Macro-Programmable Reconfigurable Stream Processor for Collaborative Multi-Agent Systems

Valeri Kirischian[1], Vadim Geurkov, Pill Woo Chun and Lev Kirischian

Department of Electrical and Computer Engineering
Ryerson University, Toronto, Ontario, M5B 2K3, Canada

ABSTRACT

Multi-Agent Systems are becoming one of the most efficient solutions for collaborative manufacturing systems. Growing demand for high speed processing of streamed data (e.g. video-streams, digital signal streams, communication streams, etc.) in the recent multi-agent environments requires the adequate cost-efficient stream-processing platforms. Platforms based on the embedded microprocessors often cannot satisfy performance requirements due to limitations associated with the sequential nature of data execution process. During the last decade, development and prototyping of the above embedded platforms has started moving towards utilization of the Field Programmable Gate Array (FPGA) devices. However, the programming of an application to the FPGA based platform became an issue due to relatively complicated hardware design process. The paper presents an approach which allows simplification of the application programming process by utilization of: i) the uniformed FPGA platform with the dynamically reconfigurable architecture, ii) a programming technique based on a temporal partitioning of the application in segments which can be described in terms of macro-operators (function specific virtual components). The paper describes the concept of the approach, presents the analytical investigation and experimental verification of the cost-effectiveness of the proposed platform comparing to the platforms based on sequential micro-processors. It is also shown that the approach can be beneficially utilized in collaborative multi-agent systems for flexible manufacturing or advanced robotic systems.

1 Corresponding author: Tel.: 1- (416) 979-5052; Fax: 1- (416) 979-5280; E-mail: vkirisch@ee.ryerson.ca

1. MOTIVATION AND RELATED WORKS

Nowadays, when time-to-market requirements becoming stronger the agent-based collaborative design & manufacturing becomes one of the most popular solutions for the complex flexible manufacturing systems (FIPA, 2007). On the other hand, the modern manufacturing and advanced robotic systems require embedded high-performance processing systems that can: i) manipulate with large volumes of source data and generate high-speed streams of control signals, ii) communicate with other components of the system via high-bandwidth LAN and iii) communicate via the Internet. All the above functions require high speed real-time processing of streamed data. This is true for the tasks where video / image processing and digital signal processing are involved, which is usual for the most of state-of-the-art robotic and flexible manufacturing systems. During the last decade, these computing systems have been designed around embedded microprocessor platforms coupled with host computer (e.g. PC-platforms coupled with embedded controllers). That approach allowed rapid adaptation of the uniformed computing platform to the custom application by software programming using procedural languages (e.g. C++, Java, etc.) or special frameworks (e.g. JADE, 2004). The hardware prototyping stage in the development process has been excluded or was limited to custom interfacing of the host PC-platform and/or Programmable Logic Controller (PLC) to the actual peripherals. Another advantage of this organization was the ability of easy connection to the Ethernet routers because of embedded Ethernet adaptors in most of the PC platforms. That is why microprocessor based systems became the component basis of the modern agent-based collaborative design and manufacturing systems. According to (Wooldridge and Jennings, 1995), term “agent” is used to denote a hardware / software computing system that implies the following properties: autonomy, social ability, reactivity and pro-activeness. In particular, agents are entities that perform some tasks or set of tasks, appear autonomous, behave intelligently, can communicate and cooperate with other agents (Y. Ding and T.J. Nye, 2004). The architecture of the multi-agent systems mostly is hierarchical (D. Wang at al., 2004), where shared resource agents are allocated in lower level of hierarchy and associated with actual manufacturing resources. These resource agents could be utilized for collaborative manufacturing. The schedule of the jobs for the actual machines on the shop floor depends on the Monitor Resource Agent. This monitor can optimize the waiting time of the machines, the eventual setup times and technical time of working of every machine (R. Iannone at al., 2004). Therefore performance of the entire

system depends on the technical setup and technical time of every machine. In many cases these periods of time depend on respond time of local control system of the machine, accuracy of a data acquisition system, communication bandwidth and, finally, on performance of the computing platform controlling the technological process in the machine. Microprocessor-based platforms often cannot provide the requested performance due to physical limitations of microprocessors with a sequential instruction execution process. Another constraint for increasing their performance comes from the concept of procedural (sequential in nature) software programming. Thus, in case of high speed data-stream processing applications a usual approach is the development of an Application Specific Processor (hardware accelerator) and further implementation in Application Specific Integrated Circuit (ASIC) or in the Field Programmable Gate Array (FPGA) devices. However, the downsides of an ASIC development are a long time-to-market, high cost of the development process, and complete inability for further modification of the ASIC while in production. All the above practically eliminates utilization of ASP implemented in a form of an ASIC from collaborative design and manufacturing process. The implementation in the FPGA device(s) allows decreasing a time-to-market period and minimizing the cost of further modifications. FPGA devices have made possible achieving a processing speed close to that of ASIC-based systems and can be reconfigured remotely (e.g. via the Internet). That is why, in the proposed research the reconfigurable FPGA platform has been selected as the component basis for the “agent” for the shop floor level of multi-agent system. The organization of a multi-level multi-agent system utilizing the Uniform Reconfigurable Processing Modules (URPM) on the shop-floor level was presented in (V. Kirischian at al., 2005). In this work it has been shown that the URPM can behave according to the definition of the “agent” but providing much higher performance than PC platform. It has become possible because of the latest achievements in the FPGA technology, which provides embedded PCs, gigabit interfaces and wide variety of configurable on-chip resources inside an FPGA device. The first prototype of the above platform was developed, manufactured and tested. However, the programming of an application (job task) to the developed platform has not been investigated.

On the other hand, the process of rapid prototyping of a regular FPGA platform is relatively time consuming and expensive. In reality, programming of an application to the FPGA platform is similar to the development of the Application Specific Processor (ASP) for the specific FPGA. This development process

usually is based on the hardware description languages (e.g. VHDL, AHDL, Verilog, etc.) and detailed knowledge of the FPGA hardware organization. These languages are not procedural and therefore allow description of the processes which can run in parallel on the FPGA hardware. Furthermore, all FPGA vendors provide special CAD tools that simplify an ASP design. Vendors also provide libraries of so-called IP-cores to simplify and accelerate the design of an ASP. IP-core is a configuration file for a function specific processor (e.g. FFT, FIR-filter, matrix multiplier, etc.). The above hardware and software means have made ASP development process much easier than it was before. However, some serious constraints still exist. The major problem is associated with development, circuit design, prototyping and verification of the ASP and implementation in the working machine. For the complex ASPs where large (multi-million system gates) FPGA devices have to be utilized, development, prototyping and verification stages are time consuming and expensive. There are several reasons for that:

i) Combining multiple IP-cores into one custom ASP design using existing CAD tools results in a very complex routing inside the FPGA chip and therefore requires a large and expensive FPGA,

ii) Percentage of logic & routing resources non-utilized for the actual processing is relatively high,

iii) Compilation cycle of complex designs is quite long even on recent dual-core workstations,

iv) FPGA platforms for system prototyping are relatively expensive (~$10,000 - $50,000) and

v) ASP design based on the OTS (Off-The-Shelf) FPGA platform requires highly qualified personnel with 2-5 years of experience in certain areas of application. In most of the cases these developers are not specialists in the actual technological process but are hardware designers.

All the above are strong obstacles for utilization of existing FPGA platforms as the design tool. The same reasons are preventing wide utilization of FPGA technology in shop floor control systems for the high-performance technological processes. As it was shown in (V. Kirischian, at al., 2005), in the case of integration design and manufacturing stages on the shop-floor level a special uniformed computing platform is needed. The proposed approach is based on utilization of partially reconfigurable FPGA devices. These FPGAs allows reaching higher cost-effectiveness by utilization of partial configuration by means of spatial and temporal partitioning of the FPGA resources during the task execution. Reconfigurable processor with temporal partitioning of computing resources exploits the same logic and

routing resources of the FPGA for different tasks or segments of the task in different periods of time. This allows minimization of hardware resources per task while keeping high performance of the applications. This is true for most of streamed data processing applications such as: video / image processing (i.e. A. Dasu and S. Panchanathan, 2002), DSP (i.e. M. Kaul at al., 1998) and digital communication (i.e. A. Alsolaim at al., 2000) applications. In this approach, processing within the task is divided to several stages. After execution of each stage temporal results are stored in the memory and new IP-core (configuration data file) reconfigures computing resources in the FPGA for a new segment of the algorithm to be executed. Some of the examples of such systems could have been found in the previous years such as: PipeRench (S. Copen at al., 2000) and KressArray (R. Hartenstein, 2001), however they had much more coarse grain architecture and did not allow fine tuning of the algorithm at hand. Therefore these systems were not used in the manner of partial configuration. In recent years, significant research has been done to investigate ways for optimal temporal partitioning of task algorithms presented in the form of a Data Flow Graph (DFG) and further scheduling of its execution on an FPGA platform (C. Tanougast at al., 2003). All the above works has shown the effectiveness of the temporal partitioning of computing resources for data-stream processing applications. However, the investigation of run-time reconfiguration mechanisms and their effectiveness for temporal partitioning of FPGA resources have not been conducted. Furthermore, the problem of simplification of programming and loading an Application Specific Processor into an FPGA-based platform also has not been investigated.

Thus, the goals of the presented research have been: i) to develop the platform organization which will allow easy programming of the task by the development of application specific data-stream processors and ii) analytical investigation and experimental evaluation of the cost-effectiveness of the developed platform and comparison its cost-performance with a microprocessor-based platforms.

The research was based on the utilization of the concept of temporal partitioning of reconfigurable resources in the partially reconfigurable FPGA devices, because temporal partitioning of a task and FPGA resources would allow simplification of task programming without detailed knowledge of FPGA hardware organization. The reasons are as follows:

i) A segments of a task contain much less operations than an entire task. Therefore it would be much easier to program task segments separately by development respective stream-processing circuits.

ii) Segments or macro-operators could be selected in such a way which will allow utilization of these macro-operators for different tasks of the same class (e.g. Edge detector). Therefore, it is possible to create special library of macro-operators and associated stream-processing circuits.

iii) Each stream-processing circuit associated with the macro-operator being developed by a professional hardware designer, will have architecture optimized for a certain FPGA. Therefore, specific of the hardware implementation is "embedded" into the macro-operator similar to statements of a high-level programming language (e.g. Java or C/C++). In this case, "programming" of a task to the platform may be done by the technologist. The technologist (programmer) may not be familiar with the specifics of the actual hardware platform. He needs to know only how to use macro-operators from the given library.

iv) Programming of a task may look as simple as today's programming of Programmable Logic Controllers in a form of a sequencing graph. This is possible because sequence of task segments (macro-operators) should be known for the technologist. Thus, programming of a task to the platform will look like filling a special template where each time slot is associated with the required macro-operator and its parameterization according to specifics of the technological process. In result, the "program" will look as the schedule of the stream-processing circuits (with determined parameters) to be loaded into the FPGA in the certain time slot for execution. This "program" can be transferred to any machine equipped with the uniform FPGA-based stream-processing platform. However, processing power of the platform could be much higher than for microprocessor-based platforms.

Thus, the rest of the paper is organized as follows: Section 2 gives the background of the concept of temporal partitioning of FPGA resources and architecture of the dynamically reconfigurable macro-processor (RMP). The analytical investigation of performance characteristics of the RMP and comparison with micro-processor-based platforms is presented in Section 3. The implementation of RMP on the basis of Xilinx Virtex-II Pro FPGA is discussed in Section 4. Section 5 provides the results of experimental comparison of RMP performance and platform based on embedded microcontroller with RISC Harvard architecture and PC-based platform. The speedups and cost-effectiveness of Reconfigurable Macro-Processor have been demonstrated. Finally, conclusion summarizes the gained results in the Section 6.

2. THE CONCEPT OF RUN-TIME TEMPORAL PARTITIONING OF FPGA RESOURCES

The temporal partitioning of FPGA resources means a distribution of the same resources (logic, routing and I/O) between tasks or their segments in different periods of time. Nowadays, some FPGAs allow reconfiguration of small parts of internal logic and routing during hundreds of microseconds [8] and [9]. On the other hand, there are lots of areas of applications where data processing cycle should be completed in a range from tens to hundreds of milliseconds [10]. This allows multiple reusing of the same FPGA resources during the data-frame processing cycle. Let us consider an example of a video-processing task where video-frame (1280 x 1024 pixels) has to be captured and processed within 33 ms (30 frames / sec.). Let us also assume that the pixel data are processed on the deep pipeline with throughput of one result per clock cycle. Assuming that all pixels are progressively processed and on-chip clock frequency is 500 MHz, video-frame processing time will be close to: 1280 x 1024 x 2 ns = 2.62 ms. This is about 12.6 times (33 ms/2.62) as fast as the period of capturing of the next video-frame. Therefore, having video-frame capturing time much longer than processing time of this video-frame, temporal partitioning of hardware resources can make the implementation in an FPGA more cost-effective than in an ASIC.

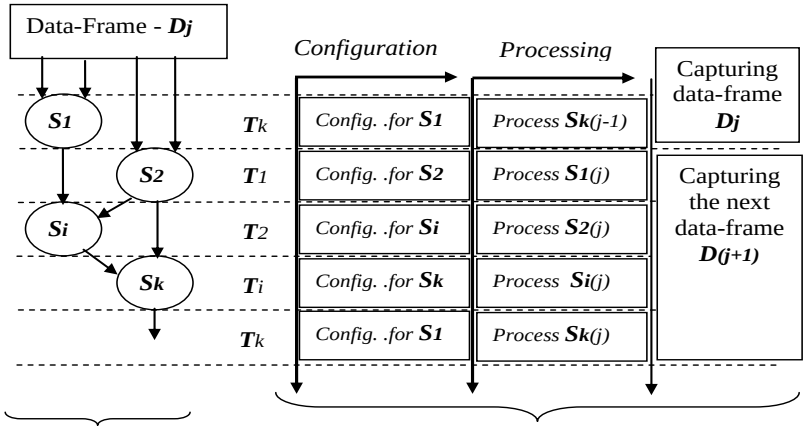

Figure 1: Example of application sequencing graph and associated data-frame processing schedule

Let us consider an example presented in Figure 1. Assuming that a data-frame processing algorithm is partitioned in several macro-operators (segments) – ***Si, i=1,2,…k*** it is possible to present data execution process in the form of a scheduled sequencing graph (Figure 1a). Data-frame processing schedule is shown

in Figure 1b. This schedule consists of data processing periods: ***Ti,, i=1,2,…k*** (according to segment - ***Si***) and the same reconfiguration periods for the next segment ***S(i+1).*** In other words, while one part (data processing slot) of an FPGA executes operations according to segment ***Si,*** another part of FPGA reconfigures its circuits for the next segment of the schedule ***S(i+1), i=1,2,..k.*** The example shows that reconfiguration time overhead can be eliminated by pipelining of the reconfiguration and computation processes while next data-frame is being captured. This allows fitting real-time requirements of an application without reconfiguration time overhead. However, extra hardware associated with run-time partial reconfiguration is needed. If data-frame capturing time is much longer than segment processing time then the number of reconfiguration cycles and thus reuse of the same logic and routing resources in the FPGA may give significant reduction in hardware comparing with static implementation. For the above example within the period of data-frame capturing (33 ms), over than 12 reconfiguration cycles may be performed. Therefore, possible reduction of the logic to be used in data-frame processing data path may be reduced up to 12 times.

Generally speaking, any stream-processing task where data-frame acquisition period is much longer than processing period of the same volume of data can benefit from the above approach.

3. MACRO-PROGRAMMING OF STREAM PROCESSING TASK ON THE FPGA PLATFORM

Any procedural programming of a task algorithm assumes temporal partitioning and sequential scheduling of task operators. In microprocessor-based approach programming is based on procedural description of the algorithm elements in a form of a program – a sequence of instructions. Each of the instructions is a small unit of information which allows control of data acquisition, execution and storage. The set of possible instructions are combined in an instruction set embedded in a microprocessor [Patterson, Hennessy]. Each instruction needs several stages which are not directly associated with data processing: instruction fetch, instruction decode, data fetch, store the result of operation, etc. Even being pipelined in RISC processors exploiting instruction level parallelism number of clock cycles per instruction is more than one clock cycle [Patterson, Hennessy]. However, extra hardware for execution of service stages of an instruction is still needed. For the complex RISC architectures the cost of extra hardware is

much higher than the cost of actual data-processing logic. Another problem comes from the sequential nature of programming which utilizes fine grain control units – instructions. Let us consider the example where elements of four vectors of data ***A, B, C*** and ***D*** have to be processed by the following formula:

$$\boldsymbol{Y = (A_k + B_k) \times (C_k + D_k)},\ k=0,1,2,\ldots 999$$

The pseudo-code should include the following operations (based on traditional instruction set):

1. Set k = 999; 2. Load $\boldsymbol{A}_k$; 3. Load $\boldsymbol{B}_k$; 4. Load $\boldsymbol{C}_k$; 5. Load $\boldsymbol{D}_k$; 6. Add $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$; 7. Add $\boldsymbol{C}_k$ and $\boldsymbol{D}_k$; 8. Get $\boldsymbol{Y}_k$ by multiplication of results from statements # 6 and # 7; 9. Update pointer $k = k - 1$; 10. Compare the pointer value to 0 (zero) and if $k > 0$ then go to statement # 2.

Thus, to process one *k*-set of vector elements it is necessary to execute 10 instructions. Even if the number of clock cycles per instruction is equal to one (ideal case for RISC), processor will need to spend 10 clock cycles * 1000 vector elements = 10,000 clock cycles for the above task segment.

On the other hand it is possible to create stream processing circuit to execute given segment of the task. The block diagram of stream-processor customized for this segment is shown in Figure 2.

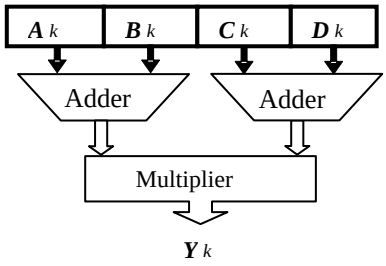

Figure 2: Block diagram of stream processor customized for the function $\boldsymbol{Y = (A_k + B_k) \times (C_k + D_k)}$

Since the data-stream processing is pipelined and no service operations (i.e. instruction fetch, decode, etc.) are needed, the computation time for the vectors with 1000 elements of data ***A, B, C*** and ***D*** will be equal to: $\boldsymbol{T}_{segment} = Latency + Output\ Rate * 1\ c.c. = 2\ c.c. + 1000 * 1\ c.c. = 1002\ c.c.$ In this example we assumed the same – each operation (Add or Multiplication) requires only one clock cycle. Thus, the performance of this function-specific processing circuit is 10 times faster comparing to the microprocessor based platform. Furthermore, if performance needs to be increased in times, then multiple stream-processors may be configured in the FPGA for parallel vector elements execution.

Thus, the approach associated with the creation of function-optimized stream-processing circuits can be very beneficial from the performance point of view.

However, the drawback is the implementation process of the function-specific processing circuits. Usually this implementation cannot be done by a programmer or technologist who is not familiar with specifics of hardware design in FPGA and associated development tools. On the other hand, hardware designers are not specialists in actual technological processes. That is why some transition mechanism has to be found to make programming of stream-processing applications easy for programmers who are not familiar with hardware design in FPGA devices and hardware description languages. Same problem raised before developers of programmable logic controllers (PLC) when programming of technological processes had to be done without detailed knowledge of microprocessors and assembly languages. The solution which has been found was "ladder-logic" programming. In the ladder-logic programming the pseudo-graphic primitives were clear for technologists. Thus, it was easy for them to program the task in the form of a sequenced schedule. This approach allowed exclusion of the detailed knowledge of programming techniques, microprocessor system organizations, etc. It is possible to use similar approach in the case of programming applications associated with the execution of high-speed data-streams. Therefore, the following approach has been proposed and analyzed.

As it was stated in [De Micheli] the data path architectural synthesis requires: i) presentation of an application (task algorithm) in a form of a scheduled sequencing graph, ii) determination of a set of functional resources and a set of constraints. In this consideration we assumed that the application to be programmed can be represented by scheduled sequential graph (e.g. Figure 1 a) where each node (segment) – ***Si, i=1,2,…k*** can be associated with a special function-specific stream-processing unit (functional unit) to be configured in the FPGA. This functional unit is not physically implemented circuit inside the chip but is a sort of so-called soft-processor or better to say – a virtual hardware component (IP-core) stored in the embedded library of Virtual Hardware Components (VHCs). Each VHC consists of three main parts: i) data-path (function-specific stream processor) optimized to the macro-operator (segment of a task), ii) interface to the memory unit(s) to store temporal data and iii) Input/Output interface (optional). It has been assumed that for each class of applications it is possible to create the library of VHCs. This library should be provided by the vendor of certain FPGA platform.

Each VHC is developed using hardware description language (i.e. VHDL, Verilog, etc) and associated CAD tools. Each ***VHC****i* -core is optimized for the respective segment – ***S****i, i=1,2,…k* and associated with certain Macro-Operator (shown in the left part of the Figure 2). Each Macro-Operator has to be written to the time-slot ***Ti,, i=1,2,…k*** t in the "Macro-Program" (shown in the right side of the Figure 3). Therefore the "Macro-Program" is the sequence of macro-operators scheduled according to the sequencing graph (presenting one cycle of data-frame execution). It is assumed that all necessary macro-operators are available for implementation of the application. Obviously, the library of VHCs can be extended by additional custom VHCs, developed for specific applications. However, in this paper we do not consider a synthesis and optimization process of the custom VHC and associated partitioning of the task sequencing graph (or data-flow graph) of an application.

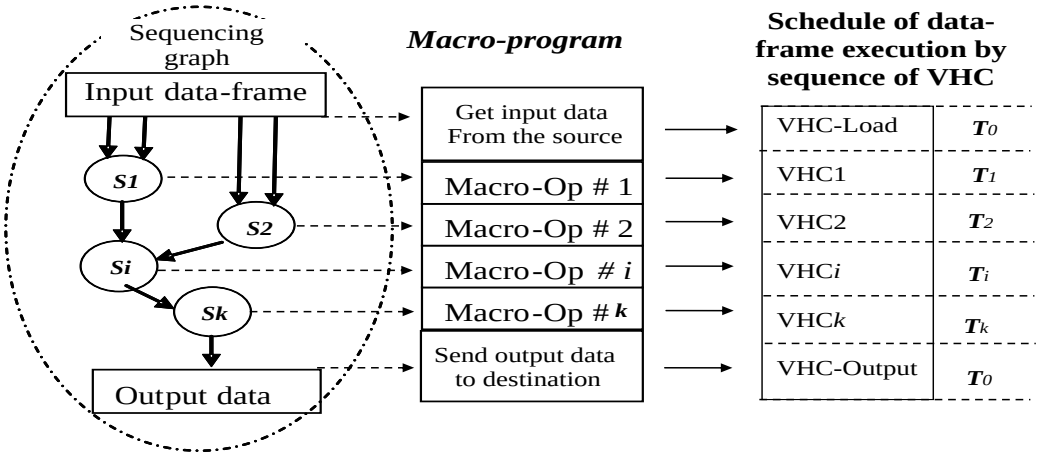

Figure 3: Programming the application using Macro-Operators

Therefore, programming of an application in hardware consists of the following steps: a) selection of the proper ***VHC****i* -core in the library for each ***Si, i=1,2,…k*** and parameterization (e.g. specification of data-frame size, type of data, etc.) and b) scheduling the macro-operators - filling the "template" of the macro-program with specification of each time-slot. As of our experiments (to be discussed in the Section 3.2) the application programming process has been dramatically accelerated. It required minutes for programming different video applications without knowledge of hardware organization of the FPGA based platform.

4. ARCHITECTURE OF RECONFIGURABLE MACRO-PROCESSOR

To implement and investigate the above concept of a dynamically reconfigurable macro-processor (RMP) a special architecture organization of a computing platform has been developed. The initial architecture consist of the following components (Figure 4): i) Data Frame Acquisition Module - DFAM, ii) Two Data Processing Slots - DPS0 and DPS1, iii) Configuration Controller for Data Processing Slots, iv) Resident of Real-Time Hardware Operating System (RTHOS) with Task Schedule File Buffer (TSF - Buffer). The initial architecture is loaded into the FPGA by the Reprogrammable Controller-Loader through the standard configuration port (JTAG). The initial architecture should satisfy the following constraints: i) size of DFAM should reflect the length of a data-frame, ii) space for DPS0 and DPS1 should be selected according to the Macro-Operator's requirements, etc. The initial on-chip architecture is stored in the Soft-Core Memory in a form of configuration bit-stream. In this memory, the library of Virtual Hardware Components (VHCs) is stored too. When the initial architecture is loaded into the FPGA, input data-frames are fed to come to the data-frame acquisition module - DFAM. This module is an application specific IP-core that configures hardware interface to the data-stream source. DFAM is responsible for receiving, pre-processing input data (e.g. decompression, decryption, etc.) and storing data in the required order in the one of the memory banks (Raw Data Bank), which plays the role of the data-frame accumulator.

Reprogrammable Controller-Loader transfers the Task Schedule File (TSF) to the RTHOS Resident. The TSF appears from the Macro-Program as it was discussed in Section 3. The RTHOS Resident stores TSF in the TSF-buffer. This buffer is located in the internal Block-RAM (BRAM) in the FPGA device. TSF then initiates loading DPS0 and DPS1 with the requested VHCs. The number of the VHC is encoded in TSF word. The TSF-word contains physical address of the VHC, which determines a place of the requested VHC in the Soft-Core Memory (Library of Virtual Hardware Components). The proposed approach assumes that the VHC location (initial address of the VHC in the Library) is known for the programmer or compiler. Using the information in the TSF-word, the RTHOS Resident sends a command to the Controller-Loader DPS. This command initiates loading the requested VHC0 (associated with the first Macro-Operator of the task) from the VHC-Cache to the DPS0 slot.

After capturing of the first data-frame in the Raw Data Bank0, RTHOS switches multiplexer connected to the Data Frame Acquisition Module (DFAM) to the Raw Data Bank1. The processing cycle starts from this moment of time. While the DFAM is capturing the next data-frame and storing it in the Raw Data Bank1, VHC0 starts processing data from the Raw Data Bank0 and stores the temporal results in the Temporal Data Bank0. During this time, RTHOS initiates loading of the next VHC1 from the VHC-Cache to the DPS1. When the data processing is complete by VHC0, RTHOS switches the multiplexer of Data Memory Interface from DPS0 to DPS1. That means the switching VHC0 to VHC1. Therefore, data processing and VHC reconfiguration processes are pipelined and overlapped. This process continues till all Macro-Operators (VHCs) listed in the Task Schedule File (TSF) are completed. At the end of the process the last VHC stores output data in one of the available Output Data Banks in the Data Memory. Then execution process returns to the first Macro-Operator in the TSF. At the same time the Data Output Module starts transmission of the output data via the Output Interface.

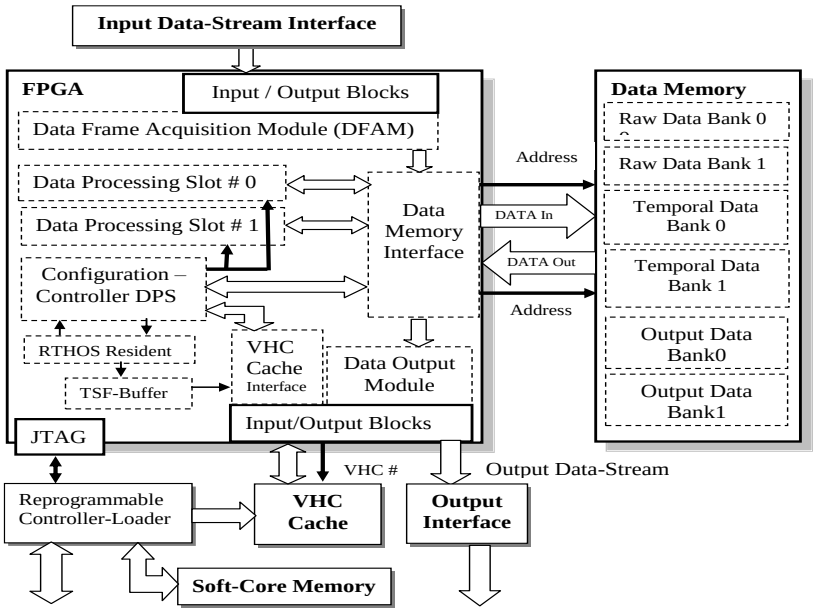

Figure 3: Architecture of FPGA-based computing platform with TPM

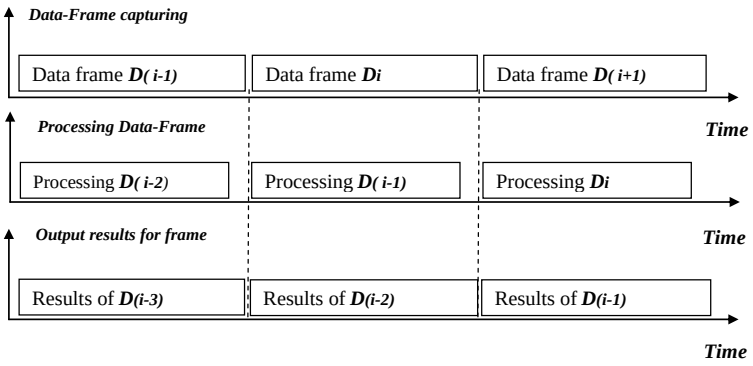

Figure 4: Pipelining of: i) Data-frame capturing, ii) Data-processing using TPM and iii) Outputting results

The above architecture allows implementation of the Temporal Partitioning Mechanism (TPM) of the FPGA resources and thus multi-cycle reuse of the same logic for different VHC-cores. Therefore, TPM makes it possible for three parallel processes to run simultaneously (Figure 4): i) Capturing the data-frame ***D(i+1),*** ii) Processing data-frame ***Di*** and iii) Outputting results of already processed data-frame ***D(i-1).***

Nevertheless, the pipelined organization of a data-frame execution and simultaneous reconfiguration of some part of FPGA resources requires extra hardware (on-chip and external) to provide control communication and temporal data storage functions. Therefore, the effectiveness of the proposed approach depends on the cost of hardware overhead and effectiveness of re-utilization of FPGA resources for stream processing stages. These issues have to be investigated to figure out the perspective of implementation of TPM in a stream-processing computing platform for furtherer industrial application.

5. ANALYSIS OF COST-EFFECTIVENESS OF THE RECONFIGURABLE MACRO-PROCESSOR

5.1. RELATIONS BETWEEN THE COST-PERFORMANCE RATIO AND THE NUMBER OF VHCS

The criterion of the cost-performance for the Reconfigurable Macro-Processor (RMP) has been defined similarly to the price-performance for embedded processors [Patterson and Hennessy] - Cost-Performance Ratio:

CPR = ***Processor cost*** / ***Performance***, where ***Performance*** = 1 / data-frame processing time.

Thus, minimization of the CPR value means maximization of cost-effectiveness of a processor.

To estimate cost-performance characteristics for the Reconfigurable Macro-Processor with TPM the Virtex family of FPGA devices was investigated. We have selected this family because this is the only family of large FPGA devices which allows partial reconfiguration [8]. For experiments we have selected the AP1000 FPGA Development platform (AMIRIX Systems Inc. [15]). The tasks associated with machine vision application (stereo-image processing) have been programmed in the form of a sequence of VHCs (IP-cores) associated with the respective segments –$\boldsymbol{S}_i$ of the above tasks. Number of partitions of the task

algorithm (equal to number of VHCs) has been adjusted to the logic resources available in each of FPGA devices from the Virtex –II family. Then, the above IP-cores have been manually loaded (via JTAG) to the Xilinx Virtex II FPGA – XC2VP100-6FF1704C in the AP1000 platform to measure the real segment processing time. At these experiments reconfiguration time was not taken into account. Processing time has been measured for each of the VHC-cores for stereo-video frames (2 x 640 x 480 pixels and 8-bit per pixel). Processing time of the data-frame - ***TDF*** for each member of the Virtex II family has been calculated accordingly. The peak performance - ***TDFmin*** has been measured for the design, which combined all VHCs in one core and then loaded to the AP1000. Results of the above experiments and calculations are presented in the Table 1. ***TDFmin*** was measured for the device XC2V8000 where non-partitioned design was loaded. Frame rate in this case was equal to 203.4 frames per second (peak performance). ***CPR*** for this case was 55.55 $/ fps. The analysis of these results are shown in Table 1 and graphically represented in Figure 5 and Figure 6. The diagram in Figure 5 shows that the performance of the RMP with TPM depends on the number of VHCs. Obviously, its performance decreases when number of VHCs increases. Same is true for the cost of FPGA device used in the RMP. However, as it can be seen in Figure 5, the cost of an FPGA device drops down faster than the RMP performance when number of VHCs increases. Thus, the value of Cost-Performance Ratio (CPR) drops down while the number of task partitions increases and reaches its minimum when the number of VHCs is equal to 10. In other words, the RMP with TPM becomes most cost effective when the number of task partitions is equal to 10. Thus, the performance of RMP with 10 VHCs is 12.5 times lower then the peak performance which could be reached by the largest FPGA device of this family. However, the cost of the FPGA used in RMP with 10 VHCs is over than 40 times cheaper.

Table 1: Experimental results for Performance and Cost-performance ratio

FPGA device	XC2V500	XC2V1000	XC2V2000	XC2V3000	XC2V4000	XC2V6000
k-number of VHCs (Macro-Operators)	22	10	6	4	3	2
Video-frame processing time	162204 us	61440 us	36865 us	24576 us	18432 us	12288 us
Frame rate (frames per second - fps)	6.165 fps	16.278 fps	27.126 fps	40.690 fps	54.254 fps	81.380 fps

Average unit cost of FPGA (US$)*	$150	$270	$550	$1120	$2210	$4180
CPR (in US$ / fps)	24.33	**16.58**	**20.28**	27.53	40.73	51.36

* Data retrieved from distributor's product price lists (DigiKey [16] and Avnet [17]

On the other hand, if smaller and cheaper FPGA is selected for the RMP (e.g. Xilinx XC2V500), the number of partitions in the task algorithm and associated VHCs has to be increased accordingly (e.g. number of VHCs = 22). In this case cost-effectiveness of the RMP based on this FPGA will drop down (see Figure 5). Thus, for the Xilinx Virtex II family of FPGA devices the best cost-effectiveness will be reached in the case when the number of task algorithm partitions and associated number of VHCs and respective Macro-operators are equal to 10. Therefore, as of Table 1, the Xilinx XC2V1000 FPGA provides the best cost-performance characteristics for the RMP.

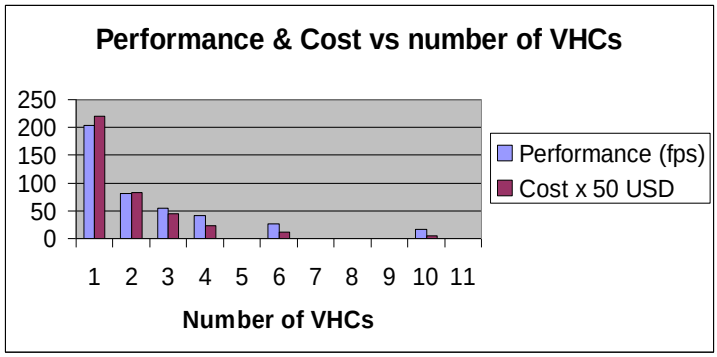

Figure 5: Dependence of the cost of FPGA device and RMP performance on the number of task algorithm segmentation and associated number of VHCs

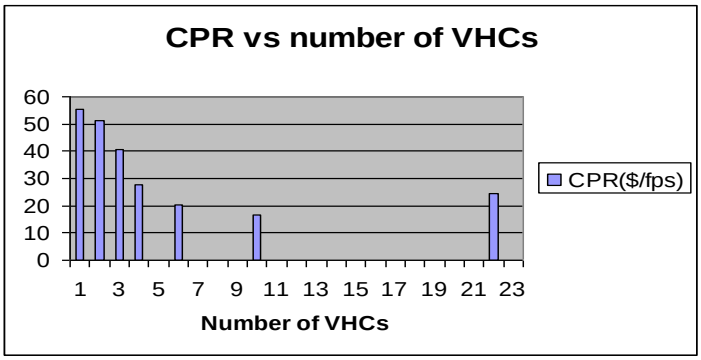

Figure 6: Dependence of the Cost-Performance Ratio on the number of task algorithm segmentation and associated number of VHCs

5.2. CREATION OF THE RMP PROTOTYPE AND EXPERIMENTAL SETUP

Based on the above results a prototype of the Reconfigurable Macro-Processor has been developed and prototyped for further experiments. This prototype has been built on the basis of Xilinx XC2V1000 FPGA and consists of: i) Temporal Data Memory (four SRAM modules combined in two dual-ported memory banks); ii) The Reconfigurable Controller-loader (built on the base of Xilinx XC95144XL CPLD); iii) Two high-bandwidth Input/Output interface ports (built on the base of LVDS drivers). The prototype of the RMP is displayed in Figure 7.

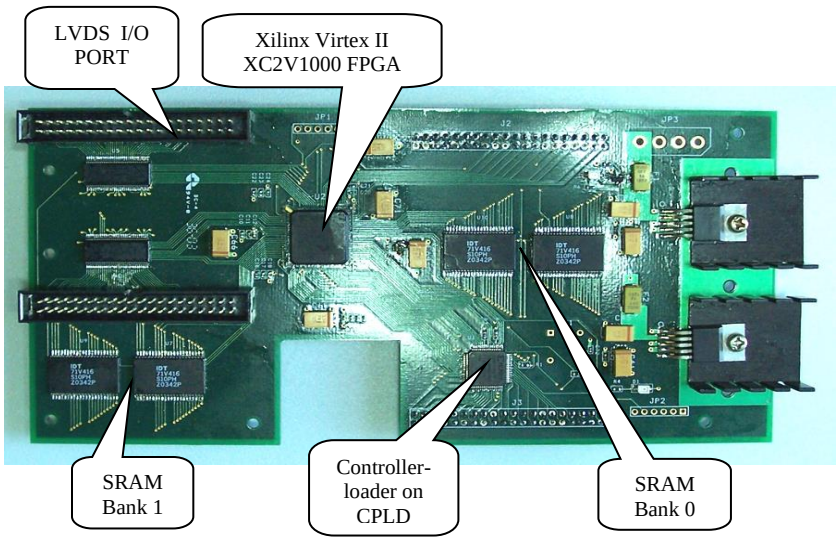

Figure 7: Virtex II FPGA-based RMP with TPM

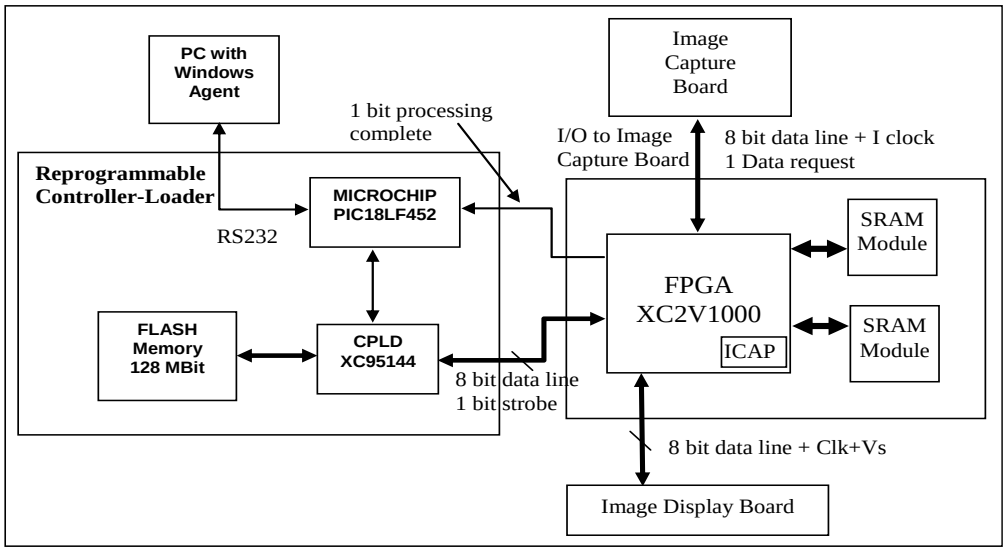

Figure 8: Setup of the FPGA platform with TPM and links to the peripheral boards

According to the architecture of the RMP presented in Section 4 and illustrated in Figure 3 an experimental setup of the RMP with TPM has been created. The diagram of the setup of RMP with TPM is shown in Figure 8. In this setup VHC-core management and communication with the host PC is built on RISC microcontroller Microchip PIC18LF452 when the core transmission to the FPGA is made on the base of Xilinx CPLD XC95144. VHC-cores are allocated in 128 Mbit Flash memory. This Flash memory can store up to 64 VHCs adjusted for data processing slot configuration in the XC2V1000 FPGA. The separation on two sub-systems has been done to estimate TPM cost overhead. Therefore, it has been determined that the total cost of the complete chipset to support TPM does not exceed US $25. It is necessary to mention that the external hardware overhead for the TPM does not change much its value if used for a larger or smaller FPGA device. The difference in cost comes only from the volume of the flash memory. Particularly, in our case, TPM overhead was less than 10% of the XC2V1000 FPGA cost and 0.22% of the XC2V8000 cost. Thus, the TPM overhead does not influence much on CPR of the RMP.

5.3. Comparison of the Cost-Performance Ratio of RMP and RISC -Processors

One of the goals of the project was to find the most cost-effective solution which can fill the gap in performance between high-speed application specific processors and low-cost sequential processors. As it was mentioned in the Section 2, Application Specific Processors (ASPs) can provide the peak performance but require long and expensive development period when sequential processors can provide easy programming and low-cost solutions but have limited performance. Thus, the goal was to estimate the cost-effectiveness of the proposed RMP solution comparing with the existing embedded controllers combined with the host PC. To compare performance and CPR of the RMP and processors used in real-time embedded logic controllers the experimental platform based on a RISC embedded micro-controller has been developed and prototyped. This platform has been designed on the base of embedded microcontroller with RISC Harvard architecture - Microchip PIC18LFXXX family. Microchip PIC controllers are one of the most popular solutions in industrial automation and custom low-cost real-time control devices. Thus, this choice has been made due to wide popularity of the above embedded controllers and their low-cost. To conduct experiments we have developed and manufactured a RISC-controller based test platform. This platform is based on the PIC18F442 Microchip RISC controller in conjunction with Xilinx CPLD

XC95144XL. This CPLD allows similar custom interfacing to different peripherals as in our RMP prototype. As of reference we also conducted experiments with the regular OTS PC platform based on AMD 3000+ processor with 2.1 GHz clock and 1GB SDRAM. Therefore, we have estimated two "extremes" in sequential processing solutions: i) low-cost real-time embedded controller and ii) general-purpose high-performance Pentium-type computer. Definitely, the fairness of such a comparison has been risen. It is usually a problem to insure "fairness" in comparison of processors with very different architecture organization: Pentium-type (Superscalar architecture), exploiting multi-issue, multi-functional instruction processing pipelines; Microchip PIC (RISC Harvard architecture) exploiting simple instruction processing pipeline; and FPGA-based stream processor exploiting long data-processing pipeline. We definitely took in account this kind of "unfairness". However, all three processors were doing to do the same job – processing certain (same) applications. Here the problem appears again, how can one compare processors being programmed in a completely different ways: using high-level procedural language for Pentium (running under non-real time Operating System); using assembly language for RISC-controller (running without any Operating System); and using hardware description language for FPGA platform. In this project the "fairness" has been considered as comparison of the above computing systems "as is". In other words, PC has been programmed and run as it is usually programmed and run in real life. Same was true for the platform based on embedded microcontroller. And the RMP prototype has been programmed using initial library of VHC-cores by manual assembling the test Macro-program.

The test application was associated with run-time image processing of640x480 grayscale images. The algorithm has been partitioned into three components: i) Laplacian enhancement algorithm on a 640x480 grayscale, ii) Sobel edge detection algorithm, and iii) Histogram calculation.

For PC platform the image capturing board and image display board was omitted. Instead, a pre-captured and saved image was used for processing. However for RMP prototype and for PIC-controller platform images have been captured by the video-acquisition module (based on CMOS video-sensors). These video-acquisition modules also have been developed and prototyped in capacity of the projects funded by the Ontario Centers of Excellence (OCE-CITO and OCE-MMO) and our industrial partners.

Image processing algorithms have been implemented in Visual C# .NET development studio in order to get realistic results that can be obtained in execution on a conventional PC. For the purpose of measuring

exact timing, a time stamp was implemented which recorded the beginning of the execution of the algorithm and its completion. The same algorithms have been programmed in CCS "C" for execution on the embedded microcontroller platform. For the RMP each of the above algorithms has been programmed on VHDL as the VHC-core: VHC1- Laplacian enhancement algorithm, VHC2 - Sobel edge detection algorithm and VHC3 - Histogram calculation algorithm. Timing has been measured by HP-logic Analyzer and Xilinx ChipScope. In the Table 2, the acquired results from three platforms are summarized.

Table 2: Results acquired from the algorithms experimentation

Platforms	Reconfigurable Macro-Processor platform	RISC-controller platform	AMD 3000+ PC platform
Processing time at platform frequency	17,642 ms running at 125MHz	19975 ms running at 40MHz	550 ms running at 2100MHz
Scaled processing time normalized to 40Mhz	55.132 ms	19975ms	28875 ms
Speedup comparing to PC	31.175 times	0.028times	1
Normalized speedup comparing to PC	523.743 times	1.445 times	1
Cost of processing unit	~ US$ 270	~US $ 5	~ US $ 150
Cost-Performance Ratio	**4.76 $/ fps**	**100 $ / fps**	**82.50 $/ fps**

As of the Table 2, performance of the RMP based on Xilinx Virtex II FPGA has been measured on frequency 125 MHz. The image (data-frame) processing time (with 3 VHCs in schedule) was about 30 times faster than a PC running at 2100MHz clock. However, in order to bring in the values, the respective results were scaled to the frequency of RISC-controller which was 40MHz. In this case the speedup of the Reconfigurable Macro-Processor was over 523 times faster than a PC, and similarly 362 times faster then RISC-controller platform. Lastly, the CPR for all platforms was calculated based on average processing unit costs. As it can be seen from the bottom row in Table 2, the best CPR still is for Macro-Processor. The cost-performance of this platform is over 17 times lower than for PC platform and 21 times lower than for embedded RISC-controller platform. Thus, it seems that the Reconfigurable Macro-Processor provides: i) dramatic speedup comparing to platforms with existing OTS processors and, ii) better cost-performance characteristics. Comparison of the same parameters taking into account the cost of the entire platforms gives even stronger evidence of RMP cost-effectiveness. As of the Table 3, the RMP being the fastest processing platform provides cost-performance characteristics in 2-3 orders of magnitude better than

existing sequential processors from both sides of their spectrum. It is understood that the above comparison does not consider all details and architectural / programming specifics. However, this initial study based on actual experiments on systems "as is", running the same applications implemented by the same developers may be considered as the rough estimation of the cost-effectiveness of the proposed RMP.

Table 3: Performance and CPR of the entire processing systems

Platforms	Reconfigurable Macro-Processor platform	RISC-controller platform	AMD 3000+ based PC platform (OTS)
Processing time at platform frequency	17,642 ms running at 125MHz	19975 ms running at 40MHz	550 ms running at 2100MHz
Actual Speedup comparing to OTS PC-platform	31.175 times	0.028times	1
Cost of the system	~ US$ 450	~US $ 75	~ US $ 650
System CPR	**7.94 $/ fps**	**1500 $ / fps**	**357. 54 $/ fps**

6. SUMMARY

The paper presents a new approach to the development of a computing platform for multi-agent systems for collaborative manufacturing systems. The proposed Reconfigurable Macro-Processor (RMP) can fill the performance gap in industrial automation which has appeared between high-performance logic based application specific processors (ASP) and general purpose microprocessor based computing platforms. However, growing demand for high speed processing of streamed data (e.g. video-streams, digital signal streams, communication streams, etc.) in the recent multi-agent environments (e.g. high-precision machines and advanced robotics) requires the adequate cost-efficient stream-processing platforms. Thus, the platform based on run-time reconfigurable Field Programmable Gate Array (FPGA) devices has been proposed. The architecture of this platform is not static but dynamically changes during the task execution. This concept allows multiple reusing of the same logic resources for different parts of the task algorithm. Thus, this concept makes possible dramatic reduction of the logic resources and associated cost which increases the cost-efficiency of data-stream processing. To provide necessary run-time control of the dynamic reconfiguration of FPGA special Temporal Partitioning Mechanism (TPM) has been developed, prototyped and tested. The programming of an application to the RMP has been analyzed. The solution which can be beneficial for collaborative manufacturing has been proposed. This solution assumes that the

application (task) can be described in a form of macro-operators and then scheduled according to the task algorithm sequencing graph. Each macro-operator is associated with the macro-operation specific processing core (soft-processor), called a virtual hardware component (VHC). The VHCs may be developed by third party enterprises specialized on certain sectors of hardware design (e.g. DSP-cores, image-processing, etc.). VHCs may be then transferred in any location via the Internet to be included into the application specific libraries of VHC-cores. Then programmer or technologist can combine requested VHCs in Macro-program by filling a special graphic template similar to “ladder-logic” programming. Thus different enterprises having the Internet access to the database of VHCs provided by different vendors can create macro-programs for RMP platforms for certain technological processes. Since a Macro-program is relatively small software unit, it can be sent even via low-bandwidth shop-floor LANs to the actual machines equipped by uniform RMP-platforms. Thus, the proposed approach can be beneficially utilized in collaborative multi-agent systems for flexible manufacturing or advanced robotic systems.

The paper also presents analysis of the cost-efficiency of the RMP with TPM. It has been found that the number of partitions of the task algorithm reflected in the Macro-program directly influences the cost-performance ratio of the RMP platform. The best cost-efficiency could be reached when the number of VHC-cores in the Macro-program is in the range from 6 to 20. However, the processing speed in this case may be in order of magnitude lower than peak data processing rate reached in the ASP with static architecture.

The comparative analysis of performance and cost-performance characteristics for the RMP and modern sequential processors has been conducted. This analysis was based on the experimental results gained on the prototype of RMP, experimental platform based on the embedded RISC-controller and Off-The-Shelf (OTS) PC as the reference platform. All experiments have been done for video and image-processing applications. It has been estimated that RMP platform could provide cost-performance ratio in 2-3 orders of magnitude better than conventional processors. However, the above comparison did not consider all architectural differences of the computing platforms as well as programming specifics. The estimation has been done for platforms “as is”. In other words, each platform has been programmed and run as it is usually programmed and run in real practice.

In conclusion, the Reconfigurable Macro-Processor with Temporal Partitioning Mechanism may be considered as the cost-effective solution for collaborative multi-agent manufacturing systems for the applications which requires high-performance data-stream processing.