ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 1, No. 2, 2016

PARALLEL ORDERED-ACCESS MACHINE COMPUTATIONAL MODEL AND ARCHITECTURE

Anatoliy Melnyk

Lviv Polytechnic National University, 12, Bandera str., Lviv, 79013, Ukraine
Authors e-mail: aomelnyk@polynet.lviv.ua

Submitted on 28.12.2016

Abstract: **The article presents the new computational model which we name the parallel ordered-access machine because of its base – the parallel ordered-access memory. It also describes the computer architecture which implements proposed computational model and owing to this does not have such a limitation as the memory wall and provides parallel conflict-free memory access. The efficiency of the proposed ordered-access machine computational model is evaluated and an example of its implementation is presented.**

Index Terms: **Computational model, Computer architecture, Parallel ordered-access memory, Parallel ordered-access machine.**

I. INTRODUCTION

In the process of computer program development, which is written in the internal computer language to organize a sequence of computations given by the algorithm, an identification of the program components is used. An identification of some type provides finding the program components in the memory in accordance with access type. In modern computers the program components are identified by the addresses of the memory locations which hold them. In this case under the program components we mean the instructions, the data items, and the addresses. Each instruction of the program specifies the type of the operation to be executed, the addresses of the initial and intermediate data items to be processed similar to final data items, and (or) the address of the next instruction location in the main memory but only if its address is not calculated by the program counter incrementing. The appropriate computational model, which is based on the address memory access and on the identification of the program components by the addresses, supposes to execute each instruction, its loading together with the data items from the main memory locations pointed by the addresses, to execute the operations of data processing and to store the final data items in the main memory locations pointed by the addresses [1, 2, 3, 4, 5]. That is, the sequence of calculations, given by the algorithm, is provided in this computer by loading the program components according to the addresses of their locations in the main memory.

The interest in creation of a new computational model is caused by the need for increasing the computer's performance. The issue is especially important nowadays when we talk about a decrease in the computer performance growth by means of improving the integrated technology.

The main architectural factor that limits the performance of the modern computer, whose program components are identified by the addresses, is just the implemented computational model. Usage of this computational model is caused by the fact that the random-access memory (RAM), which the computer main memory is built on, is addressable and provides random access in each cycle, but only to one location [1], [4]. Thus, because the same memory unit is used for both data items and instructions, it allows only the sequential request by definition; the serial nature of computations is already laid in the principles of the modern computer architecture.

The problem of the RAM bottleneck is partly solved using the memory divided into the multiple memory banks. This statement is confirmed by the fact that the shared memory computers belong to the most powerful and fast ones [6, 7, 8, 9]. Commercial shared memory computers provide parallel data processing, but their programming needs more time and it is often quite difficult to provide high performance [10, 11, 12, 13, 14].

In addition, only one program counter can be used in the computer that implements this computational model. That is another bottleneck that constrains the ability to increase the computer's performance. Parallel loading of several instructions from the main memory, as it is in the superscalar architecture, and packaging of several consecutive instructions into one long instruction word, as it is in the VLIW architecture, allows the increasing of performance, but significantly complicates the computer.

It should be noticed that other developed and tested computational models that use an identification of the data items according to their content or to some events (associative computers, data flow machines, reduction computer architecture, etc. [15, 16, 17, 18, 19]), have not become an alternative to the traditional one that uses an identification of the program components by addresses.

The basic idea of the suggested computational model is to use the identification of the program components by their positions in the arrays. This computational model supposes indexing of each instruction and each initial, intermediate and final data items, holding them in the parallel conflict-free ordered access memory, ordering them inside this memory in the arrays according to their indices and further parallel processing of these arrays that provides significant decreasing of the data processing time and, accordingly, leads to the computer performance increasing.

The paper is organized as follows. Section 2 describes in detail the most used computational models in modern computers: stack-, content- and random-access machines. Section 3 outlines the conventional memory types and challenging problems of their using in high-performance computers. Section 4 proposes the method of the program components identification by their indices in accordance with matrices. Section 5 represents the basics; organization and functioning of the parallel ordered access memory which provides the parallel conflict-free memory access and ordering data according to their indices. Section 6 is dedicated to the new computational model description and comprises the problem description model and the execution model. Section 7 describes the new parallel ordered-access machine architecture and functioning. Section 8 then combines the results from the individual stages of the flow. Finally, Section 9 presents our conclusions.

II. KNOWN COMPUTATIONAL MODELS

Computational model is a foundation of computer architecture. The concept of computational model comprises the set of the following three abstractions: the basic items of computations, the problem description model, and the execution model [20]. The basic items of computation are the items the computation refers to and the kind of computations (operations) that can be performed on them. To determine the basic items of computations the method of the program components identification is used. The problem description model refers to both the style, which specifies how problems in a particular computational model are described, and the method, procedural or declarative, of the problem description. The execution model interprets how to perform the computation, the execution semantics and control of the execution sequence. The computer architecture may be looked upon as a tool to implement a computational model.

A few computational models are known for now. The most used in the modern computers are the following computational models: stack-, content- and random-access machines, which are examined below.

A. Stack-access machine computational model

In the stack-access machine computer architecture the computational model based on the access to the data items according to the stack pointer value is being used [5], [21, 22, 23]. In this architecture the final data items are received by processing of the initial and intermediate data items according to the algorithm that is described in the program written in advance. Each instruction of the program determines the computer work during the time needed for executing a single operation. According to this computational model the data items are placed in the memory locations in such a way that their addresses can be pointed by the counter. This eliminates the need to specify the addresses of the data items in the instruction and reduces the time of its execution.

The disadvantage of this computational model is the sequential data processing and low performance of the computer that implements this computational model.

B. Content-access machine computational model

In the content-access machine computational model every data item that is administered during the algorithm execution has the key. Data item, its part, or pattern can be used as the key. Data item is stored in the memory location together with its key and is submitted to processing when its key coincides with the given one. It allows the stored data item to be located and retrieved though its physical address is unknown, and without recourse to a sequential search. The values of the keys are formed on the basis of compliance with the order of the algorithm operations execution.

This computational model is implemented in the associate computers [24, 25, 26]. It showed high efficiency to solve problems of search and sort, but has not found widespread use.

C. Random-access machine computational model

The random-access machine computational model is based on the address access to the program components. It is the basis for the modern computers. This computational model was first introduced in John von Neumann papers, and in many subsequent studies of the computer architecture and programming [27, 28, 29, 30, 31, 32, 33, 34, 35]. According to this computational model the final data items are received by processing of the initial and intermediate data items in accordance with the algorithm that is described in the program written in advance. Each instruction of the program determines the computer work during the time of a single operation execution. The addresses are used to indicate the instructions and the data items (initial, intermediate and final, resulting from operations) that are stored in the main memory. They point the locations allocated to the instructions and to the data items in the main memory. The instruction indicates the type of operation being executed, for which the operation code is used, and the addresses of the main memory locations that hold the data items which the operation is to be executed on and where the final data items will be stored.

Computer that implements this computational model and has in its memory the program and the initial data works as follows: start; writing into the program counter an address of the first instruction of the program and executing of so-called machine cycles, the number of which is equal to the number of instructions that will be executed. A machine cycle includes the following steps:

fetching the instruction from the memory using an address from the program counter, decoding the instruction and checking whether this is a "stop" instruction and stopping or continuing the computation, executing an instruction, that is fetching the data items from the memory specified by addresses in the address field of the instruction, operation execution and writing the result data items into the memory at the addresses specified in the instruction, calculating the address of the next instruction and its entering into the program counter, returning to the start of the machine cycle [2].

In this way all the instructions of the program are executed. At the end the final data items will be placed in the specified memory locations.

The mentioned above computational model is characterized by wider functionality in comparison with the stack- and content-access computational models because of the identification of the program components by addresses is more flexible, but on the other hand it requires a large number of steps to implement each instruction in particular and the program in general to compute the addresses of the instructions and the data items and to transfer them between the processor registers, which results in significant decrease of the computer performance.

III. CONVENTIONAL MEMORY TYPES AND CHALLENGING PROBLEMS OF THEIR USING IN HIGH-PERFORMANCE COMPUTERS

The access method, implemented in the computer memory of the above mentioned computer architecture types, is determined by the method of the program components identification used in them. Depending on it the memory can be classified as follows [1, 2, 3, 4]:

- Sequential-Access Memory (SAM), where the sequential memory access method is implemented. This memory allows writing in or reading out data items sequentially in each cycle one after another.
- Content-Access Memory (CAM), where content memory access method is implemented. In this memory data item is searched by its key.
- Random-Access Memory (RAM), where address memory access method is implemented. This memory allows writing in or reading out the data item in each cycle by an arbitrary address.

Each memory type brings the problems when it is considered for using in high-performance computers.

The main disadvantage of the SAM is long time for the particular data item searching. In the worst case it may require reading of all previously written data items. In addition, this memory has low functionality as it does not provide more than sequential memory access. And for the last, this method is designed to work with vector data and does not provide parallel memory access.

The challenging problem of the CAM is the need to provide access to each location from its ports and the need to compare simultaneously the keys in all its locations with the given one that requires a large equipment volume and slows down the access time. In addition, this method allows parallel memory access only to the data items with the same key that limits its application.

The challenging problem of the RAM is its inability for parallel access from many ports. The reason lies in a conflict in the case of addresses convergence on multiple memory ports in the write mode. This is exactly the reason of inability to implement the parallel RAM-based processors. Two other problems: the need to store the addresses of the memory locations where the data items were written in so that if necessary they can be found and read out, and the need to put the addresses at the address input of the memory in both write and read modes.

IV. METHOD OF THE PROGRAM COMPONENTS IDENTIFICATION BY INDICES

For known methods of program components identification the new method is being proposed, where the program components are identified by their indices in accordance with matrices. According to this method the data items are identified by indices during the program development. An identifying index is assigned to each data item. These indices coincide with the data items position in the matrices of the data to be processed. Herewith the intermediate and final data are indexed in advance during the program development, namely their indices are prearranged, because of the fact that these data items will be obtained during the information processing. Besides, the indices that are prearranged for the intermediate and final data items also are identified during the program development. An identifying index is assigned to each data item identifier. The index of the data item identifier coincides with the data item position in the matrix of the identifiers of the intermediate and final data.

Like the data items the instructions of the program are indexed during its development. An identifying index is assigned to each instruction. These indices coincide with the instructions position in the matrices of the instructions to be executed.

Indexing of the data items, instructions and identifiers is performed by the rule that allows process them in the sequence given by the algorithm.

It should be noted that the initial, intermediate and final data items, the instructions, and the identifiers of the intermediate and final data items and instructions can be grouped under the same identifiers.

V. PARALLEL ORDERED-ACCESS MEMORY

Using the method of the program components identification by indices the new memory access method could be developed. The goal of this method developing is to expand the functions of the memory by providing the parallel conflict-free memory access and ordering data according to their indices during their writing, storing or (and) reading. The memory that implements this method consists of locations where the data matrix may be written in, stored, and read out. This method supposes to implement the following steps:

- an index is assigned to each data item of the input data matrix whose numerical value determines the position of this data item in the output data matrix;
- the indices are processed, for example, they are sorted in the ascending or descending order of their numerical values;
- the data items of the input data matrix are ordered according to the numerical values of their indices and the output data matrix is formed.

As it follows from the above described, the ordered memory access method, unlike the sequential memory access method, allows the extending of memory functionality, as it provides not only the sequential but also any other ordered access.

Unlike the address memory access method of implementation of the ordered memory allows not to bind the data item with the specific memory location and eliminates the need to use the addresses during data items writing and reading. Accordingly, firstly, there is no need to store the addresses of the memory locations where the data items are placed, and, secondly, there is no need to submit the addresses on the address inputs of the memory during both data write and read modes because, according to the proposed method, there is only a requirement to enter an index with each data item during its writing into the memory, which indicates the position of the data item in the output data matrix, and to organize the memory in such a way that it will provide the output data items reading in the order specified by their indices.

Organization of the memory that implements the proposed ordered access method and is named the parallel ordered-access memory (POAM) is shown in Fig. 1 [36, 37].

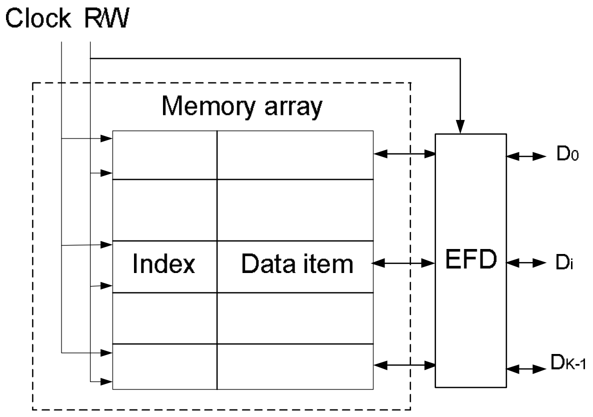

Fig. 1. POAM organization

The parallel ordered access memory consists of the memory array with P locations to store the data items and their indices, where P=kl, k is the number of the data items in the column of the input data matrix, l is the number of the data items in the row of the input data matrix. It also consists of an entering-fetching device (EFD) which enters the input data items into the memory locations, fetch the output data items from the memory locations, and forms the output data matrix with P data items, where P=mn, m is the number of the data items in the column of the output data matrix, n is the number of the data items in the row of the output data matrix. The input data items and their indices are written into the memory array by the rows of the input data matrix and the output data items are read out from the memory array by the rows of the output data matrix using R/W signal.

The input data items are written into the POAM from l ports by the rows of the matrix

$$\begin{vmatrix} ID_{0,0} & ID_{0,1} \dots & ID_{0,l-1} \\ ID_{1,0} & ID_{1,1} \dots & ID_{1,l-1} \\ \dots & & \\ ID_{k-1,0} & ID_{k-1,1} \dots & ID_{k-1,l-1} \end{vmatrix},$$

where $ID_{i,j}$ is the input data item which is placed in the i-th row (i = 0,1,…k-1) and j-th column (j = 0,1,…*l*-1) of the input data matrix.

The output data items are read out from the POAM to the m ports by the rows of the matrix

$$\begin{vmatrix} OD_{0,0} & OD_{0,1} \dots & OD_{0,n-1} \\ OD_{1,0} & OD_{1,1} \dots & OD_{1,n-1} \\ \dots & & \\ OD_{m-1,0} & OD_{m-1,1} \dots & OD_{m-1,n-1} \end{vmatrix},$$

where $OD_{s,t}$ is the output data item which is placed in the s-th row (s = 0,1,…m-1) and t-th column (t = 0,1,…n-1) of the output data matrix.

The matrix of the indices that are assigned to the items of the input data matrix appears as follows

$$\begin{vmatrix} SID_{0,0} & SID_{0,1} \dots & SID_{0,l-1} \\ SID_{1,0} & SID_{1,1} \dots & SID_{1,l-1} \\ \dots & & \\ SID_{k-1,0} & SID_{k-1,1} \dots & SID_{k-1,l-1} \end{vmatrix},$$

where $SID_{i,j}$ is the index of the input data item $ID_{i,j}$ which is placed in the i-th row (i = 0,1,…k-1) and j-th column (j = 0,1,…*l*-1) of the input data matrix.

The matrix of the indices can come into POAM together with the input data matrix or serve as the base for the ordering code calculating that is sent to the POAM.

VI. PARALLEL ORDERED-ACCESS MACHINE COMPUTATIONAL MODEL

Basing on the above described new method of the program components identification and on the new computer memory type, we propose the new computational model named the Parallel Ordered-Access Machine (POAM). As it was already mentioned above the basic idea of the proposed computational model is the use of an identification of the program components by their indices in the corresponding matrices, i.e. assigning of the indices to each instruction and to each data item, and processing row-by-row of the matrices.

As it was mentioned above the computational model comprises the problem description model which supposes the computer program development and the execution model which supposes the computer program execution. So, the POAM computational model can be implemented using two procedures: computer program development and computer program executionn.

A. Computer program development

According to the POAM computational model the following actions have to be executed in the scope of the computer program development.

Traditionally the first step supposes an algorithm development for the given problem solving and the initial, intermediate and final data items parameters definition.

The second step is dedicated to an algorithm presentation in the stage-by-stage form, when every next stage consists of the set of operations which depend of the operations of the previous stages and do not depend of the operations of the next stages. The algorithm flow graph (AFG) or its structure matrix [38] particularly can be used as such a form.

The third step is dedicated to identification of the data items and the instructions by using, for example, the AFG labeling.

Next five steps suppose to form the following matrices for each algorithm stage: the instructions matrix, the indices matrix for the instructions ordering in the frame of the computer program execution, the input variables matrix templates which are used with the aim of the each stage input data item matrices content determining that to be received in the frame of the computer program execution, the indices matrices for the input data items ordering n the frame of the computer program execution, the indices matrices for the instruction and data indices ordering n the frame of the computer program execution.

Here the index of the instruction denotes the instruction position in the instruction matrix, the index of the data item denotes the data item position in the data matrix and the index of the index denotes the index position in the index matrix.

B. Computer program execution

The diagram of the computer program execution according to the parallel ordered-access machine computational model is shown in Fig. 2.

As it can be seen, within the procedure of the computer program execution the following steps must be performed:

1. Setting the number of stages and the number of steps at each stage needed to be performed.
2. Start of the computer program execution from the first step of the first stage.
3. Loading row-by-row from the POAM the instruction, the data and the index matrices.
4. Executing of the specified by the instructions operations under the data items.
5. Indexing the results and storing them in the POAM.
6. Executing of the next steps of the first stage in the same way up to the last step. In accordance with the POAM functions the obtained results will be formed inside it as the data matrix of the corresponding stage by ordering them according to their indices.
7. Proceed to the second stage and continuing the computer program execution in the same way from the first to the last step of this stage.
8. Executing in the same way of the next stages up to the last stage. After indexing the results of the last step of the last stage and storing them in the POAM there will be formed the final data matrix.

VII. PARALLEL ORDERED-ACCESS MACHINE ARCHITECTURE AND FUNCTIONING

The structure of the parallel ordered-access machine data and instruction path is shown in Fig. 3. It consists of the data POAM, the instruction POAM, n buffers (B) and n arithmetic-logic units (ALU).

To explain in more detail the actions that have to be executed in the scope the computer program development and execution let consider the parallel ordered-access machine computational model application to the RGB2YUV algorithm implementation which is described by the following expression [39]:

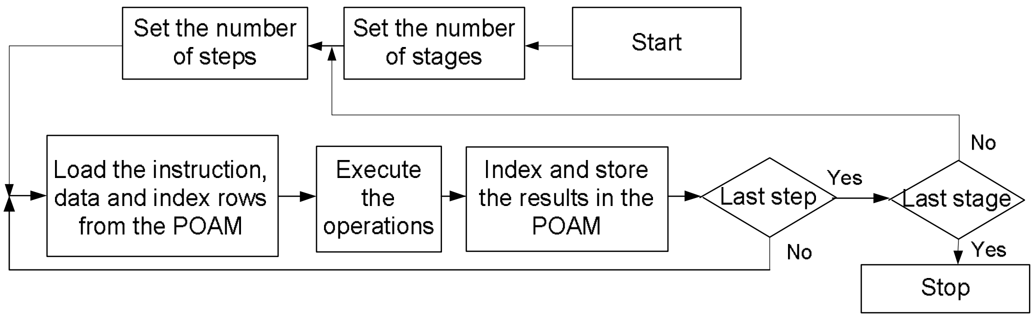

Fig. 2. The diagram of the computer program execution according to the parallel ordered-access machine computational model

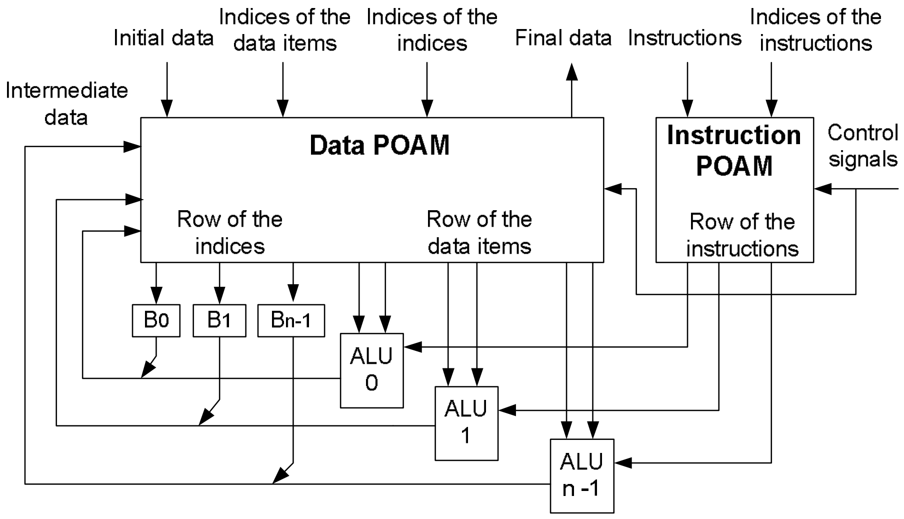

Fig. 3. The structure of the parallel ordered-access machine data and instruction path

$$Y = +K_{YR} \times R + K_{YG} \times G + K_{YB} \times B;$$
$$U = -K_{UR} \times R + K_{UG} \times G + K_{UB} \times B + C_1;$$
$$V = +K_{VR} \times R + K_{VG} \times G + K_{VB} \times B + C_1,$$

where R, G, B – the components of RGB, K – coefficients; Y, U, V – the components of YUV; C1 – the constant.

Let there are three ALUs (n=3) at the parallel ordered-access machine data and instruction path (Fig. 3) each of which has an input for the instructions entering, two inputs for the input data items entering and one output for the output data items issuing, which are the binary digits.

For an algorithm presentation in the stage-by-stage form let use its flow graph which is presented in the form that every next stage of which consists of the set of the operations which depend of the operations of the previous stages and do not depend of the operations of the next stages (Fig. 4). There are four such independent stages in this RGB2YUV algorithm flow graph. An algorithm includes 9 instructions of multiplication (mul), 7 instructions of addition (add), and 1 instruction of subtraction (sub). To this list also the instructions of transition (tr) and no-operation (nop) are added. The common quantity of the initial data items is equal to 20 (each of the values R, G, B is used 3 times, and the value C1 is used 2 times), the quantity of the intermediate data items is equal to 24, and the quantity of the final data items is equal to 3.

For the instructions and the data items identification the AFG labeling can be used. The instructions are tied to the vertices and the ALU inputs that are connected with the instruction POAM outputs, the data items are tied to the arcs and the ALU input ports that are connected with the data POAM outputs, and the indices are tied to the ALU output ports that are connected with the data POAM inputs. The main condition of the correct labeling of the AFG vertices and arcs are an unambiguous their binding to the respective inputs and outputs ports of the particular ALU.

Basing on this labelled AFG the matrices of the instructions for each of four stages can be formed.

One possible version of the instruction matrix for the first stage is presented in Fig. 5.

$$\begin{vmatrix} \text{mul} & \text{mul} & \text{mul} \\ \text{mul} & \text{mul} & \text{mul} \\ \text{mul} & \text{mul} & \text{mul} \\ \text{tr} & \text{tr} & \text{nop} \end{vmatrix}$$

Fig. 4. The instructions matrix for the AFG first stage

Similarly the instruction matrices for each of the next three stages can be presented as it is shown in Fig. 6.

$$\underset{2}{\begin{vmatrix} \text{add} & \text{add} & \text{sub} \\ \text{tr} & \text{tr} & \text{tr} \\ \text{tr} & \text{tr} & \text{nop} \end{vmatrix}} \quad \underset{3}{\begin{vmatrix} \text{add} & \text{add} & \text{add} \\ \text{tr} & \text{tr} & \text{nop} \end{vmatrix}} \quad \underset{4}{\begin{vmatrix} \text{add} & \text{add} & \text{tr} \end{vmatrix}}$$

Fig. 5. The instruction matrices for the AFG 2th, 3th, and 4th stages

After indexing of the instructions the matrix of their ordering indices for the first stage will have a view as it is shown in Fig. 7.

$$\begin{vmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & \text{ni} & 42 \end{vmatrix}$$

Fig. 6. The matrix of the instruction ordering indices

Here an index points the instruction position in the executable instruction matrix. As it can be seen from the Fig. 5 and 7, three first rows of the instruction matrix consist of only the same instruction MUL, and the 4th row consists of two instructions TR and one instruction NOP. Label ni means “no instruction”.

Similarly the matrices of the instruction ordering indices for each of the next three stages can be presented as it is shown in Fig. 8.

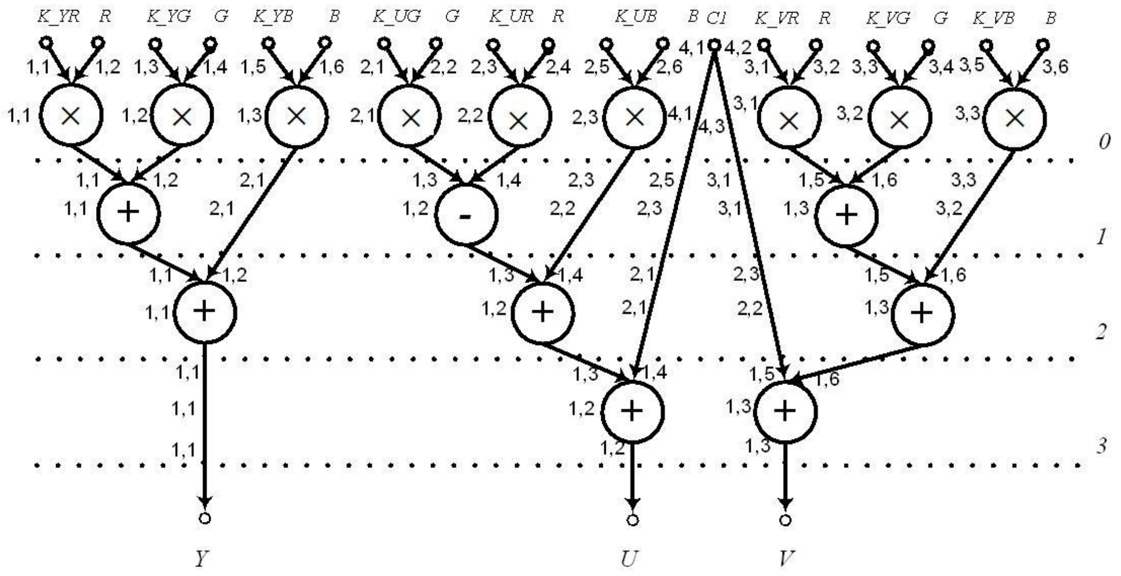
K_YR R K_YG G K_YB B K_UG G K_UR R K_UB B C1 K_VR R K_VG G K_VB B
0
1
2
3
Y
U
V

Fig. 7. Labeled RGB2YUV algorithm flow graph

$$\underset{2}{\begin{vmatrix} 11 & 13 & 12 \\ 21 & 22 & 23 \\ 31 & 32 & \text{ni} \end{vmatrix}} \qquad \underset{3}{\begin{vmatrix} 11 & 12 & 13 \\ 21 & 22 & \text{ni} \end{vmatrix}} \qquad \underset{4}{\begin{vmatrix} 11 & 12 & 13 \end{vmatrix}}$$

Fig. 8. The matrices of the instruction ordering indices for the AFG 2th, 3th, and 4th stages

The matrix of the input data items for the first AFG stage and the matrix of their ordering indices are shown in Fig. 9. Label nd means “no data item”.

$$\begin{vmatrix} K_{YR} & R & K_{YG} & G & K_{UG} & G \\ K_{UR} & R & K_{UB} & B & K_{UB} & B \\ C1 & C1 & K_{VR} & R & K_{VG} & G \\ K_{VB} & B & \text{nd} & \text{nd} & \text{nd} & \text{nd} \end{vmatrix} \qquad \begin{vmatrix} 11 & 12 & 13 & 14 & 15 & 16 \\ 21 & 22 & 23 & 24 & 25 & 26 \\ 31 & 32 & 33 & 34 & 35 & 36 \\ 41 & 42 & \text{nd} & \text{nd} & \text{nd} & \text{nd} \end{vmatrix}$$

Fig. 9. The matrix of the data items for the AFG first stage and their indices

Similarly the matrices of the data ordering indices for each of the next three stages can be presented as it is shown in Fig. 10.

$$\underset{2}{\begin{vmatrix} 11 & 12 & 13 & 14 & 15 & 16 \\ 21 & \text{nd} & 23 & \text{nd} & 25 & \text{nd} \\ 31 & \text{nd} & 33 & \text{nd} & \text{nd} & \text{nd} \end{vmatrix}} \qquad \underset{3}{\begin{vmatrix} 11 & 12 & 13 & 14 & 15 & 16 \\ 21 & \text{nd} & 23 & \text{nd} & \text{nd} & \text{nd} \end{vmatrix}} \qquad \underset{4}{\begin{vmatrix} 11 & \text{nd} & 13 & 14 & 15 & 16 \end{vmatrix}}$$

Fig. 10. The matrices of the input data ordering indices for the 2th, 3th, and 4th stages

Here an index points the data item position in the matrix of the processable data. First digit of the indices points the number of the row of the data or instruction matrix, and second digit of the indices points the number of the column of the data or instruction matrix. Particularly, the indices of the data item R are 12; 16; 34, and the first from the left side instruction of the stage 4 is indexed by 13.

After the program components will be stored in the data POAM and the instruction POAM they will be processed in the scope of the computer program execution according to the diagram from Fig. 2. If one instruction is implemented for one step then 10 steps will be needed to perform the RGB2YUV algorithm.

VIII. NEW COMPUTER ARCHITECTURE EFFICIENCY ESTIMATION

As it can be seen from Fig. 3, where the structure of the parallel ordered-access machine data and instruction path is presented, the main component which defines its throughput is the POAM. We have shown above, that the main advantages of the POAM compared to the RAM, the most used conventional memory type, are the following:

- The POAM is a multiport conflict-free access memory whereas the RAM is single-port.
- The POAM provides the data items ordering in the matrixes simultaneously with their storing. This operation is frequently used and is usually time-consuming.
- There is no need to save the data items locations addresses in the POAM. It is enough here to point out the index of the data item in the write mode. And opposite, the RAM usage requires the data item address to be stored which complicates the computer organization and increases the hardware volume.
- As the data items are not tailed to the memory locations the POAM does not have complex and slow address decoders as the RAM has. The functions of the data items ordering can be disintegrated, which makes it possible to decrease the time delays and to increase the memory clock frequency. So, the POAM bandwidth does not depend on the capacity.

We have designed and implemented in the FPGA some POAM IP cores [39] and some application-specific processors based on the POAM. For example, the bandwidth of the 8-port 32-bit POAM is approximately 200 Gb/sec. This opens up the opportunities to achieve significantly higher productivity in the single-processor computer architecture.

IX. CONCLUSIONS

The main direction of the computer systems performance increasing is their parallelization. Therefore, the parallel multiprocessor systems on a chip became the mainstream products of the microprocessor industry, the parallel ultrascale computer systems were created for complex computing problems solving, and the parallel software is being developed for their effective use. On the other hand, more software is being developed for the single-processor execution. To find the way for increasing the performance of these single processors, which will be the base for the future multiprocessor systems, is a big challenge. To address this challenge the following improvements have to be done: reduced memory wall and provided parallel conflict-free memory access. But this is not compatible with John von Neumann's computational model which serves as the base of today's computer systems.

We see the way to realize it in creation of the new computational model and computer architecture on its base which does not have limitations of the conventional computational models, including John von Neumann one.

Having analyzed the known stack-access machine, content-access machine and random-access machine computational models based respectively on a belt, content and address memory access, we propose the method of the program components identification according to their positions in the matrices and relevant computational model, based on the ordered memory access. We have named this computational model the parallel ordered-access machine because of its base – the parallel ordered-access memory. It is shown, that the POAM computational model can be implemented using two procedures: computer program development and computer program execution, which are described in the paper in detailr.

The article describes the computer architecture, which implements proposed computation model and owing to this does not have such a limitation as the memory wall and provides parallel conflict-free memory access. An example of the parallel ordered-access machine computational model implementation and the computer architecture is presented at the end of the paper.

REFERENCES

[1] Hennessy J. L., and Patterson D. A., Computer Architecture: A Quantitative Approach, 5th ed., Boston, MA: Morgan Kaufmann Publishers, 2011.

[2] Stallings, W., Computer Organization and Architecture, 5th ed., NY: Macmillan Publishing Company, New York, 2000.

[3] Tanenbaum, A., Structured Computer Organization, 6th ed., Todd Austin. Year: 2012. Pages: 801. Publisher: Pearson.

[4] Melnyk A. O. Computer architecture. Lutsk regional printing. Lutsk. 2008.

[5] Hamacher, V. C., Vranesic, Z. G., and Zaky, S. G., Computer organization, McGraw-Hill Higher Education, 1995.

[6] Goyal, A. and Agerwala, T., Performance analysis of future shared storage systems, IBM Journal of Research and Development, Vol. 28, No. 1, 1984, pp. 95–107.

[7] El-Rewini, H. and Abd-El-Barr, M., Advanced computer architecture and parallel processing, John Wiley, 2005.

[8] Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

[9] Ibbett, R. N. and Topham, N. P., Architecture of High Performance Computers II, Springer-Verlag, 1989.

[10] Lewis, T. G. and El-Rewini, H., Introduction to Parallel Computing, Prentice-Hall, 1992.

[11] Moldovan, D., Parallel Processing, from Applications to Systems, Morgan Kaufmann Publishers, 1993.

[12] Patterson, D.A., and Hennessy, J.L. Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann Publishers, 4th Edition, Inc.2005.

[13] Georg Hager, Gerhard Wellein. Introduction to High Performance Computing for Scientists and Engineers. CRC Pres, 2011.

[14] Wilkinson, B., Computer Architecture: Design and Performance, 2nd ed., Prentice-Hall, 1996.

[15] Agervala, T. and Arvind, Data Flow Systems, Computer, Vol. 15, No. 2, Feb, 1982, pp. 10–13.

[16] Gajski, D. D., Padua, D. A., Kuck, D. J., and Kuhn, R., A Second Opinion on Data Flow Machines and Languages, Computer, Vol. 15, No. 2, Feb, 1982, pp. 58–69.

[17] Gurd, J. andWatson, I., A Practical Data Flow Computer, Computer, Vol. 15, No. 2, Feb, 1982, pp. 51–57.

[18] Le Guernic, P., Benveniste, A., Bournai, P., and Gautier, T., SIGNAL – A Data Flow-Oriented Language for Signal Processing, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 2, April, 1986, pp. 362–374.

[19] Hartimo, I., Kronlof, K., Simula, O., and Skytta, J., DFSP: A Data Flow Signal Processor, IEEE Trans. on Computer, Vol. C-35, No. 1, Jan, 1986, pp. 23–33.

[20] O. Flygt. Computer Architecture. Computational Models. http://homepage.lnu.se/staff/oflmsi/DA2022/Material/CH01.pdf

[21] Schoeberl, M., Design and Implementation of an Efficient Stack Machine. In Proceedings of the 12th IEEE Reconfigurable Architecture Workshop, RAW 2005, Denver, Colorado, USA, April, 2005.

[22] Koopman, P. J., Stack computers: the new wave, Halsted Press, 1989.

[23] Bulman, D. M., Stack computers: an introduction, Computer, Vol. 10, No. 5, 1977, pp. 14–16.

[24] Batcher, K., Staran Parallel Processor System Hardware, Proc. National Computer Cont. AFIPS., 1974, pp. 405–410.

[25] Stormon, C. e. a., A General-purpose CMOS Associative Processor IC and System. IEEE Micro, Vol. 12, No. 6, Dec, 1992, pp. 68–78.

[26] Potter, J., Associative Computing – A Programming Paradigm for Massively Parallel Computers, N.Y.: Plenum Publishing, 1992.

[27] Burks, A. W., Goldstine, H. H., and von Neumann, J., Preliminary discussion of the logical design of an electronic computing instrument, Tech. Rep. Report Prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481, 1946.

[28] McCartney, S., ENIAC: The Triumphs and Tragedies of the World's First Computer, New York: Walker and Company, 1999.

[29] Blaauw, G. and Brooks, F., Computer Architecture: Concepts and Evolution. Reading, MA: Addison-Wesley, 1997.

[30] Ceruzzi, P. E., A History of Modern Computing, MA: MIT Press, Cambridge, 1998.

[31] Cortada, J. W., Historical Dictionary of Data Processing, Volume 1: Biographies; Volume 2: Organization, Volume 3: Technology., CT: Greenwood Press, Westport, 1987.

[32] Augarten, S., Bit by Bit: An Illustrated History of Computers, London: Unwin Paperbacks, 1985.

[33] Mollenho, C. R., Atanasoff: The Forgotten Father of the Computer, IA: Iowa State University Press, Ames, 1988.

[34] Polachek, H., Before the ENIAC. IEEE Annals of the History of Computing, Vol. 19, No. 2, June, 1997, pp. 25–30.

[35] Wilkes, M. V., Wheeler, D. J., and Gill, S. The Preparation of Programs for an Electronic Digital Computer, Addison-Wesley, Cambridge, 1951.

[36] Melnyk A. O. Computer Memory with Parallel Conflict-Free Sorting Network-Based Ordered Data Access. Recent Patents on Computer Science, 2015, Vol. 8(1), pp. 67–77.

[37] Melnyk A. O. Ordered-Access Memory. Lviv Polytechnic National University Publishing. 2014.

[38] Melnyk A. O., Iakovlieva I. D. OCA – Graphical System for Algorithm Structure Analysis and Processing. Korea Academia-Industrial Cooperation Society (KAIS): Smart Computing Review, Vol. 2. – No. 2. April-2012. – P. 171–184.

[39] Szeliski R. Computer Vision: Algorithms and Applications. Springer, 2011

Anatoliy O. Melnyk is a Head of Computer Engineering Department at Lviv Polytechnic National University since 1994. He graduated from Lviv Polytechnic Institute with the Engineer Degree in Computer Engineering in 1978. In 1985 he obtained his Ph.D. in Computer Systems from Moscow Power Engineering Institute. In 1992 he received his D.Sc. degree from the Institute of Modeling Problems in Power Engineering of the National Academy of Science of Ukraine. He was recognized for his outstanding contributions to high-performance computer systems design as a Fellow Scientific Researcher in 1988. He became a Professor of Computer Engineering in 1996. Since 1982 to 1994 he has been a Head of Department of Signal Processing Systems at Lviv Radio Engineering Research Institute. Since 1994 to 2008 he has been Scientific Director of the Institute of Measurement and Computer Technique at Lviv Polytechnic National University. Since 1999 to 2009 he has been Dean of the Department of Computer and Information Technologies at the Institute of Business and Perspective Technologies, Lviv, Ukraine. He has served since 2000 as President and CEO of Intron ltd. He has also been a visiting professor at Kielce University of Technology, University of Information Technology and Management, Rzeszow, University of Bielsko-Biala. Currently he is a visiting professor at the Department of Numerical Analysis and Programming of John Paul II Catholic University of Lublin..

He is an editor in chief of the proceedings “Computer Systems and Networks” and of the journal “Advances in Cyber-Physical Systems”. He is a head of the international conference “Advanced Computer Systems and Networks: Design and Application” and of the scientific workshop “Cyber-Physical Systems: Achievements and Challenges”. He has taken part in a large number of research projects in the field of computer systems as a project leader. He has published 9 monographs, 1 handbook and over 400 scientific papers and patents. He is a member of IEEE, ACM, IEE, IACSS, AESU.