Analysis of Video Filtering on the Cell Processor

Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink
Delft University of Technology
Delft, the Netherlands
Email: {Azevedo, Cor, Benj}@ce.et.tudelft.nl

Mauricio Alvarez
Technical University of Catalonia (UPC)
Barcelona, Spain
Email: alvarez@ac.upc.edu

Alex Ramirez
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
Email: alex.ramirez@bsc.es

Abstract— **In this paper an analysis of bi-dimensional video filtering on the Cell Broadband Engine Processor is presented. To evaluate the processor, a highly adaptive filtering algorithm was chosen: the Deblocking Filter of the H.264 video compression standard. The baseline version is a scalar implementation extracted from the FFMPEG H.264 decoder. The scalar version was vectorized using the SIMD instructions of the Cell Synergistic Processing Element. Results show that approximately one third of the processing time of the SIMD version is used for transposition and data packing and unpacking. Despite the required SIMD overhead and the high adaptability of the kernel, the SIMD version of the kernel is 3.1 times faster than its scalar version.**

I. INTRODUCTION

Video processing coders/decoders (codecs) are increasingly complex to obtain better compression rates and improve the picture quality. The H.264 standard [1] is an example of this trend. H.264 produces a bitstream which is, on average, twice as small as the bitstream produced by the MPEG-2 and MPEG-4 standards, for the same picture quality. However, this improvement is obtained by increasing the computational complexity.

Many processors feature Single Instruction Multiple Data (SIMD) units to accelerate multimedia processing. IBM Cell Broadband Engine processor [2] is an example of a multimedia tailored processor with eight cores that work exclusively in a SIMD fashion. The increasing complexity of multimedia kernels, however, can have a negative impact on the efficiency of SIMD processing, due to the fact that there can be less data-level parallelism with the addition of complex control structures.

This work analysis the suitability of the Synergistic Processing Elements (SPEs) [3] of the Cell Broadband Element for bi-dimensional video filtering. To evaluate the processor a highly adaptive filtering algorithm was chosen: the Deblocking Filter (DF) of the H.264 video compression standard.

Profiling an H.264 decoder shows that the DF consumes about 6% of the processing time of a non-optimized decoder [4]. However, in an optimized version of the decoder with a non-optimized version of the DF, this increases to 49% of the total processing time [5].

This paper is organized as follow. Section II presents a brief overview of the Cell Broadband Engine. The Deblocking Filter is described in Section III and its implementation on the Cell processor is given in Section IV. Section V presents experimental results and comparisons, and Section VI concludes the paper.

II. CELL BROADBAND ENGINE

In this section a brief overview of the Cell Broadband Engine is provided. This overview focuses on the characteristics of the processor relevant to the implementation of the DF. More details about the processor can be found in [2, 3].

The Cell Broadband Engine is a multi-core processor consisting of a dual-threaded PowerPC with AltiVec extension and eight Synergistic Processing Elements (SPEs). Each SPE is an in-order 2-way RISC-like Synergistic Execution Unit (SXU) with a local store (scratch-pad memory), and a DMA unit. The cores are connected by four 16B-wide data rings. Figure 1 depicts the main architectural units of the processor. Figure 1(a) shows the cores and their interconnect, while Figure 1(b) shows the main architectural blocks of the SPE.

The Power Processing Element (PPE) is a simplified version of the PowerPC processor family. It is based on IBM's 64-bit Power Architecture with 128-bit vector media extensions and has a two-level on-chip cache hierarchy. It is fully compliant with the 64-bit Power Architecture specification and can run 32-bit and 64-bit operating systems and applications. The PPE is dual-threaded and has a two-way in-order
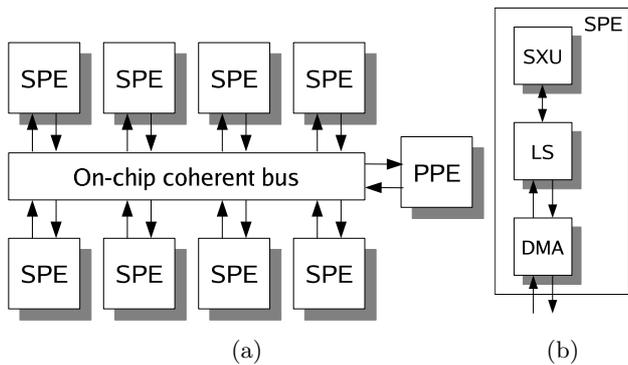
Fig. 1. Cell Broadband Engine block diagram: (a) Cores and interconnect; (b) SPE main architectural blocks

execution unit with a 23-stage pipeline.

The Cell multiprocessor has eight SPEs tailored for multimedia processing. The SPE register file has 128 registers each 128 bits wide. All instructions are 128-bit SIMD instructions with varying element width, i.e. 2×64-bit, 4×32-bit, 8×16-bit, 16×8-bit, and 128×1-bit.

As all instructions are SIMD, the compiler must manage scalar operations. To perform a scalar operation, the SPE compiler first moves both operands to a preferred slot, performs the operation, then shuffles the result with the content of a register, and finally writes it back.

The SPE can only access data and code stored in its Local Store (LS). The LS size is 256KB; code and data must fit on the LS or the programmer must manually describe data and code overlays. The LS is mapped into the memory map of the main processor to allow LS-to-LS communication, but this memory (if cached) is not coherent in the system. Data and instructions are transferred between this local store and system memory by asynchronous coherent DMA commands, executed by the DMA unit included in each SPE. All data communication from/to SPEs is done through explicit DMA commands. A DMA transfer has a maximum size of 16 KB.

Data communication can be performed in parallel with computation. This enables a double buffering strategy to minimize the effects of DMA transfer latency.

III. DEBLOCKING FILTER

The discrete cosine transform (DCT) applied in video and image compression can produce sharp edges between the blocks it operates on. These artifacts are known as blocking, because square areas in the picture are visible. The aim of the Deblocking Filter (DF) is
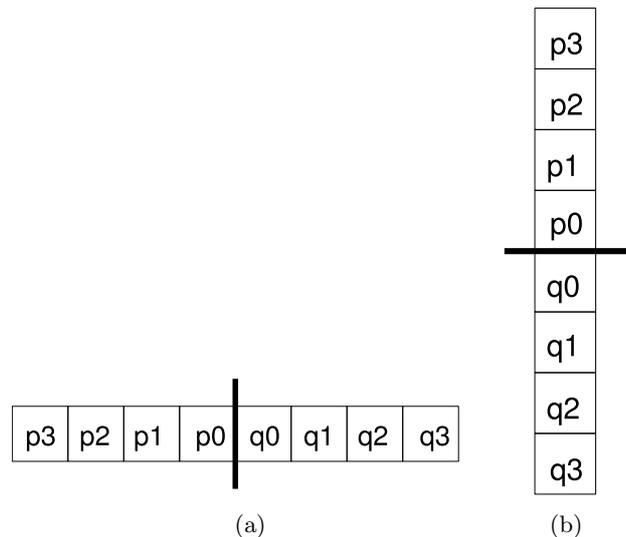

Fig. 2. Line of pixels to be filtered

to improve the appearance of the decoded pictures by smoothing the edges of adjacent blocks. In H.264 this process is defined in the standard and it is part of the coding loop, the filtered frame is used as reference by the next frames. The DF is highly adaptive and has different filter strengths depending on the type of the block being filtered (such as Intra or Inter prediction types) and the types of its adjacent blocks.

The DF process consists of modifying pixels at the four block edges by an adaptive filtering process [6]. The filtering is performed by one of the five different standardized filters, selected by means of boundary strength (BS) calculation. This boundary strength is obtained from the block type and some pixel arithmetic to verify if the existing pixel differences along the border are a natural border or an artifact.

The filters employed in the DF are one-dimensional. The bi-dimensional behavior is obtained by applying the filter on both the vertical and the horizontal edges of all 4x4 luma or chroma blocks. Figure 2 illustrates a line of pixels used in the filtering process, for the left (a) and the top (b) edges. Pixels denoted q are those from the current block, while pixels denoted p are those from neighboring blocks. Depending on the filter strength, the values of pixels $p2$ to $p0$ and $q0$ to $q2$ are modified.

The filter process starts with the vertical left edge of the macroblock (MB) and then each vertical internal edges. After the vertical edges have been filtered, the process is repeated for the horizontal edges. This process is illustrated in Figure 3, where each box represents a 4x4 luma block. The gray area represents the current MB, dark-hatched the p blocks, and the q
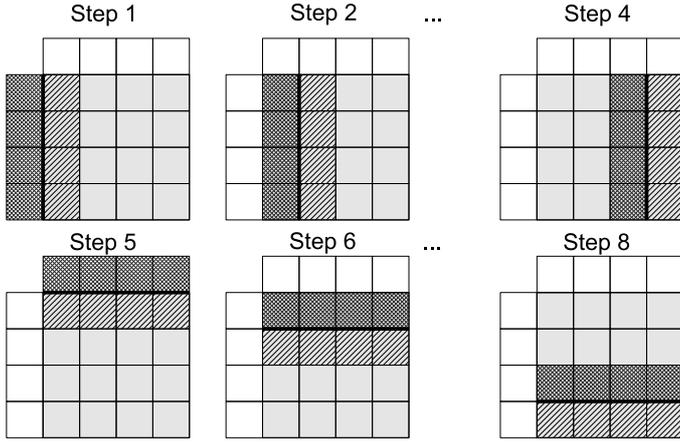
Fig. 3. Filtering of edges

blocks are light-hatched.

The strength of the filter is determined dynamically and depends on the current quantizer, the coding of the neighboring blocks, and the gradient of the image samples across the boundary. There are five strengths which the filter can apply, ranging from 0 (no filtering) to 4 (strongest one). Boundary strength 4 is applied between edges of two Intra Prediction blocks, when one of them is a MB boundary. Boundary strength 3 is applied between edges of an Intra Prediction and Inter Prediction blocks. The other are used to edges between inter prediction blocks.

Filtering is performed over a line of pixels if the conditions $(p0 - q0) < alpha$, $(p1 - p0) < beta$ and $(q1 - q0) < beta$ are met. The thresholds $alpha$ and $beta$ depend on the encoder average quantization parameter over the edge.

There are two filter processes which depend on the BS value. When the BS value is 4, the function below is applied, where pi' is the new value of pixel pi.

```
if( p0 - q0  < ( alpha << 2 ) + 2 )
{
    if( p2 - p0 < beta)
    {
        p0' = p2 + 2*p1 + 2*p0 + 2*q0 + q1 + 4 >> 3;
        p1' = p2 + p1 + p0 + q0 + 2 >> 2;
        p2' = 2*p3 + 3*p2 + p1 + p0 + q0 + 4 >> 3;
    }
    else
        p0' = 2*p1 + p0 + q1 + 2 >> 2;
}
else
    p0' = 2*p1 + p0 + q1 + 2 >> 2;
```

The function below is applied for all other BS values.

```
if( p2 - p0 < beta){
  p1'' = ((p2 + ((p0 + q0 + 1) >> 1)) >> 1) - p1;
  p1' = p1 + clip(p1'', -tc0, tc0);
}

delta' =  (((q0 - p0 ) << 2) + (p1 - q1) + 4) >>3;
delta  =  clip(delta',-tc, tc);
p0'    =  clip( p0 + delta );
```

The above code presents only the equations for p pixels. The equations for q pixels are symmetrical.

The clip functions are defined as follow:

```
clip(x, y, z){
  return x < y ? y : ( x > z ? z : x)
}

clip(x) {return clip(x, 0, 255)}
```

IV. IMPLEMENTATION

In this section the implementation of the DF on the Cell Broadband Engine is detailed. The implementation will be described in a top-down fashion. The description starts with the main loop of the kernel and gradually goes down to the inner parts of the implementation. The focus of this analysis is the computational part of the DF of the FFMPEG H.264/AVC decoder, rather than concentrating on a small computation kernel.

Two versions of the DF were implemented on the Cell processor SPE: a scalar version and a vectorized SPE version. The baseline version is a scalar implementation extracted from the FFMPEG H.264 decoder [7]. The extracted code does not include the parameter calculation of the DF. The analysis focuses on the sample filtering of the code. This scalar version was then vectorized by hand using the SIMD instructions of the SPE.

In these implementations the PPE is used only to read the parameters from the input files and to store them in main memory. After storing the parameters, the SPE threads are spawned. Thereafter, the PPE thread sends a signal to all SPEs to start the computation.

Each SPE thread processes one frame. This approach avoids data movements between SPEs and/or between SPEs and main memory as all data dependencies are between instructions executed on the same SPE. The processing starts by reading the input pointers for the samples and parameters from the main memory.

Each frame is divided into MB lines (MBs from the same row), to use the SPEs ability of performing computation and data communication in parallel. This partition is based on several factors such as the latency, maximum DMA transmission package size, number of DMA transfers, and organization of the data in the memory. The partition size is proportional to the start-up latency but inversely proportional to the number of DMA requests. or each frame, the pixel components (Y, Cb, Cr) are stored in seperate arrays. Partitioning into complete lines of MBs allows to load the pixel samples of one partition with three DMA transfers. Using smaller partitions would require to DMA each line seperately.

For every MB line there are four DMA transfers from memory to the LS. One DMA transfer is necessary for 16 lines of luma samples, two for 8 lines of each set of chroma samples, and another one for the DF parameters of the MB line. After the data is available in the LS, the processing of the MB line is performed and the results are transmitted back to the main memory.

The processing of the MB lines is performed as a software pipeline and uses a double buffering strategy. First, the data for the first MB line is requested, followed by the request of the data for the second MB line. After the data of the first MB line is available in the LS it is filtered. This way the processing of MB line 0 is performed in parallel with the data transmission of MB line 1. The pseudo-code below illustrates the process:

```
Request (MB_Line[0]);

Request (MB_Line[1]);
Wait    (MB_Line[0]);
Process (MB_Line[0]);

FOR x = 2 TO frame_height_in_MB -1
{
    Request (MB_Line[x];
    Wait    (MB_Line[x-1];
    Process (MB_Line[x-1];
    Save    (MB_Line[x-2];
}

Process (MB_Line[x-1];
Save    (MB_Line[x-2];

Save    (MB_Line[x-1];
```

The MB line cannot be immediately transmitted to memory. As can be seen in Figure 3, the processing of the next MB line changes the values of the current bottom edge samples.

The filter process is performed per MB. As described in section III, there are 8 edges, four vertical and four horizontal. First the four vertical edges are filtered and then the four horizontal. The filtering process is divided into the following steps: (1) unpack the 8-bit (8b) samples of the current and left MBs to signed 16b, (2) transpose the current and left MBs, (3) filter the vertical edges, (4) transpose the result, (5) pack back the left MB result to 8b, (6) unpack the last 4 lines of the top MB to 16b, (7) filter the horizontal edges, and finally, (8) pack back the MB result to 8b.

The computational core of the DF is the edge filtering. There are four functions required to implement the edge filtering. Luma and chroma samples require two functions each: one for Intra MB external edges blocks and another one for the other cases. These functions exhibit data-level parallelism and have been optimized with SIMD instructions of the SPE, such that the edge filtering computes 8 pixels simultaneously.

Despite their adaptiveness, the filtering functions have been implemented without branches, except for one to select between the filter processes listed above. To perform the filtering without branches, all equations of the function are computed. All branches are replaced by comparisons that result in a mask. These masks are used to select the positions of the result vectors that will be saved in memory.

V. Experimental Results

In this section the experimental results are presented. First the simulation input and tool are described followed by the analysis of the results. Based on the analysis the conclusions are drawn.

To evaluate the implementations the first eight frames of the Lake Wave video sequence, in the CIF (320 x 240 pixels) resolution, were used as input. The results were obtained using IBM Full-System Simulator for the Cell Broadband Engine processor [8].

Figure 4 depicts the number of cycles required for each frame, for the scalar implementation as well the SIMD version. For the SIMD version, the filtering of a CIF frame takes 2.2 million cycles on average for the video used in this experiment. This corresponds to 0.9 ms per frame in a 2.4 GHz version of the Cell processor or 0.68 ms in a 3.2 GHz processor. For the same input the scalar version consumes 7 million cycles, for each frame, on average. The SIMD implementation has a speedup of 3.1 over the scalar version.

For profiling purposes the SIMD version of the kernel was divided into four parts: Transposition,
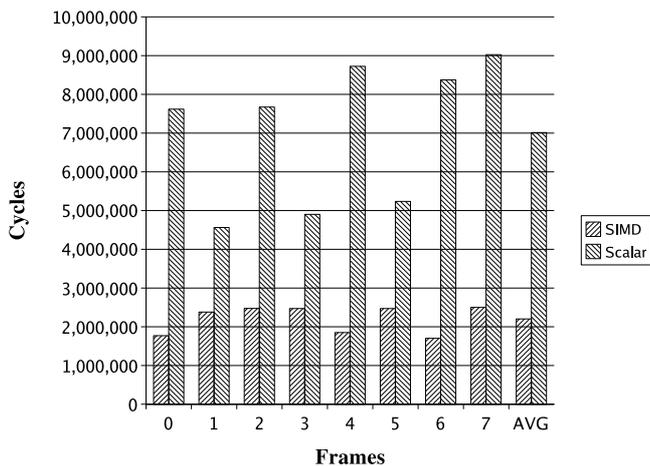
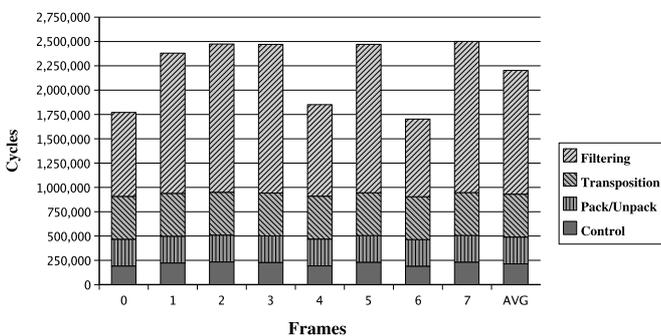Fig. 4. SIMD and Scalar DF performance comparison

Fig. 5. Deblocking Filter performance on the SPE for 8 CIF frames input data

Pack/Unpack, Filtering, and Control. We call Control all parts of the kernel other than Transposition, Pack/Unpack, and Filtering. Figure 5 depicts the number of cycles spent in each part of the kernel. Filtering consumes 47% to 62% of the cycles required to process a frame, with an average of 58%. Approximately one third of the kernel cycles is spent on the Transposition and Pack/Unpack parts, they consume on average 20% and 12%, respectively. The remaining 10% is used by the control structures of the kernel, e.g., data requests and function calls.

The results also show that the double buffering strategy hides the communication latency. The total number of cycles that the cores were stalled waiting for data from memory accounts only for 0.4% of the total running time, on average. As subsequent frames can be overlapped, the data communication influences only in the latency for the first results.

Part of the speedup is due to the mechanism required to process scalar values in the SPE, which introduces overhead. However, this overhead cannot

be measured in the present version of the simulator. Futhermore, due to this feature of the SPE, a comparison with other SIMD implementations on other platforms is not possible. Results of DF implementations on processors with SIMD execution units can be found in [9] and [4].

Experimental results, using hardware counters, show that the speedup of the SIMD implementation on the SPE compared with the scalar version on the PPE is 30%. However, this needs futher investigation, because the used methodology does not present accurate results.

This experiment shows that for video processing the SIMD implementation on the SPE can have good performance improvement over a scalar implementation. The overall speedup is considerable and pays off the extra effort of the SIMDmization of the code. The speedup of 3.1 is a good result, especially considering the high adaptability of the filtering process and the high overhead required by the SIMD processing, such as transposing and data packing and unpacking.

VI. Conclusions

This work presented the H.264 Deblocking Filter as a case study of the video filtering on the Cell Broadband Engine processor. An overview of the Cell processor and of the DF module of the video standard were presented. The DF kernel of the H.264 standard was implemented on the SPEs of the processor.

The kernel was implemented using SIMD instructions and performance was measured using the IBM Cell Simulator. Results show that approximately one third of the processing time of the SIMD version is required for transposition and packing and unpacking of data. Despite this SIMD overhead and the high adaptability of the kernel, the SIMD version of the kernel is 3.1 times faster than its scalar version, both running on the SPEs. This experiment shows that for video processing the SIMD implementation on the SPE can have good performance improvement over the scalar implementation.

As future work we plan to compare the performance of the PPE and SPE versions of the DF. Currently we are replacing the DF of the H.264 decoder of FFMPEG by our modified version and porting it to Cell processor.

References

[1] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[2] J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4, pp. 589–604, 2005.

[3] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic Processing in Cell's Multicore Architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, 2006.

[4] X. Zhou, E. Li, and Y. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," in *SPIE Conf. on Image and Video Communications and Processing, Jan*, 2003, pp. 1–800.

[5] M. Alvarez, E. Salami, A. Ramirez, and M. Valero, "A Performance Characterization of High Definition Digital Video Decoding Using H.264/AVC," in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, 2005, pp. 24–33.

[6] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz, "Adaptive Deblocking Filter," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 614–619, 2003.

[7] "The FFmpeg Libavcoded." [Online]. Available: http://ffmpeg.mplayerhq.hu/

[8] "IBM Full-System Simulator for the Cell Broadband Engine Processor." [Online]. Available: http://www.alphaworks.ibm.com/tech/cellsystemsim

[9] J. Lee, S. Moon, and W. Sung, "H.264 Decoder Optimization Exploiting SIMD Instructions," in *Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on*, vol. 2, 2004.