VISUAL PROGRAMMING TOOL FOR COMPUTER WITH DATA FLOW COMPUTATION CONTROL

Branislav MADOŠ, Ján HURTUK, Marek ČAJKOVSKÝ, Erik KUDRA
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/602 3023,
e-mail: {branislav.mados, jan.hurtuk, marek.cajkovsky}@tuke.sk, erik.kudra@student.tuke.sk

ABSTRACT

The paper presents the brief overview of the design and implementation of visual programming tool for visual creation of data flow graph (DFG) and its automatic transformation to the program code for computer with the data flow computation control that is designed at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics, Technical University of Košice.

Keywords: *visual programming, dataflow computer, dataflow graph*

1. INTRODUCTION

Today most common computer architectures are based on the John Von Neumann's principles and are using the control flow model, where computation control is using the flow of instructions.

In most of architectures of control flow computers the next instruction for execution is pointed by the special register called program counter (PC) and this instruction is executed even in the situation when needed operands of instruction are not present. Control flow computing model has the serious limitations in exploiting parallelism in the phase of the program code execution which is one of main disadvantages in comparison with data flow control model.

Data flow control of computation on the other hand controls the flow of program execution not by the flood of instructions, but by the flood of the data. The program code is created in the form of the data flow graph (DFG) that is describing the data dependencies between instructions in the program.

In the dataflow paradigm, the instruction is executable only when operands of the instruction are present and data dependencies between instructions ensure the proper order of instructions execution. Dataflow computation control has main advantage in the possibility to exploit the inherent parallelism on the instructions level in time of the program execution.

In 2009 a prototype of the dataflow computer was developed at the Department of Computers and Informatics, FEI, TUKE [1] [2] [3][4][5].

Prototype was verified and tested with use of the FPGA technology. Xilinx WebPackISE software and the Xilinx Spartan 3AN Development board were used. Dataflow program for this computer is represented by the binary file that consists of 128b vectors, which are representing program instructions.

To allow comfortable program code creation, visual programming tool was designed, that allows creation of dataflow graph in visual form and transforms DFG into the textual form.

Next parts of this paper are describing design and basic features of the developed CASE tool.

2. DESIGN AND IMPLEMENTATION OF CASE TOOL FOR PROGRAMMING OF DATAFLOW COMPUTERS

This part of the paper describes the design and implementation of the CASE tool for programming of dataflow computers. The structure of this chapter is divided into sub chapters that chronologically describe the process of the CASE tool creation.

2.1. Design of the dataflow graph

The main feature of the CASE tool is drawing of the dataflow graphs. The advantage of drawing and visualization of graphs versus writing program in the form of the source code in textual form is in easy understandability by users as well as in less chance of committing syntax errors. This chance is reached by the possibilities and limitations of applications graphics editor. CASE tool offers a possibility to program a simple dataflow graph. The word "simple" means, the graph does not contain any primitives for generate predicates and copying of instruction operands. The graph does not contain any switch and merge primitives, because these primitives need predicates to properly work. On the other side, dataflow graph consist of these primitives:

- Operating nodes
- Input data
- End elements
- Connecting lines

Operating nodes can be divided into two groups. The first group is a group consisting from binary operations (*addition, subtraction, multiplication and division*). The second group consists only from one unary operation (*negation*). Because operations have to work with some data, for this purpose, the application disposes with second group of primitives - Input data. Input data can be constant or variable. In the application's dataflow graph, constants represent numbers. Variables are for defining places where the data will be entering into the graph from outside.

The ending element is the last group of graph's primitives. This element does not influence the final source code for a dataflow computer. Recursive algorithms which run behind the scene of a program are using these ending elements to determine the first node of dataflow graph. Input data and operating nodes have to be connected to each other. For this reason the application contains connecting lines, which are representing edges in data flow graph. These four groups of graph primitives contain enough components for successful creation of dataflow graph.

2.2. Design and implementation features of the application

Features of CASE tool are closely connected with dataflow graph, which was described in previous subchapter. The features are divided into the groups as well.

2.2.1. Features of graphical editor

The CASE tool belongs into the group of graphical programming tools. The first and most prominent functionality of the tool is a graphical editor. In this part of the application it is possible to visually generate a dataflow graph. The editor includes the following main features:

- *Inserting nodes* – the user can select a type of node. Next with the help of mouse's cursor put it to the desired position in the scene of graphical editor.
- *Making edges (connecting lines)* – with these edges the user can make connections between nodes of dataflow graph. One of the features of the editor is the ability to control the validity of connections. It means, for example that the user cannot create the loop in the graph.
- *Moving nodes and edges* – they give a possibility to move items from one position to another for better visualisation of dataflow graph. Edges are moving automatically with the connected nodes.
- *Removing nodes and edges* – the user can remove nodes or edges from the graph in editor as well as insert them. If the user erases a node with the edges, these will be deleted too.
- *Colouring sub-graphs* – the user can change colour of sub-graph for better visualisation of partial solution.
- *Zooming* – the user can zoom in or out a scene visualized in the graphical editor.
- *Group selection* – there is the possibility to select more than one node or edge and subsequently edit this group of elements, it means delete or move entire selected sub graph.

Second part of the graphical editor is a list of instructions. This list contains information about activating framework in a "smart" form in which user can read information better than in normal 128 bits binary vector form. Smart representation of instruction has the following form:

<Address>_<Description>_<Left Input>_<Right Input>

Every node has an address. The address is integer number from interval <0, 126> because target dataflow architecture accepts only 127 nodes.

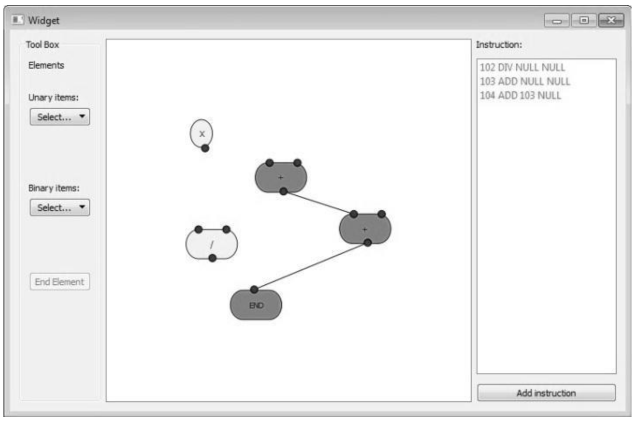
Each of implemented instructions has the description for better identification of the type of instruction. As the proof of the concept developed CASE tool supports these types of instructions: addition (ADD instruction), subtraction (SUB instruction), multiplication (MUL instruction), division (DIV instruction) and negation (NEG instruction).

Finally the instruction node has the left and a right input. These two fields contain information about edges. When the left or right input pin is free (free means no connection to another node), the left or right input will contain "*NULL*" string. When a node is connected to a binary or unary operation via the edge, the application uses an address of that instruction for the left or right input.

If the node has a connection to constant 123 on the left pin, the left input will consist from a prefix *Const* and next value of constant in parentheses. In this case it is *Const(123)*. On the other side the application uses prefix *Var* for variable node and name of variable is written between parentheses.

The list of instructions contains all binary and unary instructions which are used in a graphical editor. It means, each binary or unary node used in graphical scene has one line in list of instructions. The user can as well use it to find instructions in the graphics editor. Constants, variables and "END" nodes are not included in the list of instructions because their presence in this form is not relevant for the dataflow computer. Implemented graphical editor with the list of instructions in the smart form is shown in the Fig. 1, where is shown the graphical editor (the biggest part in the middle of the figure) with the dataflow graph and the list of instructions (part which is located on the right side of the figure).

Editor of the dataflow graph is capsuled in the main window of the application and is unique for every one project which is opening it.



102 DIV NULL NULL
103 ADD NULL NULL
104 ADD 103 NULL

Fig. 1 Implemented graphical editor with the list of instructions

2.2.2. Features of the main window

Features of the main window are including all of the functionality which helps the right operation of the CASE tool application. These features also help to the intuitive work of the program. Further points describe the most important features which are implemented in the application:

- Creation of the new projects and work with them. Users are able to create projects in parallel with the use of more than one graphical editor.
- Load and save of the projects to and from the disk. Users can save non-finished projects to the disk and continue in work. It is also possible to open recent files from the main menu in the application.
- Add instruction by wizard. Wizard is the feature, which helps the user to add instructions directly inside to the list of instructions. Wizard helps to keep the right syntax of instructions of the whole dataflow graph.
- Show properties of activation framework. Activation framework is represented as the 128 bits binary vector and to find part of information hidden inside this vector is very hard. For this reason the application contains features to show this information in readable form, also with highlighting of the text in different colours.
- Export code for target dataflow computer. This code is the result product of the CASE tool. Users can find this code in the special dialog.

Additional features for the improved work with the application are no less important. For example users can show or hide elements of the application to increase space of graphics editor. Instructions have a colour which indicates the connected state. Connected state occurs when users connect instruction to the main dataflow graph (main dataflow graph includes end element in its body). The application contains key shortcuts and many tips as well. At last but not least, users can use the application under several of resolutions of the screen.

2.3. Design algorithm unique for application

The application computes a lot of calculations during the process of creating data flow graph. Many of these calculations are very easy, but for some of them the application uses more difficult algorithms. The following subchapters take a closer look to these algorithms.

2.3.1. Algorithm for graph align

Users can lose the sight over the complex data flow graph. Algorithm for graphs align makes the complex dataflow graph better readable. In connection with graph the term level was established to describe where a node appears vertically. The level of a node in a graph is given by the number of links you have to travel up from the *END* element in order to reach it. The *END* element is therefore at level 0 (no links need to be followed), and its children nodes are at level 1. Their children nodes are at level 2, and so on. The level of a node is therefore the same as the y-coordinate where it will be drawn in graphics scene. To make it easier on ourselves, and to avoid having to detect whether one node bumps into or overlays another, in algorithm is written a rule that there will only be one node per vertical grid line. This may mean that the graph is a little too spread out horizontally, but it is a whole lot easier to deal with. What this means in practice is that the *END*'s x-coordinate will be equal to the total number of nodes in its left sub-graph. To suit the recursive nature of graphs, the algorithm itself is recursive. Given the *END* element of a graph and its level, algorithm draws its left sub-graph starting one level down, draws its right sub-graph one level down, and then draws the parent's at its level. Algorithm by the first draws the left sub-graph, then the *END* element, and lastly the right sub-graph, because after drawing the left sub-graph it has enough information to draw the *END* element (in essence, it knows the number of nodes in the left sub-graph).

2.3.2. Process of saving and loading projects

The application uses a special structure of file with postfix *.ygg* for saving projects. This structure consists of XML tags. In these tags all important information is coded for description of all nodes and connection lines used in a project. The application stores these elements of every node inside graphics scene: *x-coordinate, y-coordinate, type, address, value, text and colour.* For edges application stores these elements: *address of node from which line starts, address of destination node, type of connection pin from which line starts* and *type of destination node's connection pin.* The order of these elements is important for successful loading of project from the disk. The final design and implementation of the CASE tool for programming of dataflow computers is shown on the Fig. 2, along with the aligned graph.

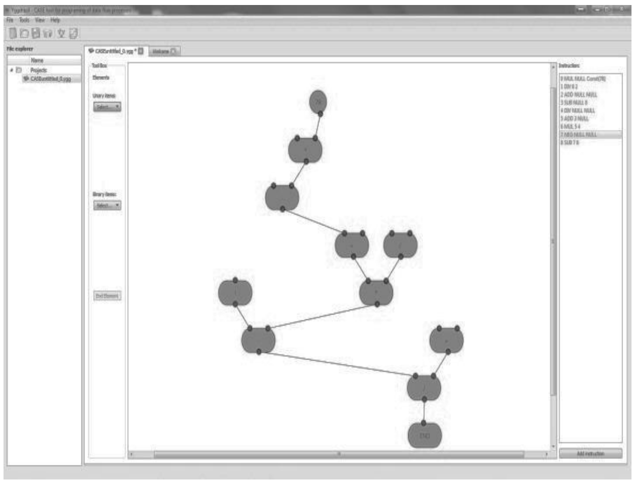


Fig. 2 Final design of the CASE tool for programming of dataflow computers

3. CONCLUSION

CASE tool for programming of data flow computers brings to developers of prototype of dataflow computer better tool for work with that prototype. Whole creation of programs for target architecture is easier, because drawing "code" is easier than writing 128 binary bits vectors that are accepted by the data flow computer. However CASE tool has still some limitation. Code – vectors are generated in string form and developer has to move this code to the target machine. This moving process takes a time. Avoid this process is the biggest possible improvement of this process. Making closer connection between CASE tool and dataflow computer and make moving process automatic can resolve that problem.

ACKNOWLEDGMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and KEGA 008TUKE-4/2013 Microlearning environment for education of information security specialists. The projects are being solved at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice.

REFERENCIES

[1] VOKOROKOS, L. - MADOŠ, B. - ÁDÁM, N. - BALÁŽ, A.: *Priority of Instructions Execution and DFG Mapping Techniques of Computer Architecture with Data Driven Computation Model*, SISY 2011: 9th IEEE International Symposium on Inteligent Systems and Informatics: 8. - 10.9.2011: Subotica, Serbia P. 483-488 Budapest: Obuda University, 2011.

[2] VOKOROKOS, L. - MADOŠ, B. - ÁDÁM, N. - BALÁŽ, A.: *Innovative Operating Memory Architecture for Computers using the Data Driven Computation Model*, In: Acta Polytechnica Hungarica: Special Issue on Celebration of 60th Anniversary of the Foundation of Technical University of Košice. Vol. 10, no. 5 (2013), p. 63-79. - ISSN 1785-8860.

[4] MADOŠ, B.: *Architecture of Multi-Core System-on-the-Chip with Data Flow Computation Control*, In: International Journal of Computer and Information Technology (IJCIT). Vol. 3, no. 5 (2014), p. 958-965. - ISSN 2279-0764.

[5] ÁDÁM, N.: *Mikroprogram pre riadenie procesu spájania operandov v architektúre DF KPI*, In: Acta Informatica Pragensia. Vol. 2, no. 2 (2013), p. 77-96. - ISSN 1805-4951.

[6] CHOVANCOVÁ, E. - DUDLÁKOVÁ, Z. - FORTOTIRA, O. - RADUŠOVSKÝ, J.: *Multicore processor focused on voice biometrics, In: ICETA 2014*: 12th IEEE International Conference on Emerging eLearning Technologies and Applications: proceedings: December 4-5, 2014, Starý Smokovec. - Danvers: IEEE, 2014 S. 73-77. - ISBN 978-1-4799-7738-3

Received February 9, 2015, accepted March 2, 2015

BIOGRAPHIES

Branislav Madoš (Ing., PhD.) was born on 20th May 1976 in Trebišov, Slovakia. In 2006 he graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. He defended his PhD. in the field of Computers and computer systems in 2009; his thesis title was "Specialized architecture of data flow computer". Since 2010 he is working as anAssistant Professor at the Department of Computers and Informatics. His scientific research is focused on the parallel computer architectures and architectures of computers with data driven computational model.

Ján Hurtuk (Ing.) was born on 4th October 1988 in Kežmarok. In 2013 he graduated (MSc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. Since 2014 he is studying as a PhD. student at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. His scientific research is mainly focused on the computer security.

Marek Čajkovský (Ing.) was born on 17th December 1986 in Veľký Krtíš, Slovakia. In 2011 he graduated (MSc.) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice and received the engineering degree. Since 2011 he is PhD. student at Faculty of Electrical Engineering and Informatics at Technical University of Košice. His research is focused on computer security, the title of his doctoral thesis is: Identifying Security Threats by System Services Calling. His professional interests include programming, computer networking, computer security and UNIX based operating systems.

Erik Kudra (Bc.) was born on 23th December 1988 in Michalovce, Slovakia. In 2011 he graduated with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice and received the bachelor degree. Since 2012 he is Ingstudentat the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His professional interests include development and testing software for mobile and desktop platforms.