Acta Informatica Pragensia
2(2), 2013, 51–58, DOI: 10.18267/j.aip.23
Online: aip.vse.cz

Section:
Peer-reviewed papers

Subgraph Colouring Technique for Computer with Data Driven Computation Model

Liberios Vokorokos[1], Branislav Madoš[1], Norbert Ádám[1], Marek Čajkovský[1]

[1] Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic

{liberios.vokorokos, branislav.mados, norbert.adam, marek.cajkovsky}@tuke.sk

Abstract: Paper deals with the data driven computation control paradigm, which is alternative to the mainstream control flow model of program execution control. In the second part of the paper we are introducing computer architecture that is proposed at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, according to the data driven computation control model with use of tile computing principles. In the final part of the paper, subgraph colouring technique that is unique characteristic of proposed computer architecture is described.

Keywords: Data flow, Data driven, Data flow graph, Subgraph mapping, Subgraph colouring

1 INTRODUCTION

Research oriented in multiple directions in the field of high performance parallel computer architectures brought in 60's and 70's of the last century the popularity to the class of parallel computers, that are based on the data driven control of program execution, generally referred as the data flow architectures. Instead of instructions controlled flow of program execution with use of program counter concept, the flood of operands is used to control the program execution process.

Program for the data flow computer is prepared in the form of directed graph G = (V, E) and its representation in machine readable form is called Data Flow Graph (DFG). Instructions, called actors, are represented as the vertexes $v_i \in V$ in the DFG. Edges $e_i \in E$ are interconnecting source and destination actors in the DFG and are representing paths in which operands are flowing through the DFG in time of program execution. Tokens are representing presence of operands on the edges. Edge is not only the interconnection between actors but also determines which actor is source of operands and which is the consumer and assures proper execution of the DFG. Load of the DFG into the operating memory of data flow computer is called data flow graph mapping and instructions are mapped into the memory to the activation frames. Instructions that are placed in operating memory are fired immediately after they became executable (instruction is executable when all operands that are needed for instructions execution are present). Instruction is activated when it is fired and all hardware resources that are needed for instructions execution are available.

There are pure data flow architectures, which are divided into two groups called static and dynamic data flow computer architectures, according to the model of execution of DFG. Static model allows only one token placed on the edge; dynamic model allows more than one token placed on the edge at the same time. Each token has its own tag in dynamic model, to identify its position in the token flood. Tokens must differ in the tag, when are placed on the same edge and there is process of searching of the tokens with the same tag which is called matching. Unlike dynamic model, there is disadvantage of the static dataflow computer architecture in impossibility to use dynamic forms of parallelism, including loops and recursion. Dynamic data flow model allows dynamic forms of parallelism, both loop and recursive, that appear in the time of program execution [1, 2, 3, 4, 5, 6, 7].

Besides of pure data flow architectures there is possibility to define also category of hybrid data flow architectures. Those can be seen as the computer architectures which are combining both control flow and data flow control mechanisms in the program execution control.

2 PROPOSED ARCHITECTURE

Architecture of computer-on-the-chip with static data driven computation model, proposed as the part of the research of dataflow computer architectures at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, is developed according to the principles of tile computing. It is possible to describe this approach as the effort to build highly scalable architectures of computers with use of the basic building blocks that are standardized and are reused in architecture many times in the manner that allows easy scalability with little or no change of used components.

Proposed architecture consists of *Processing Elements (PE)*, which are forming *Processing Array (PA)*. PA consists of n × n bi-directional layout of PEs, where *n* is the number of PEs placed on the edge of PA. Besides of all each PE comprises of the operating memory in the form of *Activation Frames Store (AFS)*, *Arithmetic and Logic Unit, (ALU)* and *Control Unit (CU)*. At the edges of PA are *Input/Output Elements (IO)* localised and each of IO is close connected to the neighbouring PE (Fig. 1.). Number of IOs is 4n, where *n* is the number of PEs placed on the edge of PA. There is ratio (R) between the number of PEs and IOs and this ratio can be determined by the formula (1).

$$R = n^2 / 4n \tag{1}$$

IO element is used for creation of communication connection with the components of the computer that are localised outside of the chip, or in multichip configuration of the computer IOs are forming bridges that are interconnecting PAs of neighbouring chips.

Elements of the architecture, represented by PEs and IOs are interconnected via communication networks. One of networks is formed by local communication channels, interconnecting pairs of neighbouring elements and allows concurrent communication of elements pairs. Another network is forming global communication network and is interconnecting all elements in the form of the bus. Global communication network do not allow concurrent communication of element pairs and is dedicated to the data flow program mapping.

Prototype of the computer built according to the designed architecture consists of 64 PEs distributed across the chip in 8 x 8 bi-dimensional layout, with 32 IOs located at the edges of the PA. Field Programmable Gate Array (FPGA) technology is used. As the hardware of the computer prototype Xilinx Spartan 3AN development board is used and as the software development tool Xilinx ISE Design Suite was used. Detailed information about architecture and computer prototype can be found in [3, 8, 9, 10].

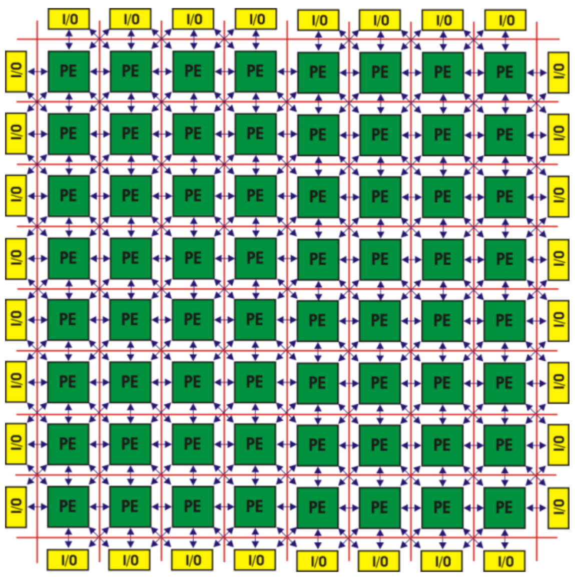

Fig. 1. Processing array of the computer consists of 64 PEs and 32 IOs.

2.1 ACTIVATION FRAME

Instructions are represented via 144-bit vectors that are mapped in the form of activation frames into the activation frames stores of respective processing elements in time of data flow graph mapping, when computer and memory are in *Load mode*. Each instruction vector consists of components according to the Fig. 2.

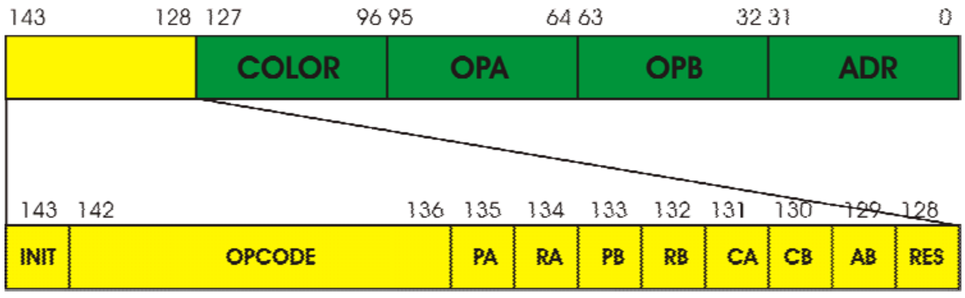

Fig. 2. Instruction represented by 144 bit vector stored in activation frame.

Where

INIT is 1bit long indicator of instruction executability

OPCODE is 7bit long operation code of instruction

PA is 1bit long indicator of operand A demand for instruction execution

RA is 1bit long indicator of operand A availability

PB is 1bit long indicator of operand B demand for instruction execution

RB is 1bit long indicator of operand B availability

CA is 1 bit long indicator of immediate operand A

CB is 1 bit long indicator of immediate operand B

AB is1bit long indicator of instructions target operand

RES is 1 bit reserved

COLOR is 32 bit long COLOR tag of subgraph

OPA is 32 bit long operand A

OPB is 32 bit long operand B

ADR is 32 bit long target address of the instruction

Architecture brings concept of instruction initialization. Initialization is the basic prerequisite of instructions execution. It means that it is not possible to fire and execute instruction, which is not initialized. Initialization can occur in the time of data flow program mapping, when instructions are mapped into the activation frame store in initialized form, or can occur in time of program execution, when instructions are initialized with use of special instructions, that are part of the instructions set architecture. Instruction is initialized when INIT bit of instructions vector is set to 1. Another process is called deinitialization of the instruction. It can occur in time of data flow program mapping, when instruction is mapped into memory as deinitialized, or it is possible to deinitialize instruction in time of program execution via special instructions, that are part of the instructions set architecture. Another form of this process occurs when instruction is executed, because immediately after execution, instruction is deinitialized. There is possibility to use multiple cycles of de/initialization process in time of program execution and this is unique characteristic of this data flow architecture.

3 DATA FLOW SUBGRAPH COLOURING

Program of the data flow computer is prepared in the form of directed graph. It is possible to partition the data flow graph into the subgraphs and this process is iterative. Let us denote subgraphs with their respective colours. In proposed architecture, each subgraph is denoted with 32-bit COLOR tag, which is part of instruction vector (bits 127:96). Each instruction stored in the memory is tagged with colour of the subgraph as it is shown in the example at Fig. 3., where "yellow" subgraph has COLOR tag $(00000000\ 00000000\ 00000000\ 00000100)_2$ (leading zeros of 32-bit COLOR tag vector are omitted in the Fig. 3 in favour of more readable notation), "green" subgraph has COLOR tag $(00000000\ 00000000\ 00000000\ 00000110)_2$ and "white" subgraph has COLOR tag $(00000000\ 00000000\ 00000000\ 00000111)_2$.

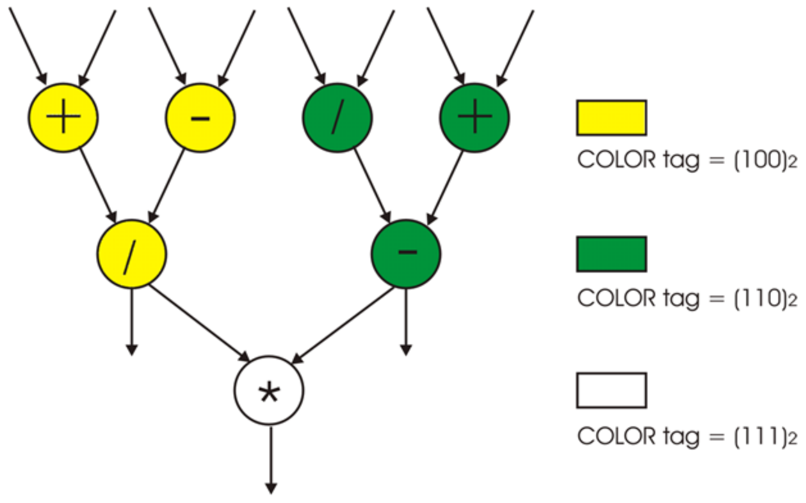

Fig. 3. Example of data flow graph partitioned into 3 subgraphs and tagged with different COLOR tags.

Proposed architecture brings concept of de/initialization operations not only for the entire data flow graph via IG (Initialize Graph) and KG (Kill Graph) instructions that are part of the instructions set architecture, but also of de/initialization of subgraph operations via ISG (Initialize SubGraph), ISGO (Initialize SubGraph with Operands) and KSG (Kill SubGraph) instructions.

In the basic concept of the COLOR tag usage, all instructions mentioned above has operand, which is 32-bit long and represents COLOR tag of subgraph on which instruction is applied. It is possible for example to initialize each of subgraphs with use of instruction IG $(00000000\ 00000000\ 00000000\ 00000100)_2$, IG $(00000000\ 00000000\ 00000000\ 00000110)_2$ or IG $(00000000\ 00000000\ 00000000\ 00000111)_2$. Because of specific construction of operating memory of the computer, operation is applied in parallel on all activation frames stored in memory that are tagged with respective COLOR tag in one machine cycle. However, this straightforward approach has its shortcomings. If we are considering "green" and "yellow" subgraphs as two partitions of one subgraph which is needed to be initialized, it is not possible to perform this operation in other way but to use sequentially two IG instructions with $(00000000\ 00000000\ 00000000\ 00000110)_2$ and $(00000000\ 00000000\ 00000000\ 00000111)_2$ operands respectively. This means not only use of more instructions of the program, but also more machine cycles are needed. That is why more complex approach was chosen.

Instructions which are using COLOR tag are not unary, but have two operands. First is COLOR tag which is 32-bit vector, as mentioned above, second operand is 7-bit vector that represents the number of bits of COLOR tag, which are compared, when it is evaluated, if instruction on which de/initialization operation is applied is part of target subgraph. Bits are counted from MSB of COLOR tag vector. It is meaningless to compare 0 bits of COLOR tag vector because in that case operation is applied on all activation frames stored in memory (This is performed by IG and KSG operations respectively) and there is no possibility to compare more than 32 bits of COLOR tag vector. That is why value of the second operand is restricted to the range of <1:32> in decimal.

Version of instructions with two operands is powerful, because it can target more than one subgraph at once and that means operation of de/initialization can be performed on more subgraphs at the same time, in one machine cycle. In the example mentioned above, if IG operation with operands $(00000000\ 00000000\ 00000000\ 00000110)_2$, $(0111111)_2$ or $(00000000\ 00000000\ 00000000\ 00000111)_2$, $(0111111)_2$ is applied, subgraphs with 31 MSB bits set to the value $(00000000\ 00000000\ 00000000\ 0000011)_2$ are targeted. Because only one bit left, there are two possible subgraphs, which can be targeted, with 1 LSB bit set to 0 and 1 respectively. In the example on Fig. 3 those subgraphs are „green" and „white".

If the first operand of instruction is set to the value $(00000000\ 00000000\ 00000000\ 000001xx)_2$ where xx is the 2 bit vector of any value and second operand is set to the value $(0111110)_2$, there are 30 MSB bits compared and vector consisting of 2 LSB bits left uncompared. It means 4 subgraphs with 2 LSB bits set to the values $(00)_2$, $(01)_2$, $(10)_2$, $(11)_2$ are targeted. In the example shown in Fig. 3, all 3 subgraphs meet the condition and are targeted with this instruction.

In the example shown on Fig. 3, there is no possibility to target "yellow" and "white" subgraphs at once without targeting "green" subgraph.

4 CONCLUSION

Paper briefly introduced the data flow paradigm of computation control as the alternative to the mainstream control flow paradigm and described main features of the computer architecture with data flow computation control that is proposed at the Department of Computers and Informatics, Faculty of

Electrical Engineering and Informatics, Technical University of Košice. Special attention was paid to the description of the data flow subgraph colouring, which is the specific and unique feature of proposed architecture, and is not implemented in other architectures. Subgraph colouring allows with use of instructions that are part of instructions set architecture initialization and deinitialization of other instructions not only in time of program load but also in time of the program execution. There is possibility to use multiple cycles of de/initialization process in time of program execution and this feature brings to the architecture certain features of control flow model. Main advantage of this approach is that this feature makes it possible to preserve construction simplicity of the static data flow architectures and at the same time allows the implementation of language constructs such as the recursion and loops; features that are achieved in dynamic data flow architectures but only for the price of greater complexity of the design. Another advantage is the opportunity to end the execution of whole block of instructions in any time of its execution, in case there is condition fulfilled, that execution of this instructions block is unnecessary and there is possibility to deinitialize all instructions of the block in one machine cycle. This, together with other features of architecture, allows increase in the effectiveness of the program code execution without significant distortion of the benefits of data flow control of the program execution.

ACKNOWLEDGEMENT

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10. Project is being solved at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice.

5 REFERENCES

[1] ARVIND, A., CULLER, D.E.: *The tagged token dataflow architecture (preliminary version)*, Tech. Rep. Laboratory for Computer Science, MIT, Cambridge, MA, 1983.

[2] DENNIS, J.,B., MISUNAS, R.P.: *A Preliminary Architecture for a Basic Data Flow Processor*", Proceedings of the 2nd Annual Symposium on Computer architectures., 1974.

[3] MADOŠ, B., BALÁŽ, A.: *Data Flow Graph Mapping Techniques of Computer Architecture with Data Driven Computation Model*, SAMI 2011 Proceedings of 9th IEEE International Symposium on Applied Machine Intelligence and Informatics, Smolenice, Slovakia, 27. - 29. january 2011, pp. 355 - 359, IEEE Catalog Number: CFP1108E-CDR, ISBN 978-1-4244-7428-8.

[4] SHIMADA, T., HIRAKI, K., NISHIDA, K, SEKIGUGHI, S,: *Evaluation of a prototype Dataflow Processor of the SIGMA-1 for Scientific Computations*", In: Proc. 13th Annual Symposium On Computer Architecture, 1986, pp. 226-234.

[5] VEEN, A.: *Dataflow Machine Architecture*, ACM Computing Surveys, December 1986, pp. 365-396.

[6] VLASOV, V.V., KRAYNIKOV, A.V., KURDIKOV, B.A.: *A Data Flow Computer System*, In: Izvestiya LETI (Proceedings of Leningrad Electrotechn.Inst.), St.Petersburg, Vol. 436, 1991, pp. 3-7.

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

[8] 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 : Óbuda University, 2011.

[9] VOKOROKOS, L., MADOŠ, B., BALÁŽ, A., ÁDÁM, N.: *Architecture of multi-core computer with data driven computation model*, Acta Electrotechnica et Informatica, december 2010, Košice, Slovakia, pp. 20-23, ISSN 1335-8243.

[10] VOKOROKOS, L., MADOŠ, B., BALÁŽ, A., ÁDÁM, N.: *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. 1 - 17, 2013 , ISSN 1785-8860.