A Novel Memory-centric Architecture and Organization of Processors and Computers

Danijela Efnusheva, Goce Dokoski, Aristotel Tentov, Marija Kalendar
SS. Cyril and Methodius University - Faculty of Electrical Engineering and Information Technologies
Karpos II bb, PO Box 574, 1000 Skopje, Macedonia
E-mail: {danijela, gocedoko, toto, marijaka}@feit.ukim.edu.mk

***Abstract*—The modern computer systems that are in use nowadays are mostly processor-dominant, which means that their memory is treated as a slave element that has one major task – to serve execution units data requirements. This organization is based on the classical Von Neumann's computer model, proposed seven decades ago in the 1950^{ties}. This model suffers from a substantial processor-memory bottleneck, because of the huge disparity between the processor and memory working speeds. In order to solve this problem, in this paper we propose a novel architecture and organization of processors and computers that attempts to provide stronger match between the processing and memory elements in the system. The proposed model utilizes a memory-centric architecture, wherein the execution hardware is added to the memory code blocks, allowing them to perform instructions scheduling and execution, management of data requests and responses, and direct communication with the data memory blocks without using registers. This organization allows concurrent execution of all threads, processes or program segments that fit in the memory at a given time. Therefore, in this paper we describe several possibilities for organizing the proposed memory-centric system with multiple data and logic-memory merged blocks, by utilizing a high-speed interconnection switching network.**

***Keywords*: Explicit parallelism, Field Programmable Gate Array (FPGA), high-performance computing, processor architecture and organization, processing in memory.**

I. INTRODUCTION

Computers are an important part of the modern human life, which cannot be imagined without the use of these electronic devices. The purpose of these complex systems is to perform data processing, data storage, data movement to and from the computer, and control of the whole system operation, [1]-[3]. These functionalities are provided by several basic computer components, including: central processing unit (CPU), memory (which is generally hierarchically organized), input/output devices and interconnection busses responsible for movement of data, address and control signals.

The central processing unit is one of the most complicated parts of the computer system that has ever been created by the human beings. The processor has the main role in the computer system, since it handles the instruction and data flow, controls the communication with the memory and input/output devices and thus coordinates the whole system operation, [4]-[6]. As a result, computer architects constantly face with the challenge to develop novel architectural solutions that can maximize the computer performance, while retaining the cost, power and functional requirements. Regarding this, they should consider three aspects of computer architecture design, such as: instruction set architecture, organization (memory system, memory-processor interconnect, internal processor), and hardware logic design.

The constant race of the various computer technologies resulted in a wide range of processor architectures, including CISC, RISC, Superscalar, VLIW, EPIC, Vector, [6]-[13], and Data Flow, [14]-[20]. These architectures provide various benefits and drawbacks, and are characterized with different ways of parallel programs execution, organization and instruction set architecture. Each of them is developed with the intention to overcome some of the problems of its predecessors and thus to provide better computing performances. However, besides the great advances in computer systems technology, their architecture and organization, the evolvement of multi-cores and various parallelization techniques for program execution, current computer architectures are still dominantly based on the classic Von Neumann's model, [21]-[23]. Main focus and prime role in this type of computer architectures and organizations is dedicated to the execution units of different type, and the memory is treated as slave element which main function is to serve the execution units data requirements.

The existing model of processor-centric computer architecture allows performance scale only if these two conditions hold: the processing core has sufficient work to do, so it can mitigate the cache miss latencies, and the processor has enough bandwidth to load the changes into the cache memory without excessive delay. However, the contemporary technologies for memory production can't cope with processor's requirements for data speed and bandwidth. As a consequence, there exists definitely a substantial gap of more than a couple of times between the processor working frequency, and the available memory data transfer speed, [24]-[27]. As a result, superscalar processors, [28], which are capable to execute several instructions per clock cycle, always lack of data, due to lower memory working frequency and reduced number of internal processor registers. Moreover, the Itanium EPIC processor, [29], did not manage to achieve the expected success, because of the problems with the memory speed.

A few decades ago, in the 1990[ties], some researchers predicted that the memory behavior would be preponderant over the global performance of the computer system. Their proposals suggested integration of the memory and processing elements into a single chip, creating memories with processing capacity. This merged chip is known as: smart memory, computational memory (C-RAM), [30]-[34], processor in memory (PIM), [35]-[41], intelligent RAM (IRAM), [42]-[49], etc. Recent work in this area lead to several architectural approaches, which can be classified based on the role of the merged chip: main processor(s) in the system, special-purpose processor, co-processor or intelligent memory system, [24]. For example, IRAM, [48] is implemented as a vector co-processor to a general purpose MIPS processor into the VIRAM single-chip computer.

The aim of this paper is to propose a novel memory-centric processor architecture that provides a stronger merge between memory and processing elements. This is achieved by adding a processing hardware directly to the memory blocks used for storing programs, thus allowing simpler instruction decode and execution, easier management of data requests, and direct communication between the program (code) and data memory blocks, without the use of registers. These logic-memory merged chips are named as self-executing units. The memory-centric architecture should be organized to work with multiple self-executing units, in order to provide concurrent execution of all threads, processes or program segments that fit in the memory, at a given time. Therefore, in this paper we propose and evaluate several models of computer system design with multiple data memory blocks and self-executing units, connected via high-speed interconnection switching network.

The paper is organized in five sections. Section two presents the current state, discussing a variety of modern processor architectures and organizations that are in use today. Section three describes the novel memory-centric architecture and its basic building blocks, providing details about the hardware design and its verification. The next section proposes several ways of organizing the proposed memory-centric architecture with multiple self-executing units. The paper ends up with conclusion, stated in the last section.

II. State of the Art

The complexity of modern processor architectures and the constant race of various computer technologies resulted in a wide range of computer architectures, each with its own advantages and disadvantages, but with the ultimate goal to increase the overall computer system performances. Therefore, the research of computer architects was aimed at developing various mechanisms for parallel computing that will provide efficient utilization of the system hardware resources. Generally, there are three different forms of parallel computing that have been created, including: instruction- (execution of more than one instructions in a single processor cycle), data- (execution of single instruction stream on multiple data streams) and thread-level parallelism, (concurrent execution of unrelated and distinct tasks), [1]. Most modern computer systems support several types of parallel processing in order to achieve better computing performances.

One of the first computer architectures, such as the Intel IA-32, belongs to the Complex Instruction Set Computer (CISC) design which takes advantage of microcode and supports a wider range of variable-length instructions, [6]. In order to reduce the complexity of these instructions and to provide hardware-based control of their execution, Reduced Instructions Set Computing (RISC) was introduced, [7]. Further research led to the idea that dividing the work of a single processor to multiple execution units would speed up the instructions execution. This resulted with superscalar and Very Long Instruction Word (VLIW) architectures that were designed to take advantage of Instruction Level Parallelism (ILP). A superscalar architecture consists of a number of pipelines that are working in parallel, and relies on hardware to detect and overcome data dependencies. On the other hand, Very Long Instruction Word (VLIW) architecture, [11], uses software solution (compiler) to mark independent operations that can be executed simultaneously. The limits of the parallelism, defined by the length of the VLIW instruction is an issue that has caused development of explicitly parallel instruction computing (EPIC) architecture, [9]. Despite the advantages of EPIC over VLIW, the IA-64 Itanium architecture could not manage to solve all of VLIW's problems. Other alternative to the conventional control flow architecture in providing concurrency in execution of programs is the dataflow architecture, [20]. This architecture is only a concept that has never been implemented in a real hardware.

Each of the discussed processor architectures is described in table 1. The given table shows that pipelining is one of the most used parallelization techniques. This ILP method allows parallel execution of N different instructions in N diverse pipeline stages, so the pipeline length is proportional to the theoretical increase in speed, [2]. Further performance improvements are achieved when multiple pipelines are simultaneously executed on multiple execution units, like in superscalar processors, [20]. Pipelining as an ILP method can be also combined with vector processing, thus allowing data-parallel vector operations to be executed on multiple pipeline execution units, [5]. The achievable parallelism in such processor is dictated by the number of execution units, which also applies to other processor architectures that can be organized to work with multiple execution units, like: Superscalar, VLIW, EPIC and Data Flow.

The performance of computer systems primary depends on the CPU execution time, which is secondary related to the average memory access time, [25]. As a result, computer architects are faced up to the problem of decreasing the CPU execution time, while improving the memory bandwidth and keeping the processor busy all the time. There are several mechanisms that have been developed to target this problem, including: multi-level cache memories, separation of memories for storing programs and data (Harvard architecture), speculative and re-order execution, branch prediction algorithms, etc, [6]. Further improvements are achieved by hardware parallelization i.e. use of multi-core or multi-processor systems that support multithreading, [3]. This approach introduces more intensive work with the memory resources, causing a bottleneck in the system. A possible solution to this problem is integration of processing modules into memory, such as the IRAM-based approach.

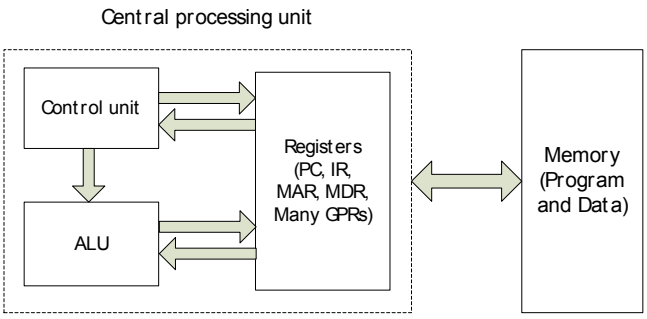
TABLE I
COMPARISON OF DIFFERENT PROCESSOR ARCHITECTURES

	Features				
	Program execution	**Parallel Processing**	**Parallelization Techniques**	**Number of operations in cycle**	**Instruction Format**
CISC Architecture	Control flow	Yes, instruction level	Each instruction executes more operations	Depends on the instruction complexity	Variable length complex operations
RISC Architecture	Control flow	Yes, instruction level	Pipeline that implements several stages	Depends on the pipeline depth (4 or 5 stages)	Fixed length, (usually: 16,32,64)
Superscalar Architecture	Control flow	Yes, instruction level	Several parallel pipelines	Depends on the pipeline depth (8 - 10 stages) and the number of execution units	Fixed length, (usually: 16,32,64)
VLIW Architecture	Control flow	Yes, instruction level	Fixed number of mini-operations in instruction word (implicit parallelism)	Depends on the number of execution units and mini-operations	Fixed length: more mini-operations in one word
EPIC Architecture	Control flow	Yes, instruction level	Variable, but limited number of mini-operations in instruction word (explicit parallelism)	Depends on the number of execution units and mini-operations	Variable length: more mini-operations in one word
Vector Architecture	Control flow	Yes, data level	Same operation is executed on different fixed length vectors	Depends on the number of execution units and vector length	Fixed length, (usually: 16,32,64)
Data-flow Architecture	Data flow	Yes, instruction level	Variable, but limited number of instructions (depends on the availability of input operands)	Depends on the number of execution units and active instructions	Packet instruction format

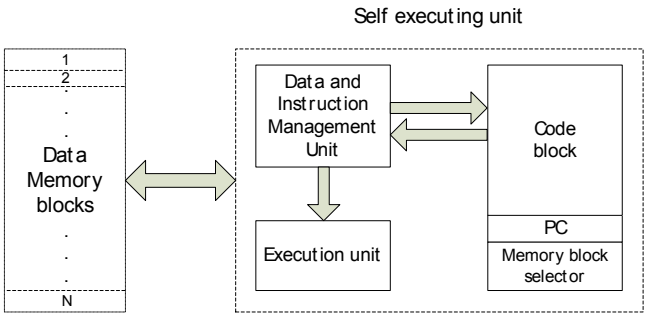
III. DESIGN AND VERIFICATION OF THE NOVEL MEMORY-CENTRIC ARCHITECTURE

The standard Von Neumann based architecture specifies a model of computer system, which consists of memory and processing parts that are strictly separated. In such a system, the memory is used to store both data and instructions, and the central processing unit is purposed to read and decode program code, load/store data to/from cache or/and registers and execute arithmetical-logical operations over the loaded operands. This organization suffers from limited throughput (bottleneck) on the processor-memory interface, caused by the memory and processor speeds discrepancy. Actually, it is well known that the memory system operates at an average access time that is much greater than the processor execution time, [49]. This dissimilarity leads to many wasted processing cycles, since the memory system is not capable to constantly feed the CPU and keep it busy.

Assuming that the processor technology slowly reaches its upper bounds on chip complexity and speed, we suggest that the bottleneck problem should be targeted by introducing novel concepts in computer architectures that will provide closer tie between computing and memory resources, and as well will allow higher utilization of parallel computing. Therefore, our research is directed towards the development of a novel memory-centric computer architecture that organizes the memory system in a completely different way from the long enduring Von Neumann-based systems. In our approach, the memory system is observed as a set of blocks, quite similar to the virtual memory concept. Actually, in the first steps, the memory is separated into many data blocks and a code block that is enhanced with some processing and control capabilities (see Figure 1b). We have named the logic-memory merged block a self-executing unit, since it adds some execution hardware into the code memory, used for storing programs. Having this organization, the memory system has complete control over the program execution (instruction decode, data transfers and ALU operations).



a) Von Neumann computer architecture



b) Memory-centric computer architecture

Fig. 1. Comparison between the classical processor-centric and the novel memory-centric computer architecture

Figure 1 presents the novel memory-centric architecture; consisting of data memory blocks and a self executing unit, connected through a bus interconnect. The data memory blocks are responsible for storing the data used by the programs. The self-executing unit is purposed to immediately fetch and decode the instructions located into the code memory block, and then issue data movement and arithmetic commands over the bus. The operations are performed over the selected data operands from a certain data memory block, which is specified by a specialized register, named memory block selector. Given that the commands are directly executed, the system doesn't need to include intermediate memory resources, such as processor registers or cache memory, so they are not part of the initial memory organization proposal.

The data are stored in the data memory blocks as 32-bit signed integers, represented in second complement notation. This abstraction can be further extended to allow use of 32-bit floating-point numbers with single-precision, according to IEEE-754 format, [50]. The data operands are directly accessed via the interconnection bus, and then served to the execution unit. The operations and the functionalities that the system provides are defined in its instruction set. Basically, the instruction set architecture of the proposed system is RISC-like, and includes arithmetical-logical, shift, branching and auxiliary operations. Each instruction is fixed-length, typically specifying the operation code and the operands, given as direct addresses or immediate values. This is presented in Figure 2.

Operation code	Op1 Addr. offset/ Immediate value	Op2 Addr. offset/ Immediate value	Addr. offset of the Result
6 bit	10 bit	10 bit	10 bit

Fig. 2. Instruction format of the memory-centric ISA

In order to decrease the instruction length, the system utilizes the memory segmentation concept, allowing each data memory block to be represented as a separate memory segment. Accordingly, the self-executing unit is associated with only one memory segment at a given moment. This results with some simplifications in instructions formatting, since the memory operands are specified as address offsets, instead of complete addresses to the data memory block (segment). The identification number of the active memory segment is set by a special instruction that affects the content of the memory block selector hardware. Therefore, it is compiler's responsibility to provide efficient mapping of memory segments, and thus to allow handling of data-intensive operations without difficulties.

The memory-centric system allows several functionalities, which are specified into its instruction set architecture. Generally, the instruction set is divided into four basic groups, each having several simple instructions, with similar structure and function. The arithmetical-logical group includes arithmetical instructions for: addition with overflow detection, subtraction, multiplication, integer division (div) and modulo division (mod), and as well instructions for performing logical operations: logical bit-wise AND, logical bit-wise OR, logical bit-wise XOR and logical bit-wise NOT. The shifting group comprises several instructions for left and right logical shift and rotations. The branching group includes instructions for conditional and unconditional change of the program flow. There are several options for conditional branching, including: equal (=), not equal (!=), less (<), greater (>), less or equal (<=), and greater or equal (>=). The last group is the auxiliary group, consisting of control instructions for program termination and system halt, as well as memory instructions for: constants loading, data transfers between memory blocks, and updating of the memory block selector value.

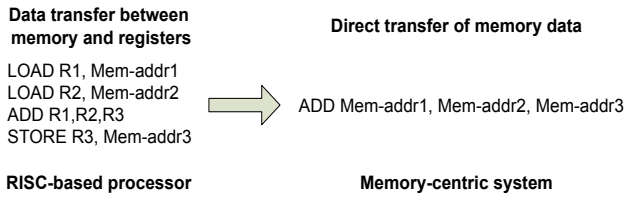


Fig. 3. Program segment written in ISA of the RISC-based load-store processor and the novel memory-centric system

Figure 3 presents a simple program segment that performs addition of two integer numbers. According to that, it is obvious that the memory-centric system reduces the number of program instructions in comparison with the RISC-based processor. This essentially comes from directly addressing the memory data, without using processor registers. Besides that, its instructions length is not longer than 40 bits, meaning that the program size is significantly reduced. This has a very positive impact on lowering the power dissipation of the memory-centric system. Other benefit, which has to be mentioned, is that the self-executing unit immediately fetches and decodes the instruction, thus speeding up the instruction execution. The only drawback of the novel proposed system is that it has to operate on memory working frequency. However, the approach of removing the processor registers brings significant improvement and simplification in the way programs are written, compiled and executed. This is very suitable for applications that perform data-intensive computations, such as: digital signal processing, multimedia, network processing etc.

The proposed memory-centric architecture is implemented in VHDL, by means of Xilinx ISE Design Suite tool. This software environment includes ISim simulator, used for functional analysis of VHDL models. In addition to that, there are several other tools for hardware synthesis and FPGA implementation. The FPGA technology is utterly suitable for research purposes, due to its advantage in terms of speed, cost, flexibility and ease of re-programmability, [51]. Therefore, in this research we make use of the XUPV505-LX110T Virtex5 FPGA board.

Figure 4 presents simulation results for the execution unit, while performing five arithmetic operations over two integer numbers: addition, subtraction, division, modulo division and multiplication. The execution unit, which is part of the self-executing unit, is described in VHDL as a module that receives two 32-bit operands, operation code and chip enable signal as an input and produces 32-bit result value and additional bit for overflow detection as an output. The simulation results shown in Figure 5 verify that the VHDL model is completely functional. In addition to that, the module has been synthesized and implemented in Virtex 5 logic, utilizing 1% of the slice registers, 4% of the slice LUT resources, which is 5% of the occupied FPGA slice resources. The synthesis results demonstrate that the module can achieve maximal working frequency of 952.381MHz.

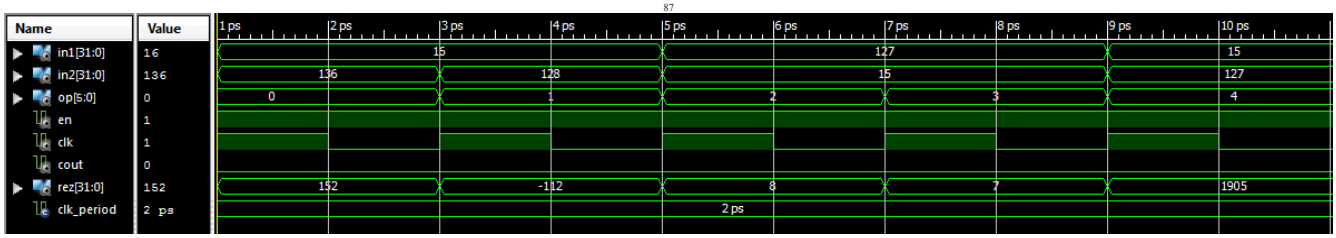


Fig. 4. Simulation results of the arithmetical operations performed by the execution unit: addition, subtraction, integer division, modulo division and multiplication

The self-executing unit is responsible to directly fetch, and decode instructions from the code memory block, and then issue commands that are executed over data operands, selected from a certain data memory block. Each data memory block is described in VHDL as an array of 1Kx4B data words, represented in second complement notation. The 32-bit data words can be also defined in IEEE-754 single-precision format as floating-point numbers, but this would require some additional modifications into the execution unit hardware. The code memory block is also designed in VHDL, as an array of 1K instruction words, wherein the first location is used to preserve the address pointer value of the current instruction, which is actually the program counter value. The code memory block is associated with only one data memory segment at a given moment, which is configured through special block selector hardware. The instruction execution and the data transfers are controlled by the instruction and data management unit, which has complete control over the execution unit, providing direct communication with the associated data memory block.

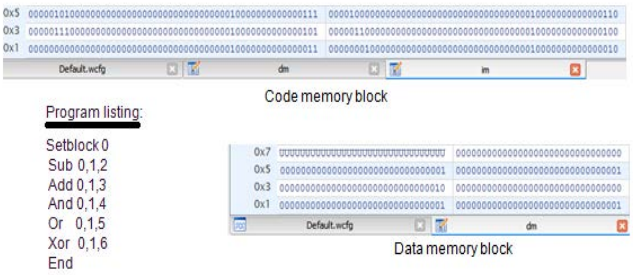


Fig. 5. Simulation results of a simple program execution on the novel memory-centric system

The functionality of the complete system is analyzed through the Xilinx ISE Design Suite environment, which allows monitoring of the memory blocks state in each clock cycle. Therefore, for simulation purposes, we have created a test scenario with a simple program, presented in Figure 5 and an arbitrary data set. The program instructions are first filled into the code memory block, and then executed over the associated data memory block. The simulation results presented in Figure 5 show that each instruction affects the data memory block with some changes, every clock cycle. Therefore, it is verified that the memory-centric system operates properly.

IV. Several Proposal for Organizing the Novel Memory-centric Architecture with Multiple Self-executing Units

The proposed memory-centric architecture that utilizes one self-executing unit and a set of data memory segments can be organized to work with multiple self-executing units. For this purpose, the system is expanded with multiple code memory segments that have execution capabilities (see Fig. 6). The integration of processing hardware into memory is very suitable for FPGA implementation, because the FPGA technology is already designed as a reconfigurable network, [52], of small memory and processing blocks.

The operating system is responsible to load the parallelizable programs and their data into various code and data memory blocks of the system. This means that each code memory block holds and executes a separate thread, process or program segment at any given moment. As a result, the system achieves concurrent execution of all the programs that fit into the code blocks memory.

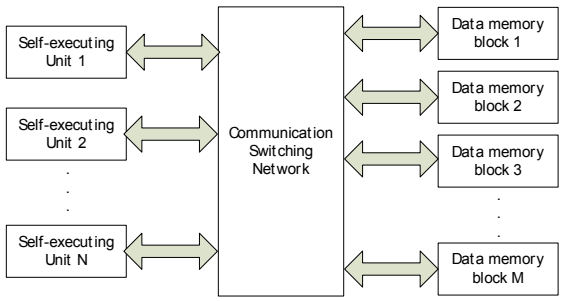


Fig. 6. Organization of the memory-centric system with multiple self-executing units

Figure 6 presents the proposed organization of the memory-centric system with multiple self-executing units and many more data memory blocks (M>>N), connected through a communication switching network. The switching network provides direct communication of N self-executing units with N data memory blocks, thus allowing parallel execution of N independent programs.

There are several interconnection mechanisms, such as AMBA, CoreConnect and Wishbone system-on-chip buses that can be used when implementing communication between multiple master and slave elements. According to the analyzes given in [53] the Wishbone bus is most suitable for the purposes of this research, basically because it is an open-source interconnect that can operate in several different modes, including: shared bus, pipeline and crossbar switch. However, each of the proposed solutions provides N-to-N mappings, which means that each code memory block is associated with only one data memory block. This can be very tricky if a program needs to operate on a larger data set, spread out over several memory segments. In older to resolve this problem, we propose a system that allows reconfigurable use of data memory blocks (see Figure 7).

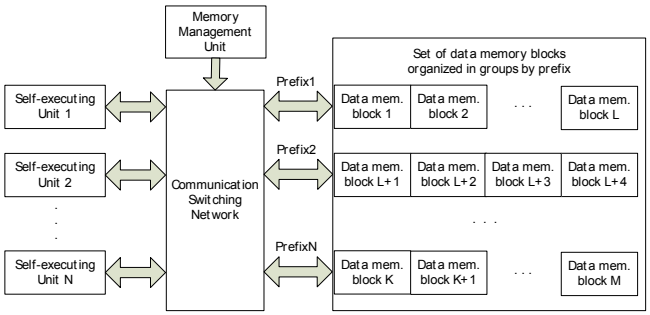


Fig. 7. Proposed model of a memory-centric system, that allows reconfigurable use of data memory blocks

The proposed model, presented in Figure 7 introduces several changes to the way data memory blocks are organized and used by the programs. This system is capable to assign a configurable number of data memory blocks to each program segment. The basic idea behind this approach is the use of prefixes that select a group of multiple data memory blocks. These prefixes are very similar to the IP prefixes used in computer networks for routing or IP assignment purposes. The management of the prefixes is performed by a special hardware (MMU), whose operation is controlled by the operating system. Once the prefixes are set, the communication switching network performs matching of each self-executing unit with a group of data memory blocks, selected by a given prefix. This approach and its applicability are still subject of research, and there are still some issues that need to be resolved.

V. CONCLUSION

The architecture and the organization of the computer systems haven't evolved much in comparison with its beginnings. Therefore, recent computer architectures are still dominantly based on the classical Von Neumann's computer model, which suffers from limited throughput on the processor-memory interface. In order to solve this problem, in this paper we propose a novel memory-centric architecture that adds processing hardware directly to the code memory blocks used for storing programs, thus allowing immediate instruction fetch, decode and execution, easier management of data requests, and direct communication between the data and code memory blocks, without the use of registers. The paper shows that the memory-centric system reduces the number of program's instructions and speeds up the instructions execution, thus providing much better performance characteristics and lower power consumption than traditional computer architectures. The proposed memory-centric system is designed in VHDL, and its correct operation is verified through simulations.

In order to provide explicitly parallel execution, the proposed memory-centric system can be organized to work with multiple logic-memory merged modules (i.e self-executing units) that will operate on separate threads, processes or program segments in parallel. A critical part of such multi-block (unit) organization is the communication switching network, which should provide connectivity of N self-executing units with N data memory blocks. In this paper we observe several solutions, and furthermore propose an approach that allows reconfigurable use of data memory blocks, based on prefixes. The applicability and the performance characteristics of these proposals depend on the system structure and implementation, which is a crucial part of our future research. However, we believe that the proposed memory-centric architecture has the potential to create a new generation of computers with increased portability, reduced size and power consumption, without compromising the performance and efficiency.

REFERENCES

[1] David A. Patterson, John L. Hennessy, "Computer Organization and Design: The hardware/software interface," 5th ed. Elsevier, 2014.

[2] W. Stallings, "Computer organization and architecture: Designing for performance," 8th edition, Prentice Hall, 2009.

[3] John L. Hennessy, David A. Patterson, "Computer Architecture: A Quantitative Approach," 4th ed., Morgan Kaufmann Publishers, 2007.

[4] G. McFarland, "Microprocessor design: a practical guide from design planning to manufacturing," The McGraw-Hill Companies, 2006.

[5] Sivarama P. Dandamudi, "Fundamentals of Computer Organization and Design," New York: Springer, 2002.

[6] D. Jakimovska, et al., "Modern Processor Architectures Overview," Proc. ICEST, Bulgaria, June 2012, pp. 239-242.

[7] Sivarama P. Dandamudi, "Guide to RISC processors: for programmers and engineers," Springer, 2005.

[8] Vojin G. Oklobdzija, "Reduced instruction set computers," Technical Paper, University of California, 1999.

[9] J. Huck, D. Morris, et al., "Introducing the IA-64 Architecture," Proc. IEEE Micro, vol. 20, no. 5. pp. 12-23., Sept/Oct 2000.

[10] C. Kozyrakis, "Scalable vector media-processors for embedded systems," PhD Thesis, University of California, Berkeley, 2002.

[11] T. M. Conte, "Superscalar and VLIW Processors," Handbook, 1996.

[12] N. FitzRoy-Dale, "The VLIW and EPIC processor architectures," Master Thesis, New South Wales University, 2005.

[13] Michael J. Mahon, et al. "Hewlett - Packard Precision Architecture: The Processor," Hewlett-Packard journal, 1986.

[14] A. L. Davis, R. M. Keller, "Data flow program graphs," Proc. IEEE Trans. On Computers, February 1982.

[15] J. Silc, B. Robic and T. Ungerer, "Asynchrony in parallel computing: From dataflow to multithreading," Journal of Parallel and Distributed Computing Practices, 1998.

[16] Ben Lee and A. R. Hurson, "Issues in dataflow computing," Journal of Advances in Computers, 1993.

[17] G. M. Papadopoulos, "Implementation of a general-purpose dataflow multiprocessor," Tech. Report TR-432, MIT Laboratory of Computer Science, Cambridge, August 1988.

[18] R. Buehrer, K. Ekanadham, "Incorporating dataflow ideas into von Neumann processors for parallel execution," Proc. IEEE Trans. On Computers, December 1987.

[19] R. A. Iannucci, "Toward a dataflow/von Neumann hybrid architecture," Proc. 15th ISCA, May 1988.

[20] J. Silc, B. Robic, T. Ungerer, "Processor architecture: From Dataflow to Superscalar and Beyond," Springer, 1999.

[21] Zomaya, A.Y.H, "Parallel and Distributed Computing Handbook," McGraw-Hill, 1996.

[22] M. Smotherman, "Understanding EPIC Architectures and Implementations," Proc. ACM Southeast Conference, 2002.

[23] Ravikanth Ganesan, Kannan Govindarajan, Min-You Wu, "Comparing SIMD and MIMD programming modes," Journal of Parallel Distributed Computing, 1996.

[24] Carlos Carvalho, "The gap between processor and memory speeds," Proc. ICCA 2002, Braga Portugal, 2002.

[25] N. R. Mahapatra, B. Venkatrao, "The processor-memory bottleneck: problems and solutions," ACM Crossroads, 1999.

[26] Christianto C. Liu, Ilya Ganusov, et al., "Bridging the processor-memory performance gap with 3D IC technology," IEEE Design & Test of Computers, vol. 22, no. 6., 2005, pp. 556-564.

[27] Damian Miller, "Reconfigurable systems: a potential solution to the Von Neumann bottleneck," Senior Thesis, Liberty University, 2011.

[28] Christoforos Kozyrakis, David Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks," Proc. 35th International Symposium on Microarchitecture, November 2002.

[29] Harsh Sharangpani, Ken Arora, "Itanium processor microarchitecture," Proc. IEEE Micro, 2000.

[30] C. Cojocaru, "Computational RAM: implementation and bit-parallel architecture," Master Thesis, Carletorn University, Ottawa, 1995.

[31] Peter M. Nyasulu, "System design for a computational-RAM logic-in-memory parallel-processing machine," PhD Thesis, Carletorn University, Ottawa, 1999.

[32] D. Elliott, et al., "Computational RAM: the case for SIMD computing in memory," Proc. ISCA '97, June 1997.

[33] Duncan G. Elliott, Michael Stumm, et al., "Computational RAM: implementing processors in memory," Journal IEEE Design & Test. vol. 16, issue 1, January 1999.

[34] Duncan G. Elliott, W. Martin Snelgrove, Michael Stumm, "Computational RAM: A memory-SIMD hybrid and its application to DSP," Proc. Integrated Circuits conference, 1992.

[35] Peter M. Kogge, Jay B. Brockman, et al., "Processing in memory: chips to petaflops," Technical report, Proc. International Symposium on Computer Architecture, June 1997.

[36] Daescu, Ovidiu, Peter M. Kogge, Danny Chen, "Parallel content-based image analysis on PIM processors," Proc. IEEE Workshop on Content-Based Access to Image and Video Databases, June 1998.

[37] Jeffrey Draper et al., "Implementation of a 256-bit WideWord processor for the data-intensive architecture (DIVA) processing-in-memory (PIM) chip," Proc. 28th European Solid-State Circuit Conference. September 2002.

[38] Maya Gokhale et al., "Processing in memory: the Terasys massively parallel PIM array," IEEE Computer, 1995.

[39] Jeff Draper, Jacqueline Chame, et al., "The architecture of the DIVA processing in memory chip," Proc. 16th international conference on Supercomputing ICS'02, USA, 2002.

[40] Thomas L. Sterling, Huns P. Zimu, "Gilgamesh: a multithreaded processor-in-memory architecture for petaflops computing," Proc. ACM Supercomputing, 2002.

[41] T. Sterling, M. Brodowicz, "The "MIND" scalable PIM architecture," Proc. High Performance Computing Workshop, 2004.

[42] D. Patterson et al., "Intelligent RAM: chips that remember and compute," Proc. Solid-State Circuits Conference, 1997.

[43] David Patterson, Thomas Anderson, et al., "A case for intelligent RAM: IRAM," Proc. IEEE Micro, April 1997.

[44] D. Patterson, et al., "Intelligent RAM (IRAM): the industrial setting, applications, and architectures," Proc. International Conference on Computer Design: VLSI in Computers & Processors, University of California. Berkeley, USA, 1997.

[45] João Paulo Portela Araújo, "Intelligent RAM: a radical solution?," Proc. 3rd Internal Conference on Computer Architecture, 2002.

[46] Brian R. Gaeke, Parry Husbands, et al., "Memory-intensive benchmarks: IRAM vs. cache-based machines," Proc. International Parallel and Distributed Processing Symposium (IPDPS), April. 2002.
[47] Joseph Gebis, Sam Williams, et al., "VIRAM1: a media-oriented vector processor with embedded DRAM," 41st Design Automation Student Design Contenst, San Diego CA, June 2004.
[48] David Martin, "Vector extensions to the MIPS-IV instruction set architecture, the V-IRAM architecture manual," Technical paper, March 2000.
[49] Danijela Efnusheva and Aristotel Tentov, "Integrating processing in RAM memory and its application to high speed FFT computation," Proc. International Conference on Information Society and Technology, Serbia, March 2014.
[50] IEEE, "754-2008 - IEEE Standard for Floating-Point Arithmetic," Technical paper, 2008.
[51] D. Efnusheva, et al., "Efficiency comparison of DFT/IDFT algorithms by evaluating diverse hardware implementations, parallelization prospects and possible improvements," Proc. Second International Conference on Applied Innovations in IT, Germany, March 2014.
[52] Andre De Hon, "Reconfigurable Architectures for General-Purpose Computing," Technical Report, 1996.
[53] Milica Mitić and Mile Stojčev, "A Survey of Three System-on-Chip Buses:AMBA, CoreConnect and Wishbone," Proc. of ICEST, 2006.