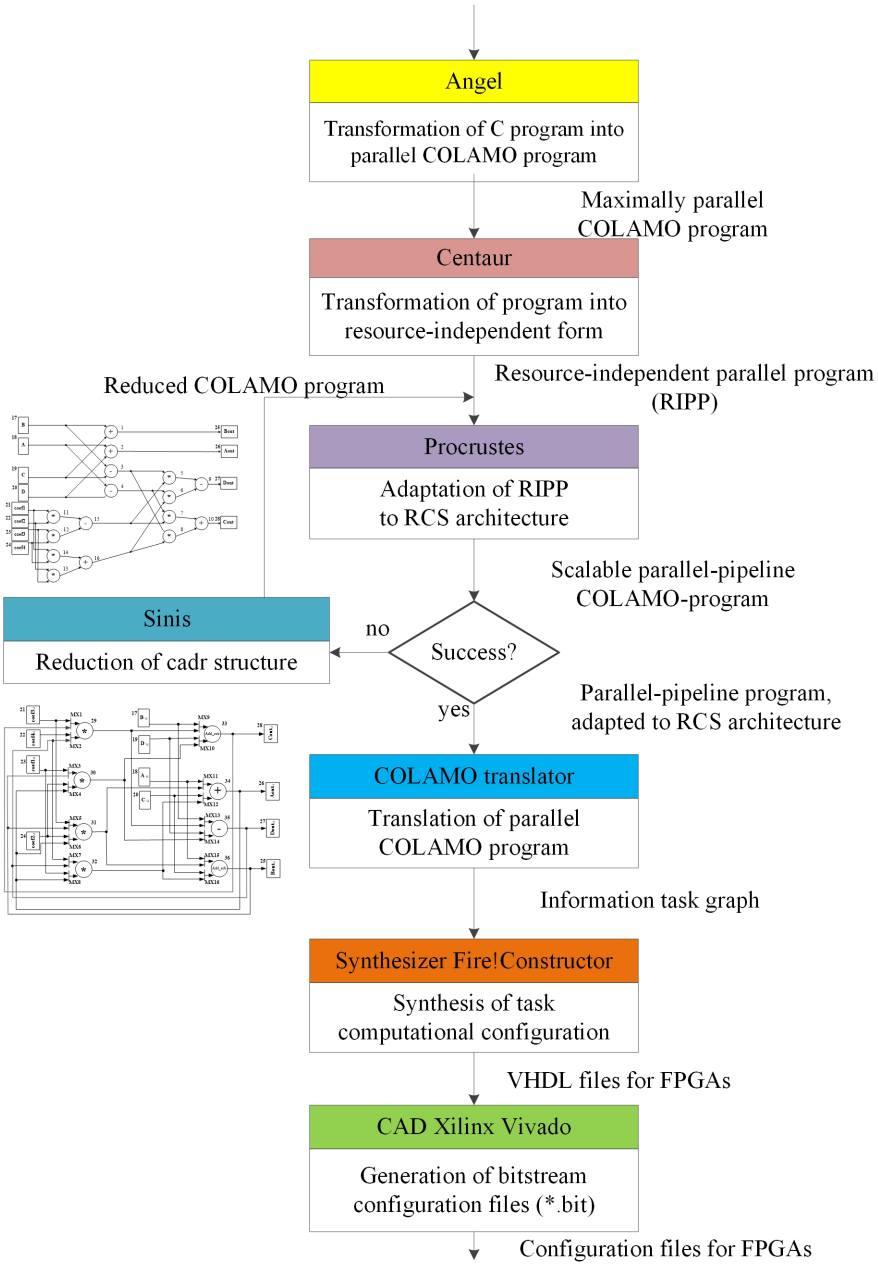

Figure 2. Structure of the "Theseus" toolchain

iterations of the TIG describe not only arithmetic and logical operations, but also subgraphs corresponding, for example, to a loop body in sequential programs. It is rather easy to generate a TIG from a sequential program by cycles unrolling. The TIG describes the maximum, theoretically possible, parallel form of the tasks calculations with the subgraphs that are distributed across layers and iterations.

Owing to the distribution of vertices and/or subgraphs by layers and iterations, it is possible to represent data dependencies among the task's fragments at different levels of the hierarchy. The variants of the sequential program, differing in the execution order of the cycle's iterations (subgraphs), differ in the TIG only in the topology of data-independent subgraphs in the layer, and are considered data-indistinguishable or equivalent. Therefore, sequential programs describing the same task with the same data dependencies, which are different in syntax and form, will correspond to the TIGs that are equivalent in the results of calculations, but differ only in the

topology of subgraphs in the layer. Due to the invariance of the representation of topologically different parallel calculations in the TIG, we can, unlike a parallelizing compiler, reduce the number of possible variants for their representation. C program which solves a system of linear algebraic equations (SLAE) by the method of Gaussian elimination is shown in Fig. 3a and its TIG is shown in Fig. 3b.

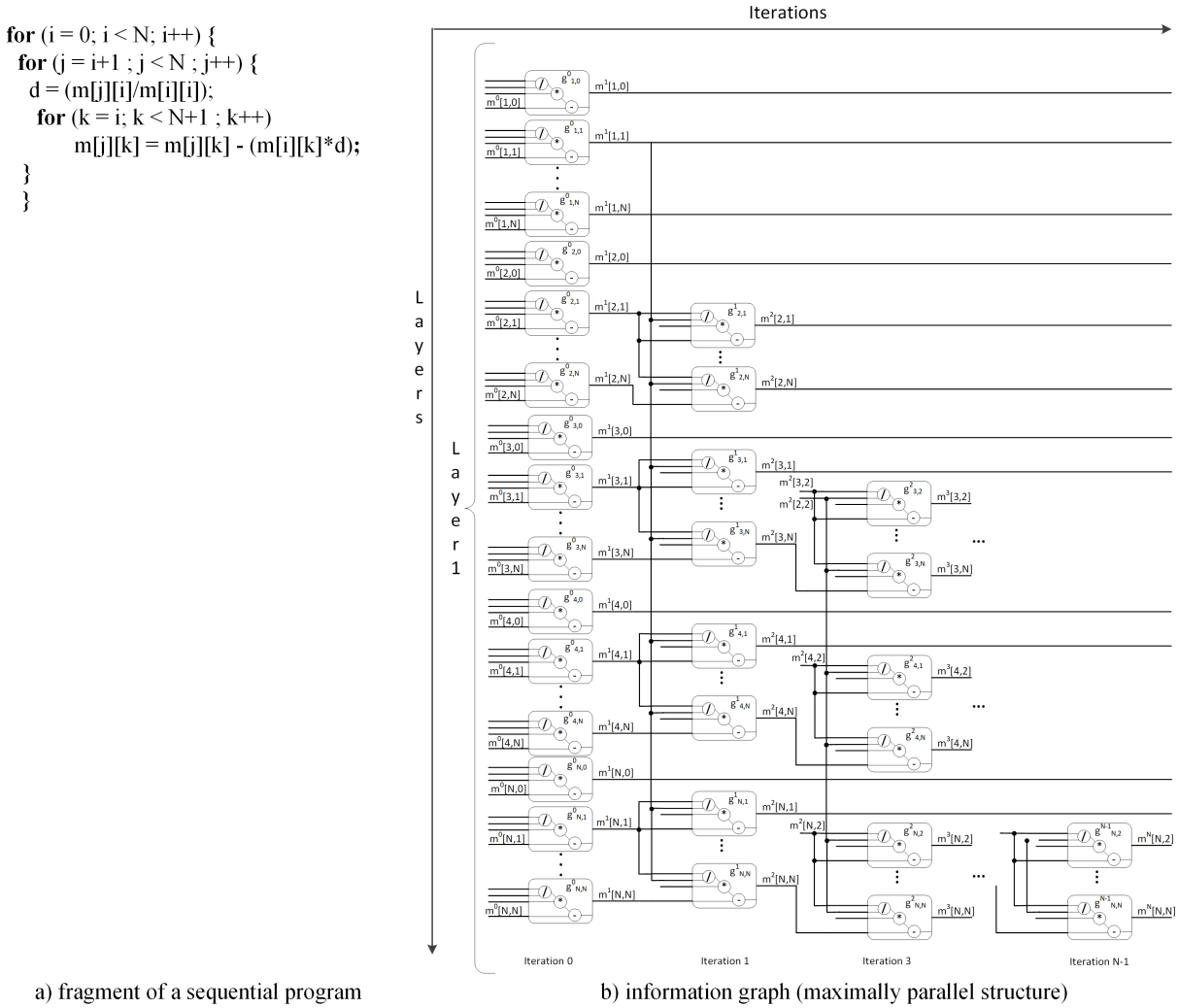

Figure 3. Solution of a SLAE using the method of Gaussian elimination

The subgraphs g^i_{jk} in Fig. 3b contain 3 operation vertices: division, multiplication, and subtraction, corresponding to the operations of the loop bodies on **j** and **k** in lines 3 and 5 in program Fig. 3a. The data-independent subgraphs g^i_{jk} belong to the layers, and dependence among subgraphs of different layers is specified by connections among subgraphs: the output $m^1[1,1]$ of the subgraph $g^0_{1,1}$ of the zero iteration is the input of the subgraphs $g^1_{(2..N,1..N)}$ of the first iteration, the output $m^2[3,2]$ of the subgraph $g^1_{3,2}$ is the input of $g^2_{(3..N,2..N)}$ of the second iteration, etc.). The specificity of the SLAE solution by the method of Gaussian elimination is reducing the processed data by one line at each iteration of the loop on **j**, so each next layer of the TIG contains fewer subgraphs g^i_{jk}.

The TIG is transformed by the Angel translator into a *cadr* structure [2]. Cadr structure is an indivisible unity of computational structure and rules for organizing input and output data flows, for which a reduction of the computational structure leads to an increase in data

streams. So, for the same task a variety of cadr structures differing in computational structures and data flows is possible. The Angel translator transforms TIG into a maximally parallel cadr structure: the operation vertices are replaced by computing devices, and the arcs are replaced by the connections of the switching system. The maximally parallel structure contains not the vertices of the graph, but computing devices with latency, frequency and performance, data processing interval, etc. Due to this, it is possible to calculate the performance characteristics and the execution time on the RCS. The maximally parallel structure, obtained as a result of transformation of a sequential program by the Angel translator, is described according to the rules of the COLAMO programming language with an implicit description of parallelism.

3.2. Centaur: Transformation of the Maximally Parallel Structure into the Resource-Independent Parallel-Pipeline Form

Maximally parallel structure performs all task operations with the minimum latency and maximum performance. This requires the hardware implementation of all operations and simultaneous supply of all input data, which is usually unattainable for real computer systems. For implementation in a computer system, the maximally parallel structure can be transformed into cadr structures that are more rational in terms of the occupied hardware resource and provide data equivalence of the calculation results. For example, the maximally parallel structure (Fig. 3b) can be transformed to a cadr structure with all hardware-implemented iterations (Fig. 4a) by defining the order of execution of its subgraphs (ordering by layers). With further ordering of the subgraphs (by iterations), the cadr structure with the implemented iterations (Fig. 4a) can be transformed to a minimum cadr structure (Fig. 4b) with a single hardware-implemented (basic) subgraph.

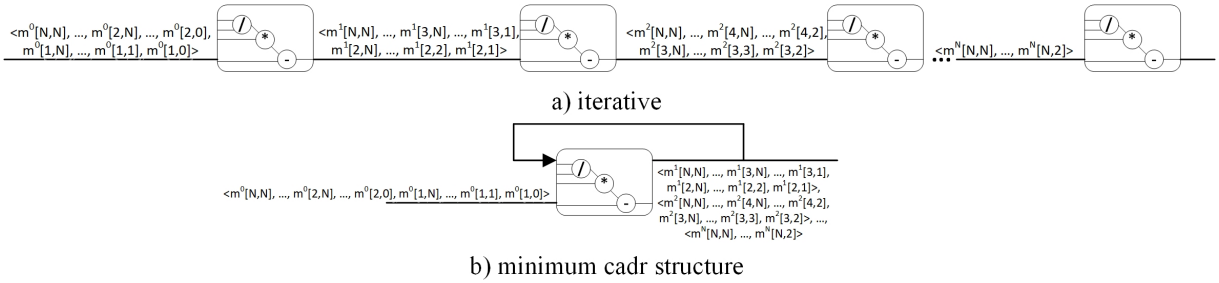

Figure 4. Cadr structures of the SLAE solution by the Gaussian method

For convertion of the maximally parallel structure into one of many possible cadr structures, it is transformed by the Centaur tool into a resource-independent parallel-pipeline form, which makes possible to change the number of hardware-implemented subgraphs using several parallelism parameters. Centaur identifies basic subgraphs for large fragments of calculations, analyzes the data dependencies in these fragments and among them, ensuring the implementation of the rules of single substitution and single assignment. Arbitrary access to the memory of a sequential program is transformed to data flow processing. All arrays and loops, used in the descriptions of the cadr structures in the COLAMO program, are automatically split into parallel (vector) and sequential (stream) components. As a result, Centaur generates a COLAMO program containing a cadr structure with parallelism parameters. Variation of these parameters transforms the cadr structure.

3.3. Procrustes: Transformation of Resource-Independent Cadr Structure to Available RCS Hardware Resource

Cadr structures (see Fig. 3b, 4a and 4b) differ in their hardware resource and the task solution times. Parallelism, solution time and hardware resource occupied by the cadr structures are specified by the following parameters: the number of layers (information-independent subgraphs), the number of iterations (information-dependent subgraphs), the number of instructions (devices) in the basic subgraph, the capacity and interval of data processing. Therefore, each cadr structure can be represented by a point (or vector) in the 5D space of possible cadr structures (Fig. 5).

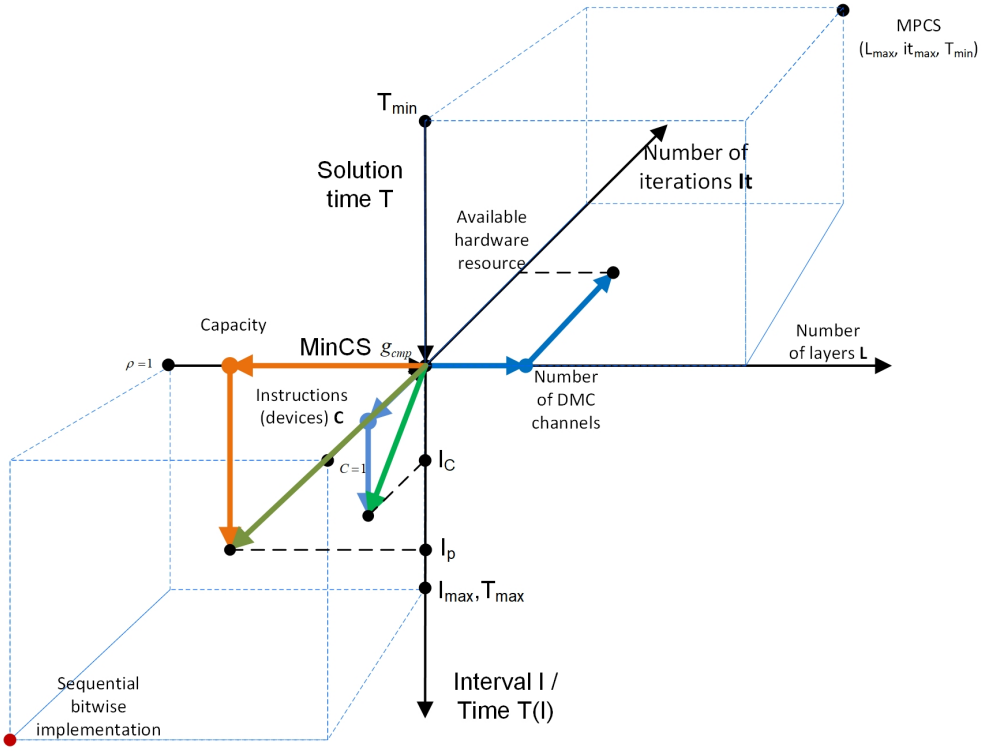

Figure 5. Space of implementations of cadr structure calculations

Space points correspond to cadr structures with various parameters of parallelism (performance) and occupied hardware resource. Variation of the performance parameters (shown by colour arrows in Fig. 5) changes the hardware resource occupied by the cadr structure, which is graphically represented as a segment between two points of space. So, porting is the variation of the performance parameters and the occupied hardware resource performed to achieve the available hardware resource. Porting is a continuous movement from the maximally parallel structure to a cadr structure located in the area of the available resource of the computer system. Porting can be defined as the sequential approaching to local and global goals. The local goal is to reach the area of available resource, and the global goal is to find a rational cadr structure that provides a specified performance rate. The search for the performance parameters of the cadr structure that satisfy both the limitations of the available RCS hardware resource and the specified performance rate is provided by Procrustes with the help of the following methodology.

According to the methodology, a cadr structure CS with hardware costs $A_{CS} = (a_1^{CS}, a_2^{CS}, \ldots, a_n^{CS})$ and performance parameters $P_{CS} = (p_1, p_2, \ldots, p_n)$ is transformed to the hardware resource $A_{HCS} = (a_1^{HCS}, a_2^{HCS}, \ldots, a_n^{HCS})$ of a reconfigurable or

hybrid computer system. It should be noted that the hardware costs A_{CS} of the cadr structure non-linearly depend on the performance parameters P_{CS}: $A_{CS} = \Psi(P_{CS})$, and one performance parameter p_i may simultaneously influence several parameters of the hardware resource.

The purpose of porting is to detect the performance parameters P'_{CS}, which are the solution of

$$\begin{cases} A'_{CS} = \Psi(P'_{CS}) \le A_{HCS}, \\ Perf(P'_{CS}) \ge Perf_3, \end{cases}$$

where $Perf(P'_{CS})$ is the performance of the cadr structure, and $Perf_3$ is the specified real performance rate.

To achieve the purpose, the following actions are required.

1. **Calculation of reduction parameters**
   1.1. Determine the critical resource and the reduction coefficient of hardware costs $K_{cr} = max\left(\frac{a_i^{CS}}{a_i^{HCS}}\right) > 1$.
   1.2. Determine the reduction coefficient $R = \lceil K_{cr} \rceil$.
   1.3. Arrange the performance parameters p_i in the tuple $< P_{CS} >$ according to the degree of influence on the critical resource.
   1.4. **If** <an unreduced parameter exists in $< P_{CS} >$>, **then**
   – select the performance parameter with the maximum impact on the critical resource: $p_i : \psi(p_i) = maxK_{cr}$.
   else: go to item 7 to select a card structure with the maximum performance.
2. **Determination of the effective reduction step** R^*.
   2.1. For the selected performance parameter p_i, determine one of the possible options for the effective reduction step R:
   1) $R^* = R$;
   2) $R^* = f(R, p_i), R^* < R$;
   3) $R^* = \|p_i\|, R^* > R$.
3. **Reduction (decreasing of the performance parameters) of the cadr structure**
   3.1. Reduce the parameter p_i with the effective step R^*: $p'_i = \Theta(p_i, R^*)$ and save it to a tuple: $< P'_{CS} >=< P_{CS} > \leftarrow p'_i$.
4. **Evaluation of the current cadr structure and selection of alternatives**
   4.1. Calculate hardware costs and performance of the cadr structure $A'_{CS} = \Psi(P'_{CS}, Perf(P'_{CS}))$.
   4.2. Evaluate the achievement of porting purpose from the conditions $A'_{CS} = \Psi(P'_{CS}) \le A_{HCS}$ and $Perf(P'_{CS}) \ge Perf_3$.
   4.3. **If** both conditions are fulfilled, **then**
   – go to item 6 to save the current parameters of the cadr structure
   else: {Analysis of alternatives for further reduction:}
   If $A'_{CS} > A_{HCS}$ {the hardware costs exceed the available RCS resource}, **then**:
   If $R^* = R$, **then**
   – increase the reduction coefficient $R := R + \Delta$, where $\Delta = \left\lceil \frac{A'_{CS}}{A_{HCS}} - 1 \right\rceil$;
   – restore the original value of the performance parameter $< P'_{CS} >=< P_{CS} > \leftarrow p'_i$;
   – go to item 1.3 for reduction with the increased coefficient R.
   Else
   – go to item 1 to decrease the critical resource using the next performance parameter p_{i+1}.

{if $A'_{CS} \le A_{HCS}$ and $Perf(P'_{CS}) < Perf_3$, then we go naturally to induction}

5. **Induction (increasing of the previously reduced performance parameters) of the cadr structure**
   5.1. Form a tuple of the previously reduced parameters $< p_{i-1}, \ldots, p_1 >$.
   5.2. Select the next item of the tuple p_k, determine its scaling coefficient according to the hardware resource $K_M = min\left(\frac{A'_{CS}}{A_{HCS}}\right)$ and the induction coefficient $Ind = \lfloor K_M \rfloor$.
   5.3. For the selected performance parameter p_k, determine one of the possible effective induction steps:
   1) $Ind^* = Ind$;
   2) $Ind^* = \phi(Ind, P_k) < Ind$;
   3) $Ind^* = \|p_k\|$.
   5.4. Increase p_k by the selected step Ind^*: $p'_k = \Theta^{-1}(p_k, Ind^*)$ and save it into the tuple: $< P'_{CS} >=< P'_{CS} > \leftarrow p'_k$.
   5.5. Go to item 4.
6. **Save the cadr structure**
   6.1. Add the current parameters of the cadr structure to the list, ordered by the minimum performance.
   6.2. **If** $Perf(P'_{CS}) < Perf_3$ **and** <there were other critical resources>, **then**
   – select the next critical resource and go to item 1.1.
7. **Select a reasonable variant for reduction of the cadr structure**
   7.1. Issue the first item of the list of cadr structures with the highest performance.

Unlike most HLS compilers [4, 5], the proposed methodology changes the parameters of the cadr structure in order to provide the rational use of available hardware resource and to achieve a specified rate of real performance. If the specified performance rate $Perf_3$ is not achievable, the result of the porting will be the cadr structure with the highest performance among all analyzed cadr structures. Unlike the methods of structural and procedural organization of calculations, which provided a rational cadr structure for a priori known and fixed resource, the transformation, according to the Procrustes' methodology, is a continuous function which depends on the architecture and configuration of the computer system. This fact ensures movement in the space of cadr structures not only "down", towards reducing parallelism (reduction), but also towards its increase (induction), if the hardware resource allows. Due to combination of reduction and induction, it is possible to find rational performance parameters of cadr structures even if the hardware resource is insufficient for the hardware implementation of the basic subgraph.

The total number of analyzed variants depends on the number of the performance parameters of the cadr structure that effect hardware costs. For the FPGA architecture, the number of memory channels, the number of Look-Up Table cells in FPGA chips, the number of Block RAM internal memory blocks, the number of High Bandwidth Memory blocks, the number of Digital Signal Processor blocks, and the number of Flip-Flop registers may vary in the cadr structure. Therefore, for FPGAs, the total number of analyzed variants of rational cadr structures with different parameters is small and does not exceed $6! = 720$.

Procrustes generates a parallel-pipeline COLAMO program, which contains the cadr structure with the performance parameters calculated for the available resource of the specified RCS. This program is transformed by the COLAMO translator and the Fire!Constructor synthesizer into configuration files for FPGAs of multichip RCSs. Then, these files are translated by Xilinx Vivado into bitstream configuration files (*.bit).

3.4. Sinis: Transformation of the Minimum Cadr Structure When the RCS Hardware Resource is Insufficient

In some cases, the available RCS hardware resource may be insufficient for hardware implementation even of the basic subgraph. Previously, the only way to organize calculations in this case was a sequential implementation on a processor. Due to the methodology, presented in the previous section, it is possible to continuously reduce the parallelism of the cadr structure, using the performance reduction by devices and by capacity, and micro-cadrs – a new form of organization of calculations. The main difference between a micro-cadr (m-cadr) MCS_2^{Op} (Fig. 6c) and a cadr structure CS (Fig. 6a) is the combination of several operations in one computing device, which leads to increase of the data processing interval, but provides one and the same balanced data processing rate. There is no need to use additional memory to store intermediate results of the input data flow processing when the cadr structure CS is implemented structurally and procedurally as two reduced cadr structures $CS/2$ (Fig. 6b).

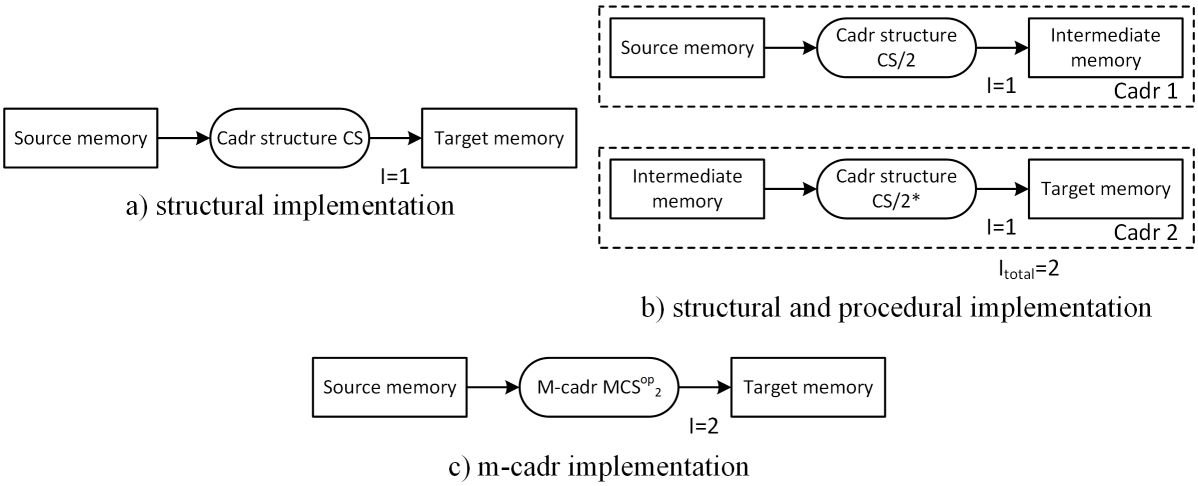

Figure 6. Methods of implementation of the cadr structure CS

Variants of generation of m-cadrs depend both on the problem area and on the solving task. The possible strategies for their creation are discussed in [14], where several m-cadrs are proposed for the task of digital signal processing. These transformations are performed by the Sinis tool, which, if necessary, is called by the Procrustes tool. Sinis obtains the results and returns them to Procrustes that uses them to find a rational version of the cadr structure according to the methodology from Section 3.3.

4. Results of Task Porting Performed by the "Theseus" Toolchain

The research of the efficiency of the Theseus high-level synthesis toolchain was carried out by porting a number of model tasks to various RCS architectures. The real performance rate of the ported solution for the Theseus toolchain, as well as for circuit engineers, was specified at least 0.6 from the peak one. Operability of the solutions obtained with the help of the Theseus toolchain was checked by running them on the corresponding RCS, and the characteristics of each task were compared with the results of FPGA designers. Transformation and porting was performed on a PC with Intel (R) Core (TM) i7-8750H @ 2.2 GHz processor, 16 GB of RAM, and

Windows 10 Pro operating system. Automatic porting was carried out for five model tasks: the symmetric-key block cipher DES, the MD5 and SHA-1 hash functions, and the Gaussian method and Jacobi method for 3-diagonal matrices. Each task was ported by FPGA designers and the toolchain to three different RCS hardware platforms (Fig. 7): Taygeta [3, 15], Tertius [15] and Tertius-3 [15].

The Taygeta RCS with the performance of 2.66 Tflops contains 4 20-layer printed circuit boards with double-sided mounting of 8V7-200 elements. Each circuit board contains 8 XC7VX485T-1FFG1761 FPGAs with 48.5 million equivalent gates, 16 SDRAM DDR2 distributed memory chips with a total volume of 2 GB, LVDS and Ethernet interfaces and other components.

The FPGA field of the desktop reconfigurable computers Tertius and Tertius-3 is not a separate board. It is integrated with a motherboard with an Intel Core I5 6300U processor. Tertius has the performance of 2.5 Tflops and contains 4 Xilinx Kintex UltraScale XCKU095 FPGAs with the capacity of 100 million equivalent logic gates each, interconnected in a ring by LVDS and GTY/GTH channels. Two dynamic memory modules with a capacity of 1 GB are connected to each FPGA, so the total memory size is 8 GB. Tertius-3 has the performance of 5.6 Tflops and contains twice as many chips of another FPGA type – 8 Virtex XCVU095-1FFVB1760C FPGAs.

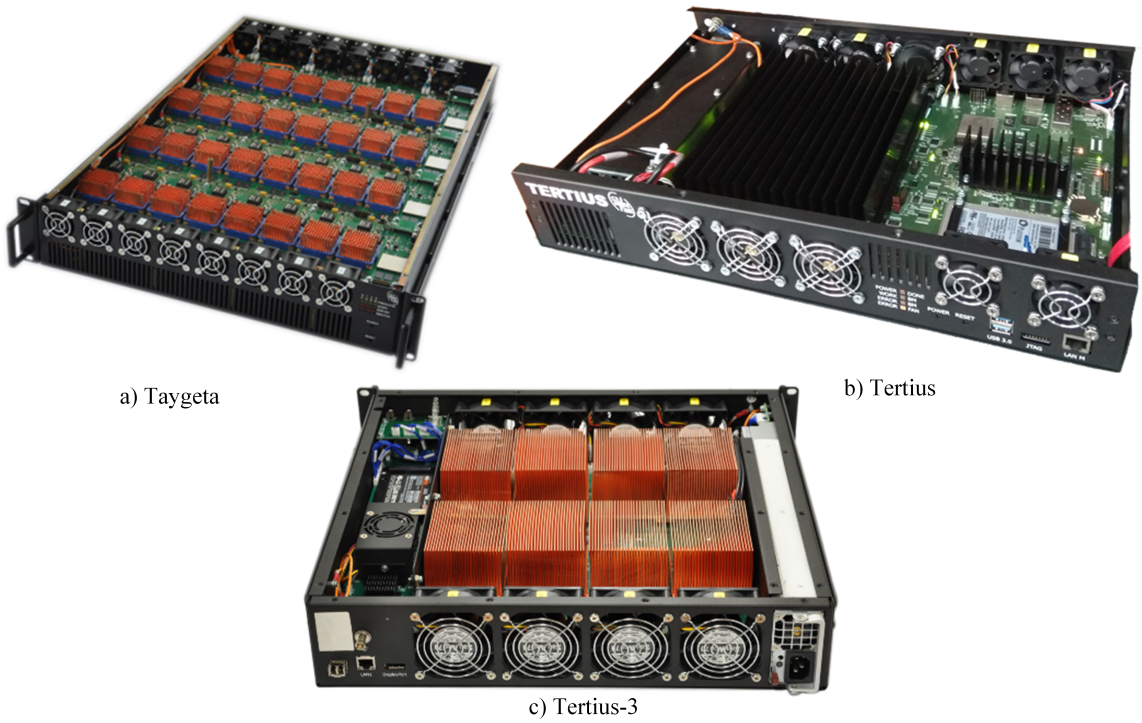

Figure 7. RCS hardware platforms for task porting

For each task, we measured the transformation time of its cadr structure, the task porting time, and the achieved performance. The cadr structure transformation time was defined as the working time of Procrustes. The porting time was considered as the sum of the cadr structure transformation time and the synthesis time of the FPGA bitstream configuration file, which depends on the logical capacity and the utilisation of the FPGA chip. With 90% utilisation of FPGA chips, it is at least 3 hours (10.800 seconds) for Taygeta and 7 hours (25.200 seconds) for Tertius and Tertius-3. According to experience of solving the same tasks by FPGA designers,

the cadr structure transformation time for all hardware platforms was taken equal to two 8-hour working days or 57.600 seconds. The achieved real performance rate was defined as the ratio of the number of subgraphs in solutions obtained by the toolchain and FPGA designer (Tab. 1).

The transformation of the cadr structure by the Procrustes tool is performed significantly, by 1–3 decimal orders, faster than by FPGA designers, which is the expected effect of automation. Since the project synthesis time for FPGAs is significantly longer than the cadr structure transformation time, the total gain in the porting time of model tasks with the synthesis of bitstream configuration files for the selected RCS hardware platforms in Tab. 1 is equal to 3–6.3 times. The real performance rate achieved by the toolchain in porting model tasks does not drop below the specified level of 0.6 and varies slightly for different RCS hardware platforms, which is explained by the architectural features of different FPGA crystals.

Table 1. Results of porting model problems

Problems	DES	MD5	SHA-1	Jacobi	Gauss
RCS Taygeta					
Porting time in seconds	10836.62	10938.86	10841.66	13429.90	12316.38
Gain	6.31	6.25	6.31	5.09	5.55
Real performance rate	**0.63**	**0.63**	**0.86**	**0.86**	**0.86**
RCS Tertius					
Porting time in seconds	25238.58	25345.13	25241.09	25912.87	26989.25
Gain	3.28	3.27	3.28	3.20	3.07
Real performance rate	**0.65**	**0.67**	**1.00**	**0.85**	**0.80**
RCS Tertius-3					
Porting time in seconds	25239.02	25344.79	25241.76	25914.02	26991.12
Gain	3.28	3.27	3.28	3.20	3.07
Real performance rate	**0.65**	**0.67**	**1.0**	**0.85**	**0.80**

The results of porting the model tasks DES and SLAE solution with the help of the Gaussian method, performed by the toolchain, were compared with the solution obtained by the Vivado HLS compiler for the Taygeta RCS. The solution of the DES model task obtained by Vivado HLS with one pipelined IP-core, which was scaled by FPGA designer to the available hardware resource of the Taygeta RCS, while its real performance rate was 5 times lower than that of the solution obtained by the toolchain. The solution, obtained by Vivado HLS as a result of porting the Gauss model task, contains one iterative stage versus 720 stages when translating the task by the toolchain. Using manual code marking with *#pragma* directives in Vivado HLS, we could get a solution for two iterative steps of the Gaussian elimination algorithm. According to the comparison of the results of porting model tasks by the toolchain, FPGA designers and Vivado HLS, we claim that the proposed methodology for the cadr structure transformation gets advantages over Vivado HLS.

Conclusion

The “Theseus” high-level synthesis toolchain, described in the paper, provides scalable solutions for multichip reconfigurable computer systems unlike the academic (DWARV, BAMBU, LEGUP) and commercial (CatapultC, Vivado HLS, Vivado Vitis) HLS tools. Automatic trans-

formation of the input sequential written in C program with no specialized code marking is performed by presenting the task in the form of a cadr structure, and by the original method of its porting to the available RCS hardware resource using formal methods of reduction of performance and hardware costs. The application of the developed methodology of cadr structure porting to the available hardware resource of the RCS significantly reduces the number of analyzed variants of parallel calculations and the porting time. Due to the use of the "Theseus" toolchain in porting a number of model tasks, it is possible to find rational solutions for multichip RCSs (with the effectiveness not less than 60% from the results of FPGA designers) for a significantly lower (in comparison with parallelizing compilers) number of transformations. Unlike well-known HLS compilers, the input C program is transformed automatically, without manual code marking or other user instructions. As a result, we get multichip configuration files with automatic synchronization of information and control signals.

Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of state assignments "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence" (topic FSEG-2022-0001).

References

1. Antonov, A.S., Afanasyev, I.V., Voevodin, Vl.V.: High-performance computing platforms: current status and development trends. Num. Meth. Prog. 22(2), 135–177 (2021). https://doi.org/10.26089/NumMet.v22r210

2. Guzik, V.F., Kalyaev, I.A., Levin, I.I.: Reconfigurable computer systems. SFedU Publishing, Taganrog (2016). 472 p.

3. Levin, I., Dordopulo, A., Fedorov, A., Kalyaev, I.: Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems. Supercomputing Frontiers and Innovations 3(1), 22–40 (2016). https://doi.org/10.14529/jsfi160102

4. Nane, R., Sima, V., Pilato, C. *et al.*: A Survey and Evaluation of FPGA High-Level Synthesis Tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(10), 1591–1604 (2016). https://doi.org/10.1109/TCAD.2015.2513673

5. Numan, M.W., Phillips, B.J., Puddy, G.S., Falkner, K.: Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and Toolchains. IEEE Access 8, 174692–174722 (2020). https://doi.org/10.1109/ACCESS.2020.3024098

6. Nane, R., Sima, V.-M., Olivier, B., *et al.*: DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler. In: 22nd International Conference on Field Programmable Logic and Applications

(FPL), Oslo, Norway, August 29-31, 2012. pp. 619–622. IEEE (2012). https://doi.org/10.1109/FPL.2012.6339221

7. Pilato, C., Ferrandi, F.: Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications. In: 2013 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, September 2-4, 2013. pp. 1–4. IEEE (2013). https://doi.org/10.1109/FPL.2013.6645550

8. Canis, A., Choi, J., Aldham, M., *et al.*: LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems. In: Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27 – March 1, 2011. pp. 33–36. ACM (2011). https://doi.org/10.1145/1950413.1950423

9. Make Slow Software Run Fast with Vivado HLS. https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf, accessed: 2023-03-23

10. Vitis Unified Software Platform Documentation. Application Acceleration Development. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf (2019), accessed: 2023-03-23

11. Tarasov, I.: Designing for Xilinx FPGAs using high-level languages in Vivado HLS environment. Components and Technologies 12 (2013), https://kit-e.ru/fpga/vivado-hls/

12. Kolganov, A.S. An experience of applying the parallelization regions for the step-by-step parallelization of software packages using the SAPFOR system. Num. Meth. Prog. 21(66), 388–404 (2020). https://doi.org/10.26089/NumMet.v21r432

13. Voevodin, V.V., Voevodin, Vl.V.: Parallel computing. BHV-Petersburg, Saint-Petersburg (2002). 608 p.

14. Dordopulo, A.I., Levin, I.I.: Performance Reduction For Automatic Development of Parallel Applications For Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations 7(2), 4–23 (2020). https://doi.org/10.14529/jsfi200201

15. Computational blocks of SRC of supercomputers and neurocomputers. http://superevm.ru/index.php?page=modern-developments, accessed: 2023-04-11