Evaluation of CELL IBM Platform Regarding Development of Advanced Real-Time Video Algorithms

Nemanja Lukić[1], Istvan Papp[2], Zoran Marčeta[3], Dušan Ačanski[4], Miodrag Temerinac[5]

Abstract – **This paper presents the evaluation of programmable platforms regarding development and implementation of advanced real-time video algorithms as well as benchmark of such solutions vs. hardware implementations. Evaluation is done using video processing framework and implementation of appropriate video algorithms on CELL IBM platform.**

Keywords – **CELL IBM platform, real-time, advanced video algorithms**

I. INTRODUCTION

Due to increased presence of digital content (multimedia systems, internet streaming) demand for advanced real-time video algorithms has emerged. Main characteristic of real-time video processing is great amount of data that needs to be processed in a given time interval. Evolution and advancement of integrated circuits made possible further rising of hardware processing power. This enabled practical implementation of theoretical advancement in field of video processing algorithms.

Realization of advanced real-time video algorithms is possible in two different ways:

1. Software realization on programmable platforms
2. Hardware realization in programmable hardware components like FPGA (*Field-Programmable Gate Array*)

Advantage of first approach is shorter period between achieving theoretical results and their practical realization. Main flaw of this approach is that it demands hardware platform with great processing power that is able to execute realized software solution in real-time.

Main advantage of second approach (hardware realization) is that it doesn't demand hardware platform with high processing power. But its main flaw is longer period between achieving theoretical results and their practical realization due to restriction introduced by development and hardware implementation of advanced video algorithms. These restrictions reflect in necessary algorithm translation into language suitable for hardware implementation (*Hardware Description Language*). From the beginning of hardware implementation any changes made to algorithms at theoretical level will cause delay, because the hardware interfaces can not be easily changed from that moment. This aspect also represents bottleneck of hardware because these changes are often in phase of development and verification.

Video processing and other multimedia applications belong to market segment where short development and implementation time and low cost of final product present main objectives. Software implementation approach offers short period for developing and implementation. Appearance of hardware architectures like CBEA (*Cell Broadband Engine Architecture*, [1]) offers new approach to development of complex algorithms such as advanced video algorithms. This architecture offers toolkit that lets programmer to implement algorithm in software in fast and efficient way on price attractive platforms. Also, this architecture offers sufficient processing power for real-time implementation. TABLE compares processing power of CBEA based processor with two the most powerful FPGA modules ([5] and [6]). Processing power is expressed in GMACS (***G**iga **M**ultiply and **AC**cumulate*) instructions because these instructions are used often in field of signal processing.

For this evaluation, algorithms for video improvement are chosen as especially desired applications in processing power and data communication. It includes algorithms for de-interlacing, scaling, up-sampling, contrast and brightness adjustment [4].

TABLE I - COMPARATIVE REPRESENTATION OF CBEA AND FPGA BASED ARCHITECTURES PROCESSING POWER

Manufacturer	Product	Frequency	Price	Processing power (GMACS)
IBM	Cell	~3 GHz	~$90	204
Altera	Stratix III	550 MHz	$400	211
Xilinx	Virtex V	550 MHz	$457	352

Goal of this paper is evaluation of CBEA based architecture regarding development and implementation of advanced real-time video algorithms in terms of complexity, efficiency and flexibility.

II. PLATFORM DESCRIPTION

CBEA presents architecture of microprocessors dedicated to distributed data processing. CBEA describes microprocessor that could be either single chip module or

[1] Nemanja Lukić, Faculty of Technical Sciences, Trg D. Obradovića 5, 21000 Novi Sad, Serbia, E-mail: nemanja.lukic@micronasnit.com
[2] Istvan Papp, Faculty of Technical Sciences, Trg D. Obradovića 5, 21000 Novi Sad, Serbia, E-mail: papp@uns.ns.ac.yu
[3] Zoran Marčeta, Faculty of Technical Sciences, Trg D. Obradovića 5, 21000 Novi Sad, Serbia, E-mail: zoran.marceta@micronasnit.com
[4] Dušan Ačanski, MicronasNIT, Fruškogorska 11a, 21000 Novi Sad, Serbia, E-mail: dusan.acanski@micronasnit.com
[5] Miodrag Temerinac, Faculty of Technical Sciences, Trg D. Obradovića 5, 21000 Novi Sad, Serbia, E-mail: miodrag.temerinac@micronasnit.com

multi chip module. Fig. 1 presents architecture of processor based on CBEA.

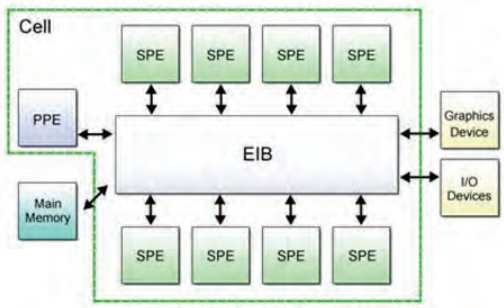

Fig. 1 - Architecture of processor based on CBE (Cell Broadband Engine) architecture

CBEA describes 4 different functional entities of processor:

1. PowerPC Processing Element (PPE, [7])
2. Synergetic Processing Element (SPE)
3. Memory Flow Controller (MFC)
4. Internal Interrupt Controller (IIC)

Component that connects these functional entities inside of processor is EIB (Element Interconnect Bus). Regarding that CBEA is open architecture, number of entities inside processor is variable and could depend on demands and characteristics of system being developed.

III. Algorithms for Video Improvement

Selected algorithms for this paper represent a standard video processing that is present in every TV device available on market. Algorithms are also representative examples for complex and data intensive calculations. Realized algorithms are de-interlacing, scaling, up-sampling, contrast and brightness adjustment.

De-interlacing is the process of converting interlaced video into a non-interlaced (progressive) video. Chrome upsampling is the process of increasing the sample rate of input video chrome components. In this realization it represents conversion from YUV 4:2:2 to YUV 4:4:4 color format. Vertical scaling is process of increasing vertical resolution of input video. In this realization this process is done using polyphase interpolation. Horizontal scaling is process of increasing horizontal resolution of input video. In this realization this process is also done using polyphase interpolation. YUV to RGB conversion is process of converting color space of input video from YUV to RGB color space. Contrast and brightness adjustment is process of adjusting contrast and brightness of input video. It is done in RGB color space. Limiting and formatting is process of preparing video sequence for displaying.

IV. Realization Description

Two modules are distinguished during realization of real-time video processing framework on CBEA platform. First module represents system software that handles input/output of the system (input video data, user parameters and processing results). Second module represents video processing itself.

Real-time video processing framework is divided into two parts regarding to hardware characteristics of CBEA based architecture:

1. Software module executed on PPU
2. Software module executed on SPU's

PPU software module is real-time video processing framework. It is a Linux application that accepts user parameters and according to them creates video processing chain. It also controls input devices (keyboard and PS3 controller). Fig. 2 presents structure of video processing framework (outlined in blue color) and video processing chain (outlined in red color). Video processing framework input is video sequence (compressed or uncompressed. Size of input video sequence is restricted to SD (720x576 or 720x480 pixel per frame) because of the nature of realized algorithms in video processing module. Video processing framework also provides utility for interactive control of framework and realized video algorithms. Results generated by video processing module can be presented on display or written to file, depending on user defined configuration. File writing is primary used during video algorithms verification process.

SPU software module is responsible for realization of selected real-time video algorithms. Fig. 2 also presents list of realized video processing blocks and their SPU assignment. The last SPU processor in processing chain writes his results into video memory of hardware platform.

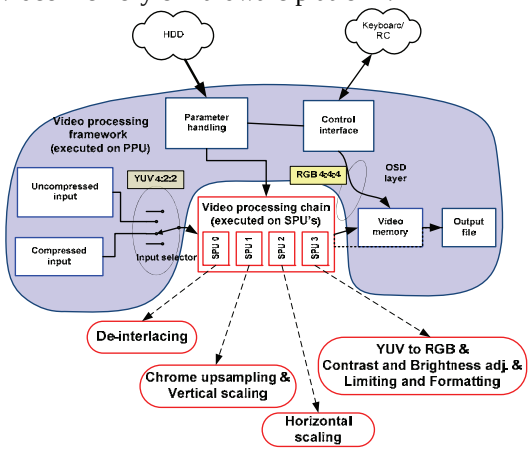

Fig. 2 - Video processing framework and chain structure

PPU processor starts video processing by sending message into inbound mailbox of first SPU element in chain of SPU elements that process video data. Last SPU processor in chain signals end of frame processing to PPU processor by insertion of an adequate message into own outbound mailbox. Data stored into outbound mailbox indicates number of processor cycles that processor executed during processing of single video frame and can be used to calculate exact time (cycle precise) required for processing single video frame.

Fig. 3 presents system data flow. First SPU element in video processing chain (after receiving message from PPU) starts data processing by fetching necessary video data prepared by PPU from main (system) memory. After processing first block of input data, it passes processed block of video data to the next SPU element in the chain. Last SPU

element in the chain passes processed video data to main memory (or directly to video memory) depending on user's system configuration.

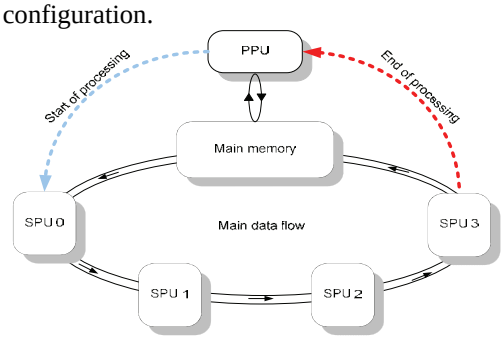

Fig. 3 - System data flow

On every SPU element special mechanism for data acquisition is implemented. This procedure is mandatory due to fact that every SPU element has limited amount of local memory (256 KB of local uniform memory dedicated to data and instructions). Regarding to the fact that whole frame/field even at PAL SD resolution (720x576 pixels per frame or 720x288 pixels per field) can't fit into local memory of one SPU element, synchronization techniques and mechanism of line buffers are developed.

V. Measurement Description and Achieved Results

Performance measurement of video processing blocks is carried out in three different phases of video processing framework implementation on CELL IBM platform. Measurement of achieved performance for each video processing block is performed using IBM system simulator.

First measuring step is measuring of performance achieved when each video processing block is implemented on PPU processor. All video processing blocks are sequentially executed on single processing core (PPU) and represent function calls of realized video algorithm. This phase simulates execution of video algorithms on platforms based on general purpose processors, regarding that PPU core represents one kind of general purpose processor.

After this phase, scalar implementation of video processing blocks is executed on SPU processing core. Scalar implementation represents type of processing where single operand (in this case pixel) is processed using single instruction (SISD, single instruction single data). For every block of video processing chain, single SPU core is reserved. In this phase each SPU core and entire system performances are measured.

After that, the scalar implementation is being vectorised ([3]) and optimized. Performances of vectorised and optimized implementation in final system are measured.

PPU scalar implementation represents realization of scalar functions set that execute appropriate video algorithm. Measuring is realized for each function. Video processing module itself is implemented as sequential calls of realized functions. During execution of video processing chain, each video processing function is called with parameters that are passed on by video framework.

Special system for parameter forwarding is realized, regarding that function parameters are passed on by video framework before each field/frame is processed. TABLE represents achieved processing time of each scalar function of video processing chain executed on PPU processor (expressed in ms).

TABLE II - Measured processing time of PPU scalar implementation of video processing blocks

Block name	Achieved processing time
De-interlacing	1.761
Chrome upsampling	30.844
Vertical scaling	352.522
Horizontal scaling	1880.122
YUV to RGB conversion	176.989
Contrast and brightness adjustment	20.924
Limiting and formatting	10.196

Scalar implementation on SPU processors represents realization of scalar functions set on corresponding SPU processor. This approach's main property is that the blocks of video processing are executed on single processor. This property caused implementation of necessary communication mechanism between SPU processors. Measuring is realized on each SPU processor.

TABLE represents realized time of video processing scalar functions on single SPU processor (expressed in ms). It is obvious that realized time is comparable to results achieved on PPU processor, or even worse. These results are expected and can be explained by fact that synchronization between video processing blocks (demanded by SPU processors implementation) doesn't exist in realization on PPU processor.

TABLE III - Measured processing time of SPU scalar implementation of video processing blocks

Block name	Achieved processing time
De-interlacing	4.6 (SPU 0)
Chrome up sampling	32.171 (SPU 1)
Vertical scaling	381.712 (SPU 1)
Horizontal scaling	2035.8 (SPU 2)
YUV to RGB conversion	228.33 (SPU 3)
Contrast and brightness adjustment	24.698 (SPU 3)
Limiting and formatting	34.43 (SPU 3)

Also, SPU processor is not optimized for scalar functions, regarding that instruction set of SPU processors consists only of vectorised instructions (SIMD, *Single Instruction Multiple Data*, [3]). SPU processor's registers are 128-bits and optimized for SIMD instructions. Compiler used for translation of scalar implementation must generate extra instructions to accomplish this translation. These extra instructions explain worse results of scalar implementation on SPU processor compared to implementation on PPU processor.

Vectorised implementation on SPU processors represents realization of vectorised functions group on corresponding SPU processor. Vectorising of functions implies translating

scalar implementation into implementation which can be realized on vector processor. This process is realized in C programming language using language extensions for SIMD architecture of SPU processor. TABLEIV represents achieved time of video processing vector functions (expressed in ms).

TABLE IV - MEASURED PROCESSING TIME OF SPU VECTORISED IMPLEMENTATION OF VIDEO PROCESSING BLOCKS

Block name	Achieved processing time
De-interlacing	0.25 (SPU 0)
Chrome upsampling	0.788 (SPU 1)
Vertical scaling	1.970 (SPU 1)
Horizontal scaling	10.510 (SPU 2)
YUV to RGB conversion	6.198 (SPU 3)
Contrast and brightness adjustment	2.756 (SPU 3)
Limiting and formatting	1.983 (SPU 3)

The frequency of field/frame processing in single implementation is calculated using following formula:

$$F = 1/M \quad (1)$$

Where F stands for processing frequency (expressed in Hz), and M stands for maximum time needed to process single field/frame of input video sequence (expressed in ms).

Maximum single field/frame processing time in scalar PPU implementation represents summation of times needed to realize all processing blocks in video processing chain because target hardware architecture (CBEA) has single PPU processor, which makes impossible parallelization of video processing chain on PPU. Maximum single field/frame processing time in scalar and vectorised SPU implementation represents time needed to realize the most time demanding (the longest) video processing block in video processing chain, regarding that the blocks are realized simultaneously on different SPU cores. Fig. 4 shows achieved frame rate in different implementations described in previous section.

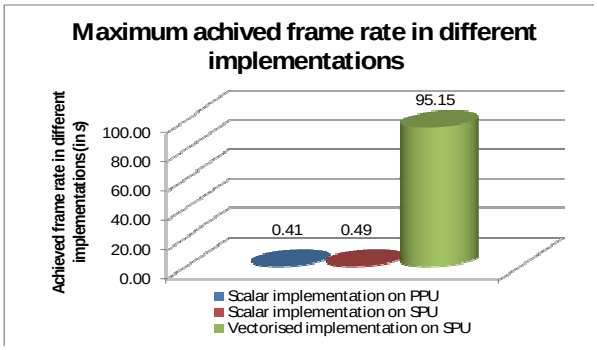

Fig. 4 - Achieved frame rate in different implementations

Scalar implementation on SPU processor achieved better performance comparing to scalar implementation on PPU processor. This can be explained by fact that in scalar SPU implementation video processing blocks are executed simultaneously on different SPU processors. The difference between these implementations is inconsiderable due to fact that the most time demanding video processing block (horizontal scaling) is multiple times more demanding compared to other video processing blocks. This explains why parallelization of processing blocks doesn't achieve greater acceleration of realized system.

Vectorised implementation on CELL processor is 200 times more efficient comparing to implementation of same algorithms on general purpose processor. This result on CELL processor is possible due to fact that it offers parallelization at two levels:

- Parallelism on algorithmic level (different blocks of realized algorithm could be simultaneously executed on different SPU cores)
- Parallelism on data processing level (usage of SIMD instruction set enables processing of up to 16 data using single processor instruction)

VI. CONCLUSION

Advanced video algorithms development using software implementation on systems based on CBEA represents easy and practical solution. There are two commercially available platforms based on this architecture (Sony Playstation 3 and QS-20 Blade Servers). These platforms offer enough processing power to perform even the most complex video processing algorithms in real-time (in this paper SD input sequence in over 95 frames per second).

Fig. 4 presents achieved frame rate in system described in this paper. In this realization 50% of CELL processor resources were used, due to the fact that only 4 SPU cores were involved (CELL processors currently available on market have 8 SPU cores). Realization on general purpose processor (PPU) used 100% of its resources. For real-time performance in this paper, frequency of 60 Hz is needed. Vectorised implementation on CELL processor achieves frequency that is 58% greater than frequency required for real-time performance while implementation on general purpose processors is well below this frequency.

Advanced video algorithm development using systems based on CBEA represent practical solution due to fact that CELL SDK toolkit ([2]) offers mechanisms for fast and efficient development of this system's applications. This significantly shortens time from algorithm changes until real-time demo comparing to hardware implementations.

REFERENCES

[1] Cell Broadband Engine Architecture, IBM, 2007
[2] Software Development Kit 2.1, Cell SDK 2.1, IBM, 2007
[3] C/C++ Language Extensions for CBEA, Cell SDK 2.1, IBM, 2007
[4] "Video Demystified, Fifth Edition: A Handbook for the Digital Engineer", Keith Jack, Elsevier, 2007
[5] Stratix III FPGA High-Performance DSP Features
[6] http://www.altera.com/products/devices/stratix3/overview/architecture/st3-dsp.html
[7] Virtex-5 Multi-Platform FPGA http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/index.htm
[8] PowerPC Architecture Book, Cell SDK 2.1, IBM, 2007