Single Input Operators of the DF KPI System

Norbert Ádám

Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
norbert.adam@tuke.sk

Abstract: The DF-KPI system is a multiprocessor system based on the dynamic data flow computing model, where each dyiadic instruction requires dynamic matching of its operands. This process is associated with the processing units of the DF-KPI system. The processing units are designed as a dynamic multi-function pipelined unit with LOAD, FETCH, OPERATE, MATCHING and COPY segments. In this article, an architecture design at a logical level of the dynamic multi-function pipelined unit, which handles processing of operand matching for single input data flow operators is suggested.

Keywords: data flow, operand matching, data flow graph, single input data flow operators

1 Introduction

One of the solutions to the problem of reaching higher computer system performance is the concept of high performance parallel computer systems' architecture. Computer systems based on the von Neumann computer principle try to solve these requirements by increasing the performance of particular computer segments. However, possibilities of this kind of speed-up are conditioned by the technological potential [5], [6], [7], [12]. Better results in increasing computer system performance can be achieved by computers based on the data flow computing model. In contrast to control flow architectures, such as the von Neumann model, data flow architectures use the availability of data to fetch instructions, rather than the availability of instructions to fetch data [2]. The data flow computation model enables us to use the program's natural parallelism, which, in turn, shortens the time required to perform a calculation. The advantage of data flow computing (or computers) is that it allows the detection of parallelism at the level of machine instructions, just like the application of data-level parallelism. The disadvantage is the complicated operand-matching control mechanism, increased communication between processing units and the relatively large program storage requirements. But it is possible to bypass all disadvantages of this architecture and get huge computing capacity systems by means of

appropriate architecture design, for example: reconfigurable chip area for processing units, interconnection networks, as are the hypercube, pyramid etc., and by elimination of redundant calculations at the level of program to data flow graph translation, making use of an efficient computation management micro-program with a well designed operand matching algorithm.

2 Data Flow Computers

In the field of the development of high performance new generation computers, architectures based on the data flow computation model are in a parallel computer class of their own. Computational process control in the data flow computation model is implemented by the stream of operands (data) prepared to execute program instructions.

Even though theories exist about data flow models [2], [3], [8], [11], many architectures have been proposed [2], [10], [11]. These can be classified as static and dynamic (or tagged-token) architectures according to the related approach to the model.

The characteristic property of data flow computers is that the data flow program instructions are waiting passively for the arrival of a certain combination of arguments, which are made available as a data control stream in a data-driven sense [3]. The waiting interval, where a program instruction waits for the arrival of operands, represents its selection phase, during which the allocation of computing resources occurs. The fundamental idea behind the data flow computational model is the mapping of tasks (Fig. 1) to the computing elements, which can increase the rate of parallelism. In general, it is necessary to decompose the computational process into smaller communicating processes represented by the data flow program [4].

The data flow program, which uses the data flow computational model, is represented by its machine representation, called a data flow graph. Data flow graphs are directed graphs that show the data dependencies between the respective instructions (operators). Their nodes represent instructions and the arcs connecting these nodes represent the operands of these instructions. An instruction can be executed if all the tokens on the incoming arcs are available: that is, if all the operands of the instruction are available.

The implementation of the data flow computer architecture depends on the form of execution of the data flow program instructions, which is implemented as a process of receiving, processing and transmission of activation symbols (data tokens).

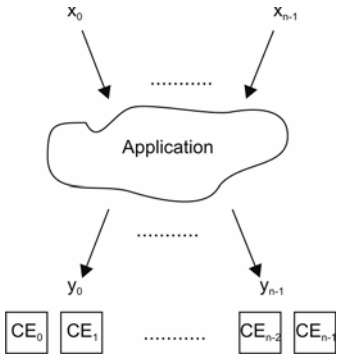

Figure 1
Tasks mapping to the computing elements of the computer system

The following are the types of pure data flow architectures, depending on the concept of processing of data tokens or on the scope of architectural support for its implementation:

- Static models (Figure 2)
- Dynamic models (Figure 3)

The static data flow model was proposed by Dennis and his research group at MIT [3]. In the static approach a node (instruction) can be executed only when all of the tokens are available on its input arcs and no tokens exist on any of its output arcs [8]. This model is capable of using data parallelism and pipelining techniques; therefore this model has found use in applications with regular computing structures. The general organization of the static data flow machine is depicted in Fig. 2.

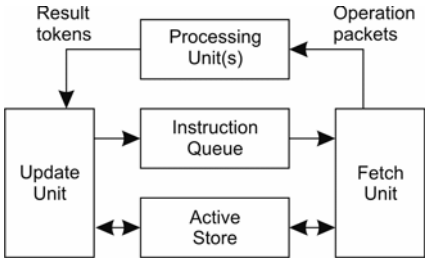

Figure 2
The general organization of the static data flow model

The *Active Store* contains the activation frames of instructions, representing the nodes in data flow graph. Each activation frame contains the operation code, slots for the operands and the destination address. An operand slot contains also a presence (check) bit to determine the availability of the operand.

The *Update Unit* performs the update of the activation tokens and checks whether the instruction is executable. If the feasibility condition is fulfilled, the unit sends the instruction through the instruction front to the instruction fetch unit.

The *Fetch Unit* fetches and sends a complete operation packet containing the corresponding operation code, data, and destination list to the Processing Unit and also clears the presence bits.

The *Processing Unit* performs the operation, forms the result packets and sends them to the Update Unit.

The dynamic data flow model was proposed by Arvind at MIT [1] and by Gurd and Watson at the University of Manchester [4]. The operator represented by a node is executable in the dynamic data flow model (Fig. 3) if all the input edges contain tokens, the symbols of which are identical. Each edge may contain more than one labeled token in this model. When executing a node, the tokens belonging together are removed from the input edges and a token with the corresponding symbol of the output edge is generated. The dynamic dataflow model uses both loop parallelism and recursive parallelism, which appear dynamically during program run-time. Such an architecture must support the process of operand matching (merging).

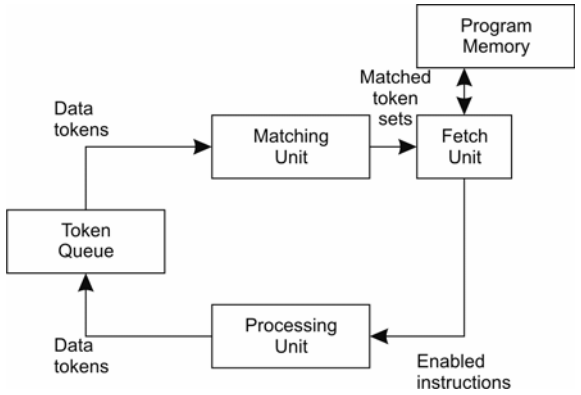

Figure 3
The general organization of the dynamic data flow model

The *Matching Unit* represents a memory which performs matching of tokens at the operator's input. The tokens are merged only if the token is destined for a double input operator, which initiates the operand matching process. If a match exists, the corresponding token is extracted from the Token Queue and the matched token set is passed on to the Fetch Unit. If no match is found, the token is stored in the Matching Unit and waits for a a partner.

The *Fetch Unit* performs a selection of the prepared instructions from the Program Memory and generates executable packages for the Processing Unit.

The *Program memory* is the instruction memory of the data flow program.

The *Processing Unit* processes operations defined in the executable packages and produces result tokens, then sends them to the Matching Unit via the Token Queue.

The *Token Queue* transports the input of tokens from the Processing Unit to the Matching Unit.

3 The DF KPI System

The DF-KPI system [6], being developed at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, has been designed as a dynamic system with direct operand matching. The combination of the local control flow model (von Neumann's principle) with the global data flow model allows us to effectively organize the parallel implementation of functional program. The architecture model of the DF-KPI computer is a part of a data flow complex system, which includes support components for data flow computing environment for the implementation of the defined application targets.

The structural organization (Fig. 4) of the DF-KPI computer architecture model consists of the following components:

Coordinating Processors (CP) are intended to manage, coordinate and process instructions of the data flow program, based on the presence of their operands, which are enabled at the CP.DI input port of the coordinating processor - either from its CP.DO output port or from the CP.DO output ports of other CPs through an interconnection network, or from a Data Queue Unit and from the Frame Store. The structure of the CP is a dynamic pipelined multiple-function system, composed of LOAD, FETCH, OPERATE, MATCHING and COPY segments.

The *Data Queue Unit* (DQU) is a unit designed to store the activation symbols (data tokens), which represent operands waiting for matching during program execution.

The *Instruction Store* (IS) is a memory of instructions of the data flow program, in the form of a proper data flow graph.

The *Frame Store* (FS) is a memory of matching (pairing) vectors, by means of which the CP detects the presence of operands to perform the operation defined by the operator (node) in the data flow graph. The short description of the item format of MV matching vector in the FS is <FS>::= <AF><V>, where AF is a flag of the operand's presence (Affiliation Flag) and V is the value of the given operand.

Supporting components of the data flow system are needed to create a realistic computing environment. In the given architecture they are formed by the following:

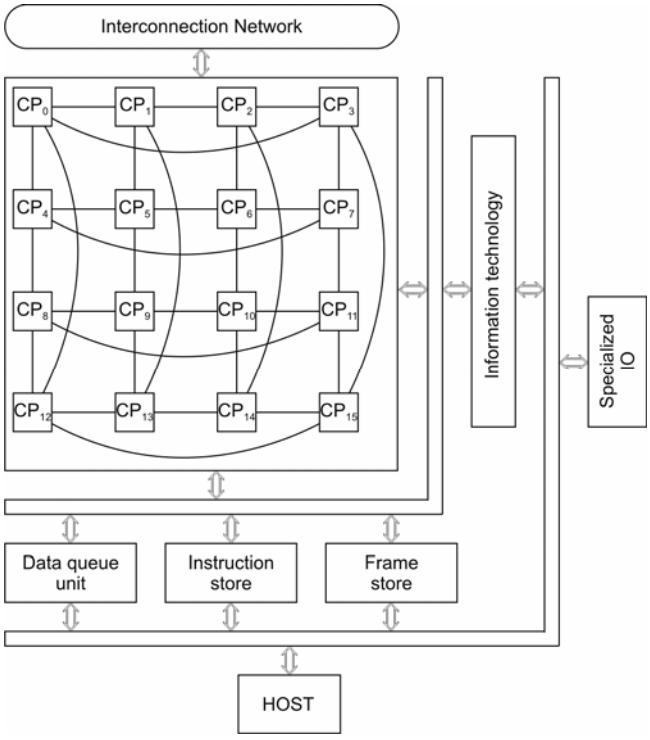

Figure 4
The DF-KPI system

The *Main computer* (HOST) provides standard functions of the computer system during data flow computing process.

The *Information Technology* unit is a unit used to create dedicated application environments (virtual reality, diagnostics, e-learning).

The *I / O processors* for fast direct inputs/outputs into the data flow module (standard I/Os are implemented by the main computer).

3.1 Instruction Format

Each instruction is represented by an operator of the data flow program. Data flow operators are generally classified as single input, double input and N ($N \geq 3$) input operators. In the case of single input operators, the input is formed by one operand, which meets the condition of the data availability firing rule, meaning

operand matching is not necessary. In the case of N input operators, the operator will be executed if all input operands are present, which are then merged in matching segment, based on the matching vector MV. The instruction set of the DF-KPI architecture consists of instructions represented by the single input operators (ACCEPT, IF, KILL, OUT, RET, SEL, UN_OP), double input (BIN_OP, CASE, DEF, GATE, LOAD, SEND, TUP) and N ($N \geq 3$) input (APPLY, CONSTR). The ACCEPT operator is an entry operator of the data flow program and subprograms. The IF operator represents a two-way branch and the CASE operator represents a multi-way switch. The KILL operator is used for consumption of input operand without any response. The OUT operator is the last operator and RET is a return operator of the data flow program or subprograms. The SEL operator selects data from the data structure defined by the CONSTR operator. The UN_OP single input operator and the BIN_OP double input operator represent unary and binary arithmetic operations. The DEF operator defines the program constant. While LOAD creates copies of the data and address part of the input operand, the TUP operator creates copies only of the data section of a data token. Running the program or subprogram is launched by the APPLY operator. A more detailed description of these operators is in [6].

The format of the data flow instructions is as follows:

$$<\text{DFI}>::= <\text{OC}> <\text{LI}> <\{\text{DST}, [\text{IX}]\}^{n}>$$

where OC is the operation code; LI is a literal (e.g. number of copies of the result); DST represents the target address for operation result; IX is a matching index for the operations.

The data flow program instruction represented by a data token is stored in the Instruction Store at the address defined by DST field. The data token has the following format:

$$<\text{DT}>:: = <\text{P}> <\text{T, V}> <\text{MVB}> <\{\text{DST}, [\text{IX}]\}>$$

where P is the priority of the data token; T represents the data type of operand with a value V; MVB defines a base address of matching vector in the Frame Store and DST specifies a destination address of the resulting DT data token. The structure of the DST field is the following:

$$<\text{DST}>:: = <\text{MF}> <\text{IP}> <\text{ADR}>$$

where MF is a matching function, with a defined set of labels {M, B}, M stands for matching (of two DTs), B stands for bypass (without DT matching); IP defines an input port {L(eft), R(ight)}; ADR is the address of the operator or function.

If the operands enter the two-input or multi-input operators, operand matching occurs. The DF-KPI architecture uses the direct operand matching control mechanism. It is based on the allocation of a Matching Vector in the Frame Store according to the activation code (procedure, call). Allocated Matching Vectors are

represented as a matching record in the Frame Store. The format of the Matching Vector in the Frame Store is as follows:

$$<SS\ [B_{ACT} + H + IX + 1]> :: =$$

$$<RC, MVS> <B_{OLD}> < DST_{RET}> <D\{[B_{NEW}]\ \{D\}\} >$$

where B_{ACT} is a pointer to the current top record; H is the size of a header of record; MVS defines the size of a matching vector; RC is the reference counter; B_{OLD} is a pointer to the previous token; DST_{RET} specifies the return address; B_{NEW} defines the base address for new matching record and D represents an operand value.

The RC field is set according to the size of the matching vector at compile-time. After the function associated with the operator has fired, the value of RC is decremented. If RC = 0, the Matching Vector in the frame store is released.

3.2 Operand Matching

As mentioned above, data tokens convey information about the status of the calculation. Their location at the input edges of a node (operators) in the data flow graph means the presence of the operand for a defined and enforceable instruction. The presence of a data token at the output edge of a node reflects the presence of the outcome of operations performed by the operator defined instructions.

The processing of a data flow graph is subject to the rules of firing an instruction (an instruction is executable if all of its operands are available) and activation (an instruction is activated when it is fired and the resources required for activation are available).

One of the most important steps based on the dynamic data flow model is direct operand matching [4], [8]. The concept of direct operator matching represents the elimination of the costly process (in terms of computing time) related to associative searching of the operands. In this scheme, a matching vector is dynamically allocated in the Frame Store memory for each token generated during the execution of the data flow graph. The current location of a matching vector in the Frame Store is determined at compile time, while the Frame Store location is determined after the program starts. Each calculation can be described using an instruction address (ADR) and the pointer to the matching vector MVB in the Frame Store. The <MVB, ADR> value pair is part of the token. A typical action is the searching for the operands pair in the Frame Store. The matching function provides searching for the tokens marked identically. After the operand has arrived to the Coordinating Processor, the matching function detects if a commonly entered operand is present in the Frame Store. Detection is performed according to matching IX index. If the operand is not there yet, it is stored in the Frame Store, in the Matching Vector specified by base address of the MVB operand, into the item specified by index IX.

The operand matching process control at the operator input is influenced by the process of matching, instruction execution and generation of a result at its output. Using a compiler producing DFG output with forward searching that allows for the detecting and eliminating of redundant computations and change order of token processing, process control can be defined as the transition of activation signs along the edges of the data flow graph (Fig. 5), between the "producer" (P) operator and the "consumer" (C) operator.

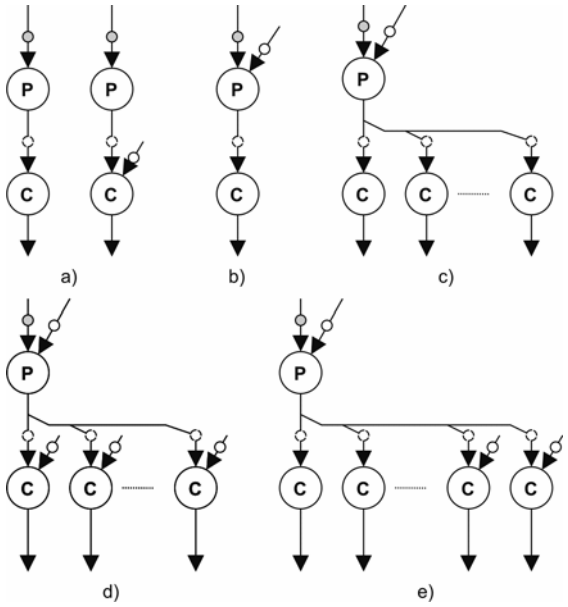

Figure 5
The operand matching (a – P-single input, b – P-double input, c – P-double input/C-single input, d – P-double input/C-double input, e – P-double input/C-u-single,v-double input)

The P and C operators can be single input, double input and management operators respectively. Varying configurations of P and C operators are shown in Fig. 5.

By processing the data flow graph the operators can be configured and connected as follows:

P-single input, C-single input (PS/CS). This configuration doesn't require operand matching. The token produced by the P producer is absorbed by the C consumers (Fig. 5a).

P-single input, C-double input (PS/CD). It requires operand matching on operator C (Fig. 5a).

P-double input, C-single input (PD /CS). Operand matching occurres only on operator P (Fig. 5b).

P-double input u×C-single input (PD /uCS). After processing the instruction defined by double input operator P, the result is distributed among u single input C operators. Operand matching occurs only with the operator P (Fig. 5c).

P-double input, v×C-double input (PD/vCD). The result is distributed between the v double input C operators. Operand matching is activated with operator P and operator C, too (Fig. 5d).

P-double input, u×C-single input, v×C-double input (PD/uCSvCD). The result of the activated double input operator P is sent to u single input and v double input C operators. Operand matching occurs with operator P as well as with double input C operators (Fig. 5e).

In this article we describe the operand matching control for configuration shown in Figure 5a.

3.3 Implementation of PS/CS and PS/CD Operators

The coordinating processor represents a dynamic pipeline system, which allows us to switch between the Load (L), Matching (M), Copy (C), Fetch (F) and Operate (O) states in a different order. The transitions between the states using a micro-management program is shown by means of a state diagram (Fig. 6). The micro-program manages the operand matching process.

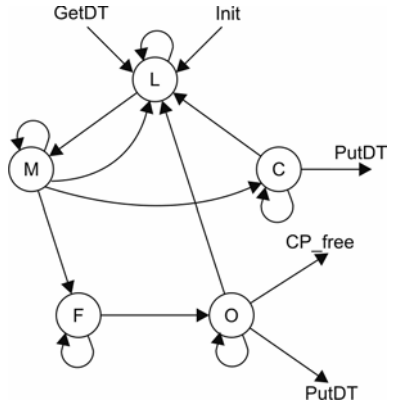

Figure 6
The state diagram of the operand matching control process

The control signals shown in Fig. 6 have the following meanings: CP_free – indicates the occupancy or availability of the coordinating processor; GetDT – read token from DQU; PutDT – write token to DQU; Init – initialization of pipelined system. The proposed architecture at a logical level of operand matching control is show in Fig. 8. FIFO registers with the following specifications have

been inserted to increase the throughput coefficient between the various stages of coordinating processor (Fig. 8):

- Between the stages L and F → register LFR
- Between the stages F and O → register FOR

Single input operator processing is done by means of micro-operations defined for each state (segment) of the dynamic pipelined multi-function unit of the current coordinating processor (Fig. 8).

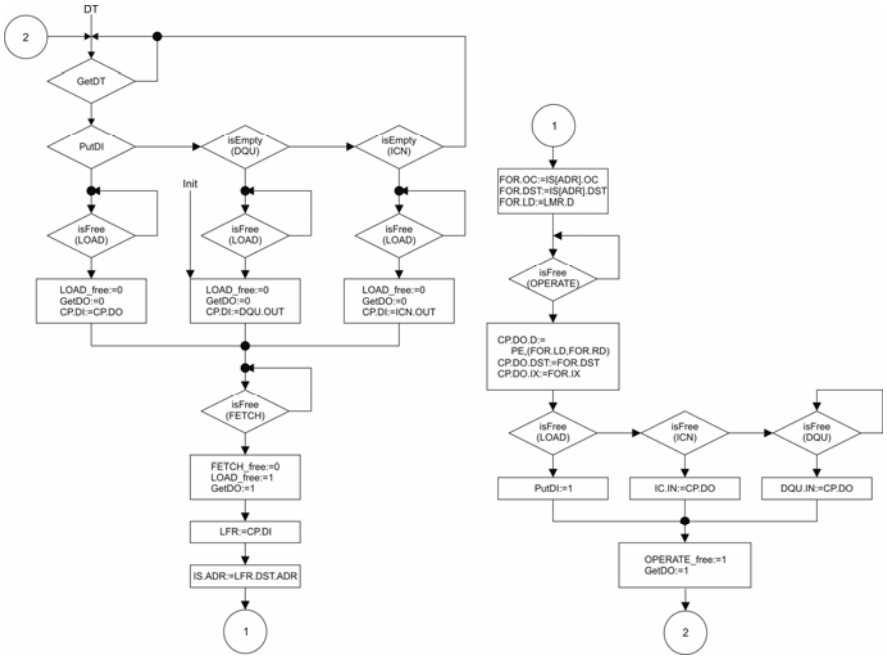

Figure 7
Micro-program for single input operators matching

The function isFree (X) tests the busy state of segment X. Micro-operations, which can be executed in parallel, are placed in a single command block (processing steps) in the program flowchart. Initialization of the coordinating processor is done by sign Init = 1. The boot command of the data flow program loads the data token from the DQU to the LOAD segment, sets the busy flag for the LOAD segment to 1 (i.e. the LOAD segment is occupied) and blocks the processing of the following tokens (GetDT = 0). If the next segment, the Fetch segment, is free, the token is loaded into the Load/Fetch register. After that, the micro program releases the LOAD segment and activates the loading of other tokens into the coordinating processor.

The control mechanism determines the DF address operator based on the LFR.DST.ADR address. The operator will be loaded from the instruction store into the Fetch/Operate register. If the Operate segment is not busy (isFree (Operate) = true), the operator is fetched from the Fetch/Operate register and processed. In the next step, if the CP is not busy, the result of the operator consumption and processing is available for processing in the same CP. Otherwise, the result is to another CP through the interconnection network. If all CPs are busy, the token is stored in the DQU.

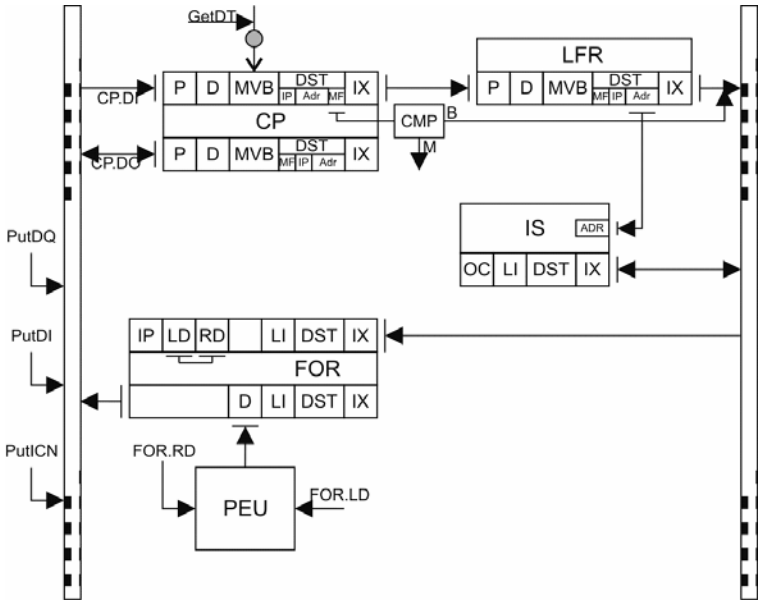

Figure 8
Pipeline system for processing of single input operators

Conclusions

This article presents the architecture design of the dynamic pipelined multiple-function system at a logical level, which handles processing of operand matching for single input data flow operators. Verification of proposal correctness can be realized by simulation or technical realization using statistical formulas and tools. At the Department of Computer and Informatics, a demand for realization of simulation tool for DF-KPI (being developed by this department) system has been recognized, within the scope of APVV-0073-07 and VEGA grant project No. 1/0646/09. Simulations are still to be performed at the level of the interconnection network (matching memory and tables), as well as at the level of the entire system. For system-level simulation compiler from a higher-level programming language to DFG is to be used (functional languages are preferred). The currently developed DF-KPI system with its principle of data processing and its parameters is intended

for solving tasks requiring brute force. The development of the DF-KPI system is focused on the fields of virtual reality [9] and computer security [13].

Acknowledgment

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0073-07 and VEGA grant project No. 1/0646/09: "Tasks solution for large graphical data processing in the environment of parallel, distributed and network computer systems" and by Agency of the Ministry of Education of the Slovak Republic for the Structural Funds of the EU under the project Centre of Information and Communication Technologies for Knowledge Systems (project number: 26220120020).

References

[1] Arvind, D. E. Culler: Dataflow Architectures. Annual Review in Computer Science, 1986, Vol. 1, pp. 225-253

[2] J. Carlström, T. Bodén: Synchronous Dataflow Architecture for Network Processors, Micro IEEE, Volume 24, Issue 5, Sept.-Oct. 2004, pp. 10-18, ISSN 0272-1732

[3] J. B. Dennis: Data-Flow Supercomputers. Computer, Nov. 1980, pp. 48-56

[4] J. R. Gurd, C. C. Kirkham, I. Watson: The Manchester Prototype Data-Flow Computer. Commun. ACM, Vol. 28, pp. 34-52, Jan. 1985

[5] Gy. Györök, M. Makó, J. Lakner: Combinatorics at Electronic Circuit Realization in FPAA, Acta Polytechnica Hungarica, Vol. 6, No. 1, pp. 151-160, 2009

[6] M. Jelšina: Design of Data Flow KPI Computer System (in Slovak). elfa s.r.o., Košice, 2004, ISBN 80-89066-86-0

[7] M. Jelšina: Computer system architectures (in Slovak), elfa s.r.o., Košice, 2002. ISBN 80-89066-40-2

[8] B. Lee, A. R. Hurson: Issues in Dataflow Computing. Advances in Computers, Vol. 37, Academic Press, Inc., San Diego, CA, 1993, pp. 285-333

[9] B. Sobota, J. Perháč, M. Straka, Cs. Szabó: The Applications of Parallel, Distributed and Network Computer Systems to Solve Computational Processes in an Area of Large Graphical Data Volumes Processing (in Slovak); elfa s.r.o., Košice, 2009, ps. 180, ISBN 978-80-8086-103-2

[10] S. Swanson, K. Michelson, A. Schwerin, M. Oskin: WaveScalar, Proc. of the 36th International Symposium on Microarchitecture (MICRO-36 2003) 2003, pp. 291-302, ISBN 0-7695-2043-X

[11] B. Verdoscia, R. Vacarro: ALFA: A Static Data Flow Architecture, In Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation, McLean, VA, USA, 1992, pp. 318-325

[12] L. Vokorokos: Data Flow Computer Principles (in Slovak), Copycenter, spol. s.r.o., Košice, 2002. ISBN 80-7099-824-5

[13] L. Vokorokos, N. Ádám, A. Baláž, J. Perháč: High-Performance Intrusion Detection System for Security Threats Identification in Computer Networks, Computer Science and Technology Research Survey, Košice, elfa, s.r.o., Letná 9, 042 00, Košice, Slovak Republic, 2009, 4, 4, pp. 54-61, ISBN 978-80-8086-131-5