DOI: 10.14529/jsfi260205

"Theseus" Parallel Compiler for Multichip Reconfigurable Computer Systems

Vyacheslav A. Gudkov[1]***, Ilya I. Levin***[1]

Modern system software for field-programmable gate arrays (FPGA) and FPGA-based computer systems implements individual task fragments as IP cores. It requires their further integration into a unified computing structure, as well as ensuring correct information of data flows. The "Theseus" parallelizing compiler was developed and characterized by a comprehensive implementation of the entire problem for a variety of FPGA chips. The created automatic parallelizing compiler significantly reduces the requirements for developer qualifications in FPGA programming and does not require marking up the source code of a sequential program with service directives. The parallelizing compiler implements a technology for converting sequential calculations into the most parallel form an information graph of an application task algorithm that is automatically mapped to the target configuration of a multichip reconfigurable computer system using formal methods for reducing the performance of the computing structure. The parallelizing compiler makes it possible to significantly reduce the conversion time of sequential programs to parallel-pipelined solutions for reconfigurable computer systems containing multiple FPGA chips connected by a spatial communication system, while ensuring a guaranteed level of real performance. Due to the efficient organization of calculations and rational use of the available FPGA hardware resources, the "Theseus" compiler will provide significantly higher real performance of a reconfigurable computer system compared to the similar HLS compilers (High-Level Synthesis Compiler). The results of creating application problems from various subject areas for the "Arktur" reconfigurable computer systems using the proposed methods for performance increasing are presented.

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

Introduction

The high complexity of developing hardware configurations for field-programmable gate arrays (FPGAs) and computer systems based on them require the creation of new development tools with significant reduction in the implementation time for FPGA solutions and the vendor qualification requirements.

Currently, HLS compilers, allowing the FPGA programming in high-level languages, are replacing automated circuit design tools based on HDL (Hardware description language). This reduces the requirements for developers of computer systems and accelerates the programming process by quickly creating individual IP cores, which are digital machines or specialized processors, for computationally laborious program fragments. Automating of IP core generation by an HLS compiler makes it possible to quickly port segments of sequential programs, for example, written in C language, to the FPGA architectures. However, this does not guarantee the efficient implementation, since dataflow organization remains the vendor responsibility, and tools for scaling and synchronization (even within a single chip) are lacking. For reconfigurable computer systems (RCS) [1, 2] with multiple FPGAs, connected via a spatial interconnection network, the configuring of the synchronization and coordination of simultaneous operation of

[1] "Scientific Research Center of Supercomputers and Neurocomputers" Co Ltd ("SRC SC & NC" Co Ltd), Taganrog, Russian Federation

multiple IP cores is a complex and labor-intensive process, comparable in complexity to the development of a new circuit design.

The "Theseus" parallelizing compiler described in this paper is designed to convert the sequential C programs in the ISO/IEC 9899:1999 standard into multichip RCS solutions with automatic synchronization of data and control signals for all used FPGAs.

The paper presents the results of the development of the "Theseus" parallelizing compiler with automatic conversion of sequential C programs to an available hardware resource of multichip RCS. The paper consists of an Introduction, five sections and a Conclusion.

The first section provides a brief overview of known HLS tools. In the second section, the synthesis of the cadr structure of the parallel-pipeline program is considered. The third section discusses methods for increasing the productivity of cadr structures. The fourth section presents the structure of the modern "Theseus" complex, as well as the main operation stages of the new Ariadne component. The fifth section presents the results of operation of the modern parallelizing "Theseus" compiler on the modern high-performance computer system Arcturus. In Conclusion, the obtained results are summarized, and directions are formulated for further research.

1. Overview of Known High-level Synthesis Tools

For FPGA programming, along with traditional hardware design systems such as Xilinx Vivado and Intel Quartus Prime, high-level synthesis (HLS) tools or HLS compilers [3, 4] are increasingly being used. These tools convert a program in one high-level languages into configuration files of specialized hardware in Hardware Description Languages (HDL). Depending on the input language, HLS compilers are either problem-oriented language translators, i.e., versions of programming languages adapted to a specific problem domain, or general-purpose language translators, i.e., high-level language dialects for traditional microprocessors with some features and limitations. The classification of high-level synthesis tools according to these criteria is given in Fig. 1.

It can be noted that despite the differences in code conversion methods and data formats, HLS compilers have common limitations [3, 5, 6]:

- synthesis of the project is performed for a single FPGA chip located on a development board or a single-chip reconfigurable computer system (RCS);
- to create an effective project, it requires pre-marking the code with directives (e.g., #pragma in Vivado), which requires deep and sometimes expert knowledge about the synthesized project;
- synchronization and scaling of IP cores within a solution are performed by the vendor independently.

The disadvantage of traditional HLS compilers is the focus on programming a single single-chip FPGA, which simplifies synthesis but limits scalability. Problems requiring simultaneous operation of several IP cores (for example, big data processing or neural network models) force the development of complex synchronization systems manually. This increases development time and increases the likelihood of errors, especially when working with multiple FPGAs.

The synthesis of multichip solutions is significantly more complicated, since it requires the support and synchronization of a large number of computing devices for various forms of computing organization, as well as the analysis and selection of rational options from a variety of possible ones. Therefore, it is comparable in complexity to automatic parallelization.

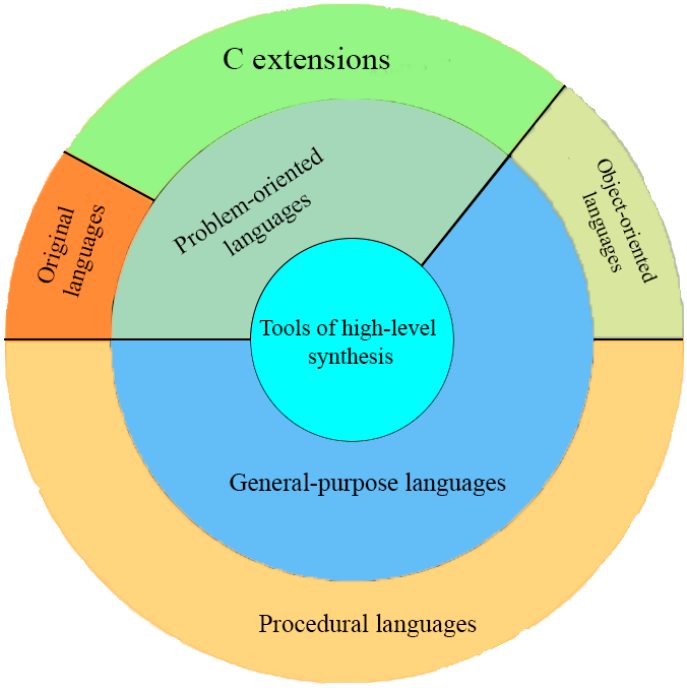


Figure 1. Classification of high-level synthesis tools

The "Theseus" parallelizing compiler developed at SRC SC and NC [7] converts a sequential C program to an available hardware resource of multichip RCS. The differences between the "Theseus" complex and the most well-known Xilinx Vivado HLS and Vitis HLS compilers are as follows:

- there are no requirements for code annotation in sequential programs, which reduces the requirements for application developers, and sequential programs become universal;
- the conversion is based on the analysis of the data dependencies in the input program, rather than manual code annotation directives, similar to #pragma in Vivado HLS;
- for high-level synthesis and scaling of its results, the parameters of parallelism in the cadr structure are reduced rather than parallelized, as is typical for a sequential program;
- arbitrary memory access in a sequential program is converted to regular read/write operations for data streams;
- automatic adaptation of the application to the architecture and configuration of a user-defined RCS;
- automatic synchronization of data and control signals across all utilized FPGA chips;
- support for problems from various application domains.

However, the parallelizing compiler [7] has significant disadvantages:

1. Creation of only one type of computing structure – information graphs with parallelization by layers and iterations.
2. The expediency of creating procedural blocks in which nested conditional transitions and recurrent expressions are effectively implemented is ignored.
3. No means for integrating pipeline and procedural blocks within a unified computing structure, limiting the parallelizing compiler capabilities.
4. A parallelizing compiler does not provide rational generation of program constructs of parallel-pipelined programs. It reduces the efficiency of synthesized applications and requires a large number of iterations in a parallel-pipelined program, which leads to significant

translation time reducing the performance of synthesized applications and requiring extensive exploration of design alternatives, which leads to long compilation times.

In this regard, it is necessary to develop a new version of the compiler, which will implement a fundamentally different approach to converting a parallel program to a parallel-pipeline program, based on the use of performance reduction. Performance reduction is defined as a proportional decrease in performance in all fragments of the information task graph with a possible reduction in hardware costs for implementing the computing structure [8]. Performance reduction can be performed by: memory channels, hardware resources, data width, and frequency.

The use of performance reduction makes it possible to adapt the cadr structure to the available hardware resources of the target RCS. However, it can lead to inefficiency in the data flows of the cadr structure and significant decrease in the performance, which is unacceptable.

2. Synthesis of the Cadr Structure of a Parallel-pipeline Program

The operation of the parallelizing compiler consists of two stages: converting a sequential program to a parallel-pipeline program and adapting the parallel-pipeline program to the configuration of the target reconfigurable computer system (RCS).

At the first stage, the algorithmic scheme of a sequential program is transformed into an information graph, the vertices (subgraphs) of which correspond to data operations, and the edges represent data dependencies between them. Information-independent vertices are located in layers corresponding to the layers of the algorithm graph [9], while the information dependencies between subgraphs from different layers are reflected by iterations. The distribution of subgraphs by layers and iterations makes it possible to visualize the information dependencies between task fragments at different levels of the hierarchy.

In addition, the information graph is invariant to various forms of description of calculations in a sequential program. Therefore, sequential programs of the same task that differ in syntax and form are represented by the same information graph. By replacing the operational vertices with computing devices, and the arcs with the connections of the switching system, the information graph is transformed into a maximally parallel cadr structure [10] that combines the computing structure of the task and the rules for organizing data flows. The cadr structure is characterized by the number of layers and iterations, data width and number of devices in the base subgraph, data processing interval, latency and operation frequency. Together these parameters determine the degree of parallelism, execution time, and hardware resource utilization.

The cadr structures of sequential programs contain recursive expressions and nested conditional transitions, specific to individual problems of digital signal processing, symbolic processing, optimization and graph theory. For their formation, it is necessary to develop new methods for converting sequential programs into an information graph and methods for displaying it in the syntax of the COLAMO high-level language, taking into account the irregularity of information exchanges and the specifics of the problem being solved.

At the second stage, the methodology [7] for converting the cadr structures is used to adapt the cadr structure to the available resources of various RCS configurations.

At selecting transformations, the dynamics of the nonlinear variation of hardware costs $A_{CS} = (a_1^{CS}, a_2^{CS}, ..., a_n^{CS})$ of the cadr structure is analyzed as its main performance parameters

$P_{CS} = (p_1, p_2, ..., p_n)$ change, taking into account data dependencies both within and between subgraphs (task fragments).

The performance parameters of the cadr structure P_{CS} include: L_i^{it} is the number of data-independent subgraphs in the layer F_{ij}; It_i^{it} is the number of iterations in the function f_i; Ch_i^{it} is the number of external channels of the cadr structure; Op_i^{it} is the number of devices in the cadr structure FG^{it}; ρ_i^{it} is the data bit width, the method of hardware implementation of the base subgraph g_i and the data processing interval I.

Each of these parameters, depending on the subject area, structure, and problem dimension, can be crucial for the critical hardware resource of the transformed program [11].

For RCS based on FPGAs, the total number of analyzed variants of rational cadr structures with different parameters is relatively small; in the limiting case, for each base subgraph it does not exceed the number of permutations of performance parameters, which in the considered case equals 5!=120. As a rule, depending on the first variable cadr parameter, the remaining transformations of the cadr structure are determined.

The influence of performance characteristics on the hardware resources occupied by the cadr structure makes it possible to unify them in one view and present the transformation of the cadr structure to the available RCS resources (the local goal of transformations) as a search for rational values of performance parameters (the global goal of transformations).

The hardware cost A_{CS} of the cadr structure depends nonlinearly on the performance parameters P_{CS} : $A_{CS} = \Phi(P_{CS})$, where a single performance parameter p_i may simultaneously affect several hardware resource characteristics.

The main problem of cadr structure transformations is to search the performance parameters P'_{CS} that will be the solution of the system:

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

where A_{HCS} is the available hardware resource of the target RCS; $Perf(P'_{CS})$ is the performance of the cadr structure; $Perf_3$ is the specified level of real performance.

However, the performance of the cadr structure depends not only on the rational use of available hardware resources, but also on the organization of data flows.

If the computational resources are not sufficient to fully implement iteratively interconnected subgraphs in the cadr structure, then feedback will be generated at performing the performance reduction by subgraphs in the cadr structure. In this case, the processing of a single input data requires $M = \frac{N}{r}$ passes, where N is the number of subgraphs and r is the degree of performance reduction across subgraphs. This leads to a necessity to provide the input data with data initiation interval $S = N * Lat$, where Lat is the latency of the reduced subgraph.

Note that regardless of the degree of performance reduction r, the processing time of one data remains will not change. Therefore, according to [12], if there exists an information graph G, consisting of a set of interconnected isomorphic subgraphs $p_1, p_2, \ldots, p_N$, and the system resources are insufficient for hardware implementation of the entire graph G, then the maximum efficiency (the specific performance) is achieved through the hardware implementation of a single subgraph p_i.

The optimal structure of graph implementation if the available hardware resources are insufficient to implement all subgraphs F_2, is presented in *Fig.* 2.

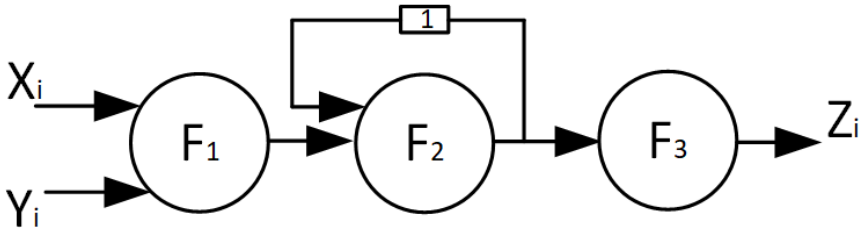


Figure 2. Optimal graph structure

To ensure a uniform data flow processing rate, the reduction transformations over the chain of F_2 subgraphs require to use the performance reductions by the value r and to subgraphs F_1 and F_3.

The use of performance reduction for iterative and procedural subgraphs leads to a significant decrease in performance, which is unacceptable at synthesizing cadr structures by a parallelizing compiler.

To improve the performance of cadr structures, a modified methodology for cadr structure synthesis has been proposed, consisting of the following steps:

1. Calculation of cadr structure 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 influence degree 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: $p_i : \psi(p_i) = max\ K_{cr}$. 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) $R^* = \|pi\|, R* > R$.

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 $A\prime_{CS} = \Psi(P\prime_{CS})$, $Perf(P'_{CS})$.

4.2. Evaluate the achievement of porting purpose from the conditions $A\prime_{CS} = \Psi(P\prime_{CS}) \leq A_{HCS}$ and $Perf(P'_{CS}) \geq Perf_3$.

4.3. If both conditions are performed, then go to item 7 to save the current parameters of the cadr structure

else: Analysis of alternatives for further reduction:

If $A\prime_{CS} > A_{HCS}$ the hardware costs exceed the available RCS resource, then:

If $R^* = R$, then

– increase the reduction coefficient $R :- R + \triangle$, where $\Delta = \left\lceil \frac{A'_{CS}}{A_{HCS}} - 1 \right\rceil$;

– restore the original value of the performance parameter $< P\prime_{CS} >= < P_{CS} > \leftarrow p_i$;

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

5. Induction (increasing of the previously reduced performance parameters) of the cadr structure

5.1. If the tuple of parameters $|S| = 0$, go to item 6.

5.1. Form the 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) $Ind^* = \varphi\,(Ind, p_k) < Ind$;

3) $Ind^* = pk$.

5.4. If increasing the parameter p_k leads to the increased data initiation interval of the cadr structure, then go to item 5.6.

5.5. Increase p_k by the selected step Ind^* : $p\prime_k = \Theta^{-1}(p_k, Ind^*)$ and save it into the tuple: $< P\prime_{CS} >= < P\prime_{CS} > \leftarrow p\prime_k$.

5.6. Go to item 4.

6. Performance improvement

6.1. Form a tuple of parameters S associated with changes in data initiation interval within the cadr structure $\langle p_{i-1}, \ldots, p_1 \rangle$. If $\mid S \mid = 0$, go to item 7.

6.2. Select the next element of the tuple p_k and determine the corresponding interval value s_k.

6.3. Determine the degree of influence of the element p_k on the data initiation interval s_k.

6.4. If the data initiation interval value $s_k > 1$, then increase the scaling factor by s_k for all parameters $p_t \notin S$.

6.5. Go to item 4.

7. Save the cadr structure

7.1. Add the current parameters of the cadr structure to the list, ordered by the minimum performance.

7.2. If $Perf(P'_{CS}) < Perf_3$ and there were other critical resources then select the next critical resource and go to item 1.1.

8. Select a reasonable variant for reduction of the cadr structure

8.1. Issue the first item of the list of cadr structures with the highest performance.

The considered methodology changes the parameters of the cadr structure in such a way as to ensure the rational use of available hardware resources and achieve a given level of real performance. If the target performance level $Perf_{req}$ is unattainable, the result will be a cadr structure with the highest performance level among those analyzed.

Unlike the methods of structural and procedural organization of calculations, which find a rational cadr structure for a predetermined and fixed resource, the transformation is a function that depends on the architecture and configuration of the computing system in this methodology. This ensures exploration of the cadr structure area not only “downward” toward reduced

parallelism (reduction), but also "upward" toward increased parallelism (induction) if hardware resources permit. Combining the reduction and induction makes it possible to define the rational performance parameters for cadr structures even when there is a shortage of hardware resources for the structural implementation of the basic subgraph.

The modified methodology for the synthesis of personnel structures allows for a finite number of permutations to find a solution that ensures the rational use of hardware resources. It also provides performance improvement of the cadr structure by reducing the complexity of data flow organization by converting the cadr structure to efficient calculation forms.

3. Proposed Performance Increasing of the Cadr Structure

The modification of the cadr structure synthesis methodology is caused by the fact that the cadr structure performance depends not only on the rational use of the available FPGA hardware resources in a reconfigurable computer system, but also on the data initiation interval. Data initiation interval depends on the processing intensity in different subgraphs of the cadr structure.

Let the computing structure consist of a set of subgraphs $P = \{p_1, p_2, \ldots, p_n\}$, where the intensity of input and output data flows is defined as $p_i = \langle s_{in}, s_{out} \rangle$ for each subgraph p_i. The data initiation interval of a subgraph p_i is calculated by the formula $S = \frac{1}{v}$, where $v = \frac{S_{out}}{S_{in}}$. The total data initiation interval of the cadr structure is determined by the maximum interval among all its subgraphs. Data initiation interval leads to the equipment downtime and, as a result, reduced performance. Methods for converting computing structures can be used to reduce the data initiation interval: nested pipelining (pipeline-to-pipeline) [13], macro-pipelining [14], and auto-substitution [15].

If the cadr structure contains several data-independent subgraphs $G = \{g_1, g_2, \ldots, g_n\}$ with feedback, the most efficient "pipeline-to-pipeline" transformation can be performed to increase specific performance. The "pipeline-to-pipeline" transformation reduces the subgraphs G by a factor r, replacing the reduced subgraphs with registers in the feedback loop. Maximum efficiency is achieved at $r = Lat$, where Lat is the latency of subgraph g_i, provided that $\mid G \mid \geq r$. In this case, instead of feeding a single datum into subgraph g_i, a data packet can be supplied, representing a slice of operands entering the reduced subgraphs G. The length of the operand packet depends on the number of registers in the feedback loop.

Let the input data flows to the subgraphs G have the form:

$$\langle x_0^n, \ldots, x_0^2, x_0^1, x_0^0 \rangle$$

$$\langle x_1^n, \ldots, x_1^2, x_1^1, x_1^0 \rangle$$

$$\langle x_2^n, \ldots, x_2^2, x_2^1, x_2^0 \rangle$$

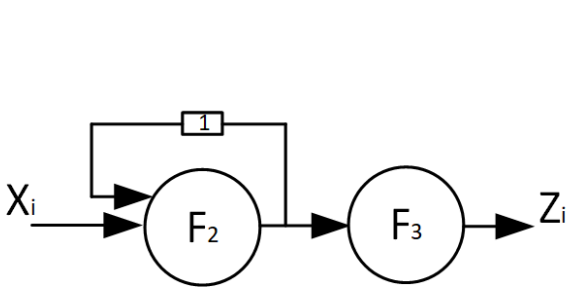
$$\vdots$$

$$\langle x_{r-1}^n, \ldots, x_{r-1}^2, x_{r-1}^1, x_{r-1}^0 \rangle$$

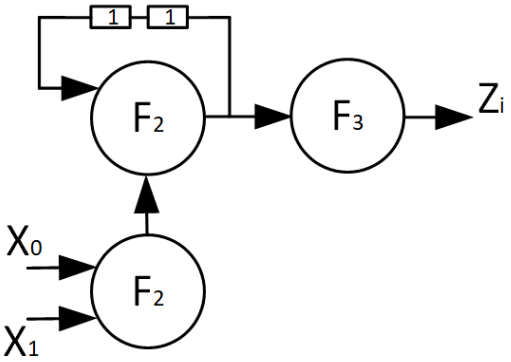
Then, the data packets for subgraph g_i will consist of r elements and have the form:

$$\langle x_{r-1}^0, \ldots, x_2^0, x_1^0, x_0^0 \rangle, \langle x_{r-1}^1, \ldots, x_2^1, x_1^1, x_0^1 \rangle, \ldots, \langle x_{r-1}^n, \ldots, x_2^n, x_1^n, x_0^n \rangle$$

The data initiation interval of the reduced cadr structure will decrease proportionally, becoming $S = \frac{Lat}{r}$. The "pipeline-to-pipeline" transformation does not reduce the task execution time, but it significantly reduces hardware costs by a factor of r and increases specific performance. The released hardware resources can be used to implement the other subgraphs of the cadr structure. If the cadr structure lacks multiple data-independent subgraphs G, i.e., $\mid G \mid = 1$, the "auto-substitution" method is appropriate for recursive procedures and recurrent expressions. In general, an expression of the form $x_i = f(x_{i-1})$ can be rewritten as $x_i = f(f(x_{i-2}))$. By expanding the expression and performing this transformation with the depth k, the feedback length and consequently the data initiation interval of linearized recurrent functions can be reduced by k times. For example, we consider an iterative process, described by the formula $x_{i+1} = ax_i + b$. Expressing x_i through x_{i-1} and substituting it into the original expression, we obtain $x_{i+1} = a(ax_{i-1} + b) + b$. Continuing substitutions for x_{i-2}, x_{i-3} and g.e., a formula expressing x_{i+1} in terms of any $x_{i-r} : x_{i+1} = a^{r-1}x_{i-r} + b + c$. As shown in [16], in the case of a linear function describing the iterative process, the coefficients can be simplified and calculated for any auto-substitution depth without intermediate steps. Thus, the latency of the iterative procedure can be reduced depending on the depth of the performed auto-substitution, which leads to a significant reduction in interval for linearized recurrent functions (Fig. 3).



(a) Computing structure before transformation



(b) Computing structure after transformation

Figure 3. Application of the "auto-substitution" method

The "auto-substitution" method leads to a certain increase in hardware costs for implementation of the computing structure.

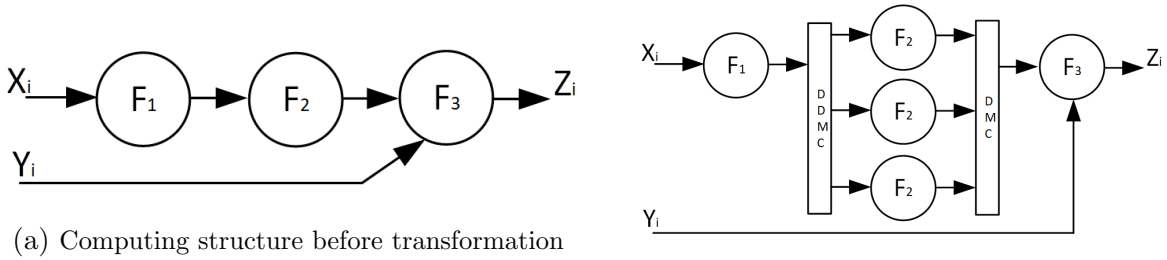
If a subgraph describes a non-linearized recurrent computational function, or if the structural implementation of the subgraph is inefficient (e.g., deep nesting of conditional branches or indirect addressing), then, according to [4], it is advisable to replace the computing structure of the subgraph with a procedural block. Procedural blocks are characterized by the number of clock cycles required to process a single operand, rather than the latency typical for pipeline blocks. In this case, the cadr structure can consist of a combination of structural and procedural blocks, which can result in varying data processing intensities between blocks and increased data initiation interval in the cadr structure.

To reduce the execution time of the procedural block, it is advisable to develop a macro-pipeline as proposed by V.M. Glushkov [14]. The essence of macro-pipeline development is scaling the procedural block by a factor L, where each procedural block sequentially receives data at intervals of $\frac{T_p}{L}$, with T_p being the execution time of the procedural block. Maximum efficiency

of the macro-pipeline is achieved when L equals the procedural block execution time T_p. In this case, the next data element from the stream is supplied to a free procedural block on every clock cycle, ensuring a dense data flow.

Implementation of a macro-pipeline does not require additional memory channels, but it does require additional hardware proportional to the value of L.

Let the execution time of subgraph F_2 be equal to T; then the data initiation interval in the computational structure (see Fig. 4a) should also be equal to T.



(a) Computing structure before transformation

(b) Computing structure after transformation

Figure 4. Application of the “macro-pipeline” method

If sufficient hardware resources are available to scale subgraph F_2 by a factor T, then the data output from subgraph F_1 can be fed to a free F_2 subgraph on every clock cycle, ensuring a dense data flow. The data flow for device F_3 is generated according to the order in which data is received at the F_2 subgraphs. The application of the discussed methods for converting the computing structure is implemented within a unified cadr structure synthesis algorithm, a generalized diagram of which is shown in Fig. 5.

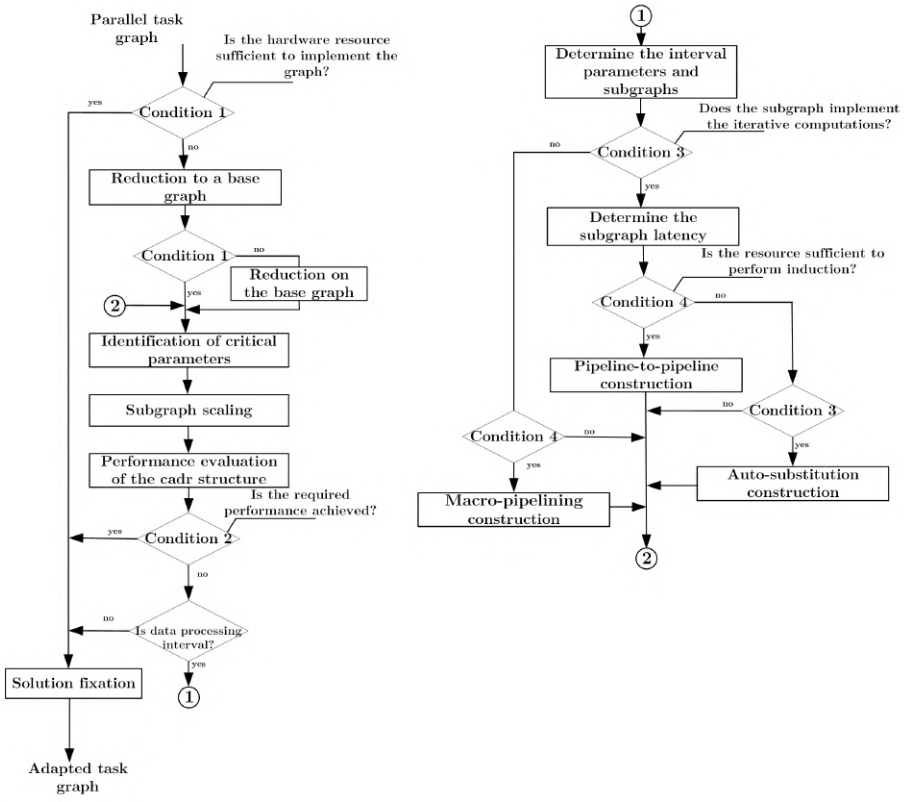


Figure 5. Diagram of the solution search for the cadr structure

The selection of the first transformation of the cadr structure depends on the type of critical resource (external memory channels or hardware resources). The subsequent transformations depend on the current hardware costs and performance parameters at each transformation stage. If the data initiation interval is in the synthesized cadr structure, then transformations must be performed to reduce it.

The combining of the solution search procedures (reduction and induction transformations) with performance improvement procedures into a single synthesis scheme – or using performance improvement procedures as an intermediate step – can lead to repeat the re-adjustment of previously determined scaling parameters of the cadr structure subgraphs and may exceed the limits of the permissible hardware resources of the target RCS. Therefore a two-pass organization of the synthesis is necessary for an efficient cadr structure of the parallel-pipeline program, which cannot be implemented within the current structure of the parallelizing compiler.

4. "Theseus" Parallelizing Compiler

To implement a new trajectory selection scheme at searching effective cadr structures, a new architecture of the "Theseus" parallelizing compiler is proposed, shown in Fig. 6.

The parallelizing compiler was extended with the "Ariadne" cadr structure synthesis program for parallel-pipeline programs and the interaction architecture between programs was modified to repeat invocation of programs for evaluating hardware costs at any stage of cadr structure synthesis and for generating different representations of the parallel-pipeline program depending on cadr structure parameters.

The "Ariadne" cadr structure synthesis program is designed to synthesize efficient implementations of cadr structure transformations for various application domains. In general, the operation of the "Ariadne" component consists of the following stages:

1. Generation of a universal parallel-pipeline program by the "Centaur" component.
2. Evaluation of the impact of scaling coefficients on the hardware costs of the computing structure.
3. Search for the efficient scaling parameters of the basic subgraphs of the task graph, with the calculation of the performance of the resulting solutions.
4. Performance improvement of the obtained solution.
5. Selection of the most efficient solution and generation of a specialized parallel-pipeline program based on it by the "Centaur" component.

The "Ariadne" operation begins with the creation of a universal parallel-pipeline program by the "Centaur" program. The universal parallel-pipeline program contains constants for describing the scaling coefficients of computing structures and memory channels, transformed computing structures into a parallel-pipeline form, as well as mechanisms for matching unbalanced data flows and synchronization elements with each other. As a rule, channel matching is required for adjacent subgraphs, and data synchronization for correct processing of data flows both within the subgraph and between them is caused by data displacement in the channels.

The number of switching structures can be large, and their analysis and processing require significant CPU time during the program translation. However, it is a necessary to assess the impact of scaling coefficients on hardware costs of computing structure, memory channels, and the data initiation interval in the channels. Based on the obtained coefficients, a search is performed for possible solutions (a set of generated values for the program scaling coefficients),

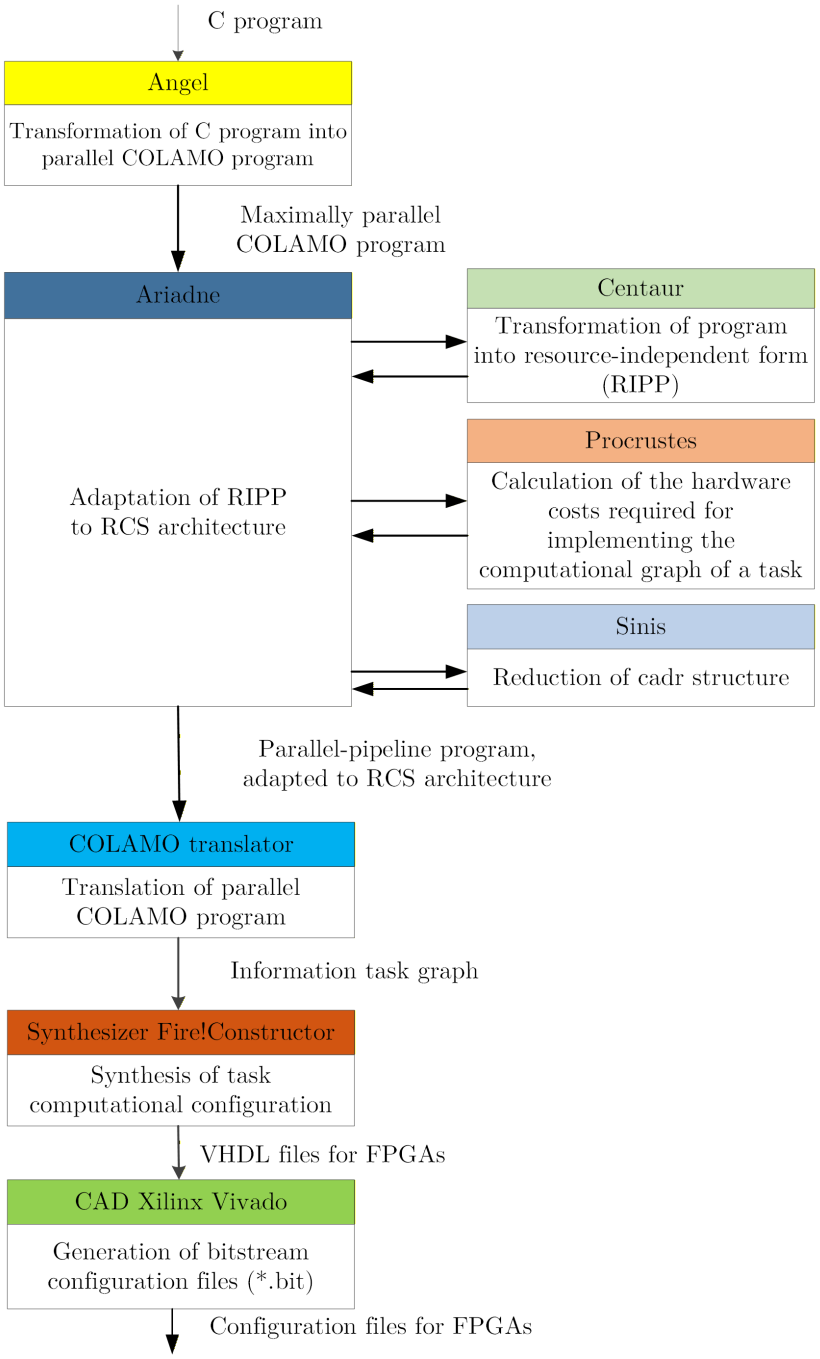


Figure 6. Parallelizing compiler architecture

the cadr structures of which satisfy with the available hardware resources of the target RCS. The resulting combinations represent potential parameters for creating the cadr structure.

Next, a performance evaluation is performed for each result. If the obtained results do not respond the given performance, then transformations are performed to increase the productivity of the cadr structure.

For each obtained solution, a sequential analysis of the scaling (reduction) coefficients is performed, corresponding to subgraphs in which input and output data flow intensity does not correspond to each other. For each subgraph, data dependencies are analyzed both within the subgraph and within adjacent subgraphs, and an efficient transformation method is determined

(pipeline-to-pipeline, auto-substitution, or macro-pipeline). The transformation begins with the subgraph exhibiting the maximum data initiation interval. After completing transformations for the current subgraph, the interval of the cadr structure is analyzed before transfer to the next subgraph.

Let us consider an example to illustrate the synthesis process of a parallel-pipeline program. Suppose that for a parallel program, the information graph of which is given in Fig. 7, the channels of external memory are a critical resource. Let us assume that the total number of all external memory channels should not exceed 10.

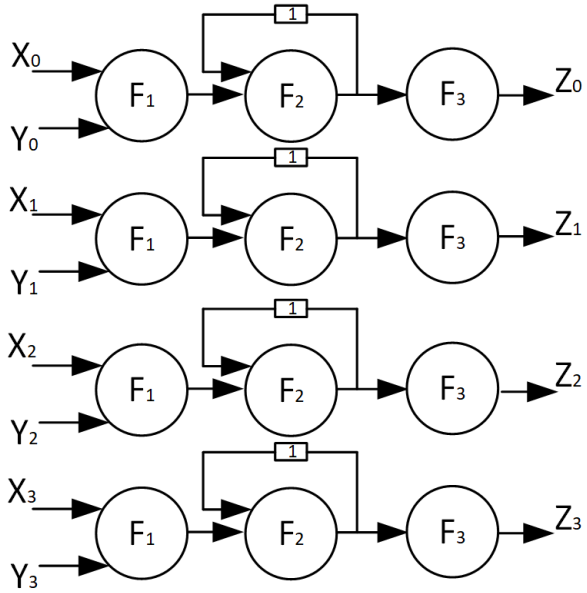
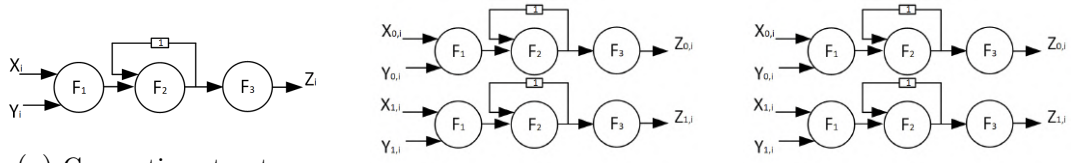


Figure 7. Computing structure of the parallel program

As a result of the synthesis, three variants of cadr structures will be obtained (Fig. 8), ensuring balanced data processing in data channels.



(a) Computing structure before transformation

(b) Computing structure after transformation

(c) Computing structure after transformation

Figure 8. Variants of cadr structures for different scaling factors

If all obtained variants of the cadr structures correspond to the specified performance level, the final solution will be the cadr structure with the maximum performance. In this case, the cadr structure given in Fig. 8c is obtained.

If the obtained cadr structures do not provide the required performance, a search for performance-improving solutions will be launched for each cadr structure. All solutions in Fig. 8 have iterative dependencies at calculating the next data input to the subgraph F_2. This indicates the necessity to organize the data initiation interval for input to subgraph F_2. The pipeline-

to-pipeline cadr transformation method can be used to increase the performance in the cadr structures with iterative dependencies.

Let us consider the pipeline-to-pipeline transformation using the cadr structure given in Fig. 8a as an example. Let the data initiation interval of the functional device correspond to its latency and be equal to 4 cycles. Then all subgraphs and external memory channels should be scaled according to the latency of the subgraph F_2, and markings in the graph is performed defining the application field of the pipeline-to-pipeline method. The cadr structure with a scaling factor of 4 and the logical elements of the method application are given in Fig. 9.

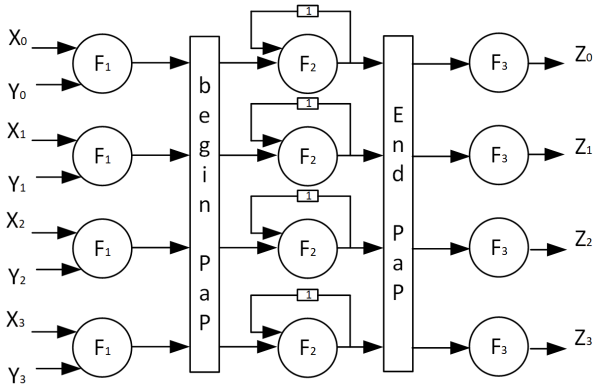


Figure 9. Cadr structure synthesized by the “Ariadna” program

The “Begin PaP” and “End PaP” serve as syntactic brackets for describing subgraphs for which the pipeline-to-pipeline transformations are applied, and they appear as directives in the parallel-pipeline program. Based on the placement of the “Begin PaP” and “End PaP” blocks in the information task graph, the Fire!Constructor synthesizer (Fig. 7) will transform the cadr structure into a single subgraph F_2 with a set of registers in the feedback loop. The modified cadr structure produced by the Fire!Constructor synthesizer is given in Fig. 10.

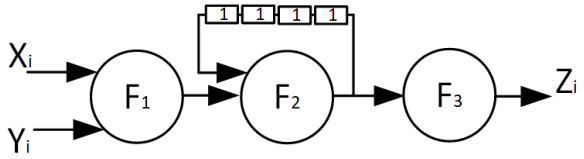


Figure 10. Modified graph using the pipeline-to-pipeline method

The scaling by a factor of 4 did not lead to an excess of elements in the memory channels and ensures correct data delivery to the computing structure. Note that the application of this approach to the cadr structures in Fig. 8b, Fig. 8c will lead to an excess of the number of elements in channels X, Y, and Z, and incorrect effect at data processing in the computing structure. Such transformation variants of the cadr structure will be discarded and not be considered as potential cadr structure options.

The “auto-substitution” method can be applied for subgraph F_2. In this case, additional solution variants for computing structure in Fig. 8 will be obtained, among which the solution with the maximum performance will be selected.

The stages of transforming the cadr structure of the input parallel program are not fixed and are largely determined by the information task structure and the type of information

dependencies. Thus, a number of symbolic processing problems are characterized by the absence of iterations, but the number of layers is extremely large. Therefore, transformations of the iterative structure for such problems are meaningless.

At the same time, in mathematical physics problems, the rational implementation of the iterative component of the cadr structure is crucial for the efficiency of the overall solution. In addition, the RCS hardware resource may not be sufficient to implement even the basic subgraph in some cases. This requires the application of the reduction transformations described in [8] to the basic subgraph, but it allows to subsequently use scaling methods and algorithms to the reduced frame structure to reduce the time for problem solution.

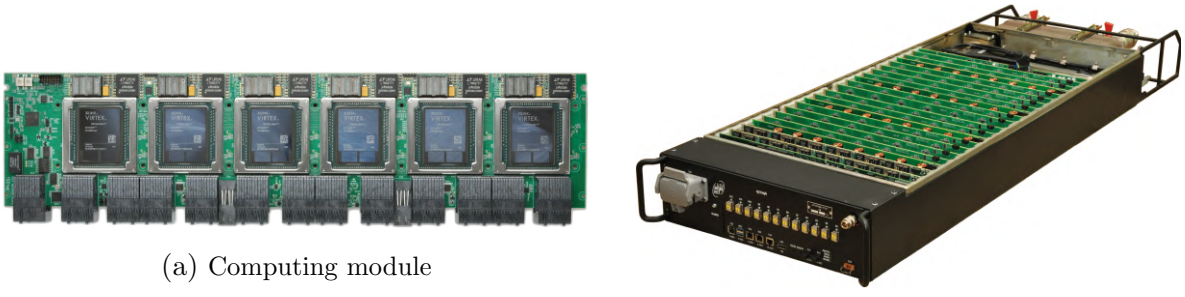
Next, the "Centaur" component generates a specialized pipeline-parallel program, based on the selected solution. The specialized application contains data synchronization and alignment fragments that are strictly necessary, based on the obtained scaling coefficient values. The output of the "Ariadna" program is a pipeline-parallel program adapted to the target architecture and configuration of the reconfigurable computer system, significantly simplified compared to a similar program produced by the previous version of the "Theseus" compiler [7]. This leads to a significant reduction in the total synthesis time of FPGA configuration files of multichip RCS.

In general, the efficiency of the generated pipeline-parallel program depends on the specifics of the information task structure and data flows, which may differ significantly in different task classes, as well as on the available hardware resources of the target reconfigurable computer system.

5. Results of Parallel-pipeline Program Synthesis

The analysis of the effectiveness of the proposed methods for performance increasing of parallel-pipelined programs, created for reconfigurable computer systems, was performed for a number of model problems. They were tested for operability by launching on the high-performance reconfigurable computer system "Arktur" (Fig. 11).

The "Arktur" computing unit, with a performance of 230 TFLOPS, contains 16 computational modules, 96 Virtex UltraScale+ XCVU37P FPGAs, 4GB of RAM, and 768GB of HBM memory.



(a) Computing module

(b) Computing unit

Figure 11. The RCS hardware for testing "Arktur" applications

The "Arktur" RCS is a universal reconfigurable computer system developed by the "Scientific Research Center of Supercomputers and Neurocomputers" (Taganrog, Russia). It is designed to solve problems in various domains: mathematical physics, digital signal processing, symbolic processing, and artificial intelligence.

Two classes of problems have been selected as model problems: linear algebra and symbolic processing. The implementation of problems from different domains makes it possible to verify the effectiveness and correctness of the modified methodology for synthesis the cadr structures and performance-enhancing methods, implemented in the "Ariadna" application. The effectiveness of the synthesis of cadr structures and the method of increasing productivity "auto-substitution" have been tested on linear algebra problems: the problem of solving systems of linear algebraic equations (SLA) by the Gauss method and the Gauss-Seidel method.

The effectiveness of cadr structure synthesis and the performance-enhancement method "autosubstitution" were tested for linear algebra problems: the specifically solving systems of linear algebraic equations (SLAEs) using the Gauss method and the Gauss-Seidel method.

Note that the Vivado HLS-synthesized solution for the Gauss model problem contains a single iterative step, compared to 3512 steps at the problem translation using the "Theseus" parallelizing system. Using manual code annotation with #pragma directives in Vivado HLS allowed obtaining a solution for two iterative steps of the forward elimination algorithm. It is still significantly lower than by the parallelizing compiler. It can be noted that there is a slight decrease in the number of pipeline steps at synthesizing a solution using the performance enhancement methods and the "Theseus" parallelizing compiler, compared to the creation of a specialized solution by application programmers – 4210.

The efficiency of SLAE solution by the Gauss-Seidel method depends on the available hardware resources; insufficient resources lead to feedback loops in the computing structure. At the same time, calculations in feedback have associative properties, which allows using the "auto-substitution" performance-enhancing method. The solution, obtained as a result of using the "auto-substitution" method, contains 5376 data processing steps and the data initiation interval of 21. This is inferior to the specialized solution, obtained by application programmers – 6563, but significantly exceeds the data initiation interval of the existing version of the compiler by 1.7 times.

The performance results for the linear algebra model problems using the parallelizing compiler, application vendors, and the HLS compiler (Vivado HLS) is presented in the Tab. 1. Performance was defined as the number of basic graphs in synthesized solutions and the data initiation interval value.

Table 1. Performance results for model problems in mathematical physics

Researches	Gauss	Gauss–Seidel
Porting time, sec	5073	7752
	Number of graphs / interval	
The existing version of the "Theseus" compiler	3512/1	5380/36
Application vendors (application of performance increasing methods)	3512/1	5376/21
Application vendors (specialized solution)	4210/1	6563/17
HLS compilers	2	-
Level of real performance	0.68	0.81

The effectiveness of using methods for converting cadr structures has been evaluated on symbolic processing problems: the encryption problem, the SSL authentication problem, and

the password hashing problem in the FreeBase database a macro conveyor and a pipeline-to-pipeline.

The DES encryption problem showed that the number of implemented subgraphs by the parallelizing compilers coincided, as the cadr structure did not contain procedural blocks or feedback loops. For the synthesis of the SSL authentication cadr structure, based on the MD5 and RC4 algorithms, the "macro-pipeline" performance-enhancing method was used. The scaling degree of the procedural subgraph implementing RC4 by the "Theseus" parallelizing compiler was 1253 subgraphs in a single pipeline, compared to 970 subgraphs implemented by application vendors. This indicates to a more efficient implementation of procedural calculations in the RC4 block. In contrast, Vivado HLS implemented only a single subgraph. The data initiation interval in the first two cases was 1, the Vivado HLS solution has 2100 cycles.

Due to the insufficient available hardware resources, the feedback loop with the latency of 12340 cycles has been implemented in the cadr structure of the password hashing problem in the FreeBase (MD5-Crypt) database. As a result of the problem implementation by application vendors using the "pipeline-to-pipeline" performance increasing method, the latency of the feedback subgraph implemented by the parallelizing compiler was reduced to 8870 cycles, which is 1.6 times higher than that achieved by application vendors (5544 cycles). This leads to the decrease of specific performance.

The results obtained for symbolic processing problems are presented in the Tab. 2.

Table 2. Performance results for model symbolic processing problems

Researches	DES	MD5-RC4	MD5-Crypt
Porting time, sec	28952	53779	36095
		Number of graphs / interval	
The existing version of the "Theseus" compiler	1335	106/1253	112/132056
Application vendors (application of performance increasing methods)	1335	7518 (6 pipe)/1	26210 (3 pipe)/24
Application vendors (specialized solution)	1740	6790 (7 pipe)/1	22176 (4 pipe)/24
HLS compilers	1	1/2100	-
Level of real performance	0.76	0.9	0.84

Solving MD5-RC4 and MD5-Crypt problems by application vendors and using pipeline-in-pipeline and macroconveyor methods and specialized solutions shows that the application vendors perform more efficient implementation of subgraphs in feedback, despite achieving the same data flow rate in the cadr structure. This results in lower latency of all feedback and reduced hardware costs. The released resource was directed by application vendors to implement an additional pipeline in the computing structure, which increased specific performance by 1.16 and 1.33 times, respectively.

Researches performed on model performance problems of synthesized solutions, presented in Tab. 1 and Tab. 2, have shown that the application of the presented methods to increase the productivity of cadr structures the pipeline-to-pipeline, auto-substitution and macro-pipeline demonstrates the productivity increasing up to 234 times on model problems by various classes, depending on the type and computing structure of the problem.

Comparing the results obtained for model problems using the previous version of the parallelizing compiler, the new compiler version, and application vendors demonstrates the significant advantage of the new version of the "Theseus" parallelizing compiler.

Conclusion

The differences between the proposed version of the "Theseus" automatic parallelizing compiler and the most well-known Xilinx Vivado HLS and Vitis HLS compilers are fully automatic transformation of the input program without requiring manual code annotations (e.g., #pragma directives), support for complex parallelism forms (macro-pipeline, pipeline-to-pipeline) unlike the previous version of the "Theseus" compiler, automatic scaling of solution to a given RCS resources and synchronization of data and control signals.

The "Theseus" compiler will significantly reduce the conversion time of sequential programs to parallel-pipeline solutions for reconfigurable computer systems with multiple FPGA modules connected via a spatial communication network, while ensuring a guaranteed level of real performance. Due to the efficient organization of calculations and rational use of the available FPGA hardware resource, the "Theseus" compiler will provide significantly higher real performance of the reconfigurable computer system compared to the similar compilers.

The new version of the "Theseus" parallelizing compiler will enable to significantly expand the class of problems to be solved. Thanks to the efficient use of the target reconfigurable computer systems hardware resources and the transformation of the CADR structure into effective computational forms (pipelinetopipeline, autosubstitution, and macropipeline), the systems actual performance is guaranteed to reach at least 60% of its peak capacity.

Future research directions for the "Theseus" compiler include expanding the classes and ranges of solving problems and reducing requirements on C-code, in particular, support for dynamic memory allocation for arrays, data loading/unloading, and support of structures and unions.

Acknowledgements

The study was supported by the Russian Science Foundation grant No. 26-11-00209, `https://rscf.ru/project/26-11-00209/`.

References

1. Guzik, V.F., Kalyaev, I.A., Levin, I.I.: Reconfigurable computer systems. SfedU Publishing, Taganrog, 2016. 472 p.

2. Hauck, S., DeHon, A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann, Burlington, MA (2008).

3. Nane, R., *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`

4. 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

5. Kamkin, A.S., Chupilko, M.M., Lebedev, M.S., *et al.*: Comparison of High-Level Synthesis and Hardware Construction Tools. Trudy ISP RAN/Proc. ISP RAS 34(5), 7–22 (2022). https://doi.org/10.15514/ISPRAS-2022-34(5)-1

6. Liang, Y., Rupnow K., Li Y., *et al.*: High-Level Synthesis: Productivity, Performance, and Software Constraints. Journal of Electrical and Computer Engineering 2012, 649057, (2012). https://doi.org/10.1155/2012/649057.

7. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: High-Level Synthesis Toolchain "Theseus" for Multichip Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations 10(2), 18–31 (2023). https://doi.org/10.14529/jsfi230202

8. 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

9. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A. High-Level Synthesis Software for Multicrystal Reconfigurable Computing Systems. Bulletin of the South Ural State University. Series: Computational Mathematics and Computer Science 11(3), 5–21 (2022). https://doi.org/10.14529/cmse220301

10. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: High-Level Synthesis Strategies for Multicore Reconfigurable Computing Systems. In: Voevodin Vl.V. (eds.) Supercomputer Days in Russia, Proceedings of the International Conference, Moscow, September 29-30, 2025. pp. 195–206. MAKS Press, Moscow (2025). https://doi.org/10.29003/m4750.978-5-317-07451-7

11. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: Software Complex for High-Level Synthesis of Configuration Files for Multicrystal Reconfigurable Computing Systems. In: Parallel Computing Technologies (PaCT'2023): Short Papers and Posters. Proc. of the 17th All-Russian Scientific Conference with Int. Participation, Saint Petersburg, March 28-30, 2023. pp. 133–142. South Ural State University Publishing Center, Chelyabinsk (2023).

12. Ivanov, A.I., Konovalchik, P.M.: Methods of organizing parallel-pipeline calculations for solving computationally intensive problems. Information Technologies 12, 38–43 (2004).

13. Kotlyarov, A.S., Levin, I.I.: Processing of Network Data Streams in Reconfigurable Computing Systems, Izvestiya SFedU. Engineering Sciences 2(204), 48–56 (2019).

14. Glushkov, V.M.: On Macro-Pipeline Computations. Computational Processes and Systems. Issue 1. pp. 61–70. Nauka, Moscow (1983).

15. Koughi, P.M. Architecture of Pipelined Computers. Radio and Communications, Moscow (1985).

16. Mikhailov, D.V., Levin, I.I.: A method for determining the stability of a recursive adaptive pipelined filter. Radio Engineering 3(6), 5–11 (2020). https://doi.org/10.18127/j00338486-202003(06)-01