Optimizing Motion Estimation for Real-time HDTV Encoding

Ronaldo Husemann
University Center UNIVATES
Av. Avelino Tallini, 171
Lajeado, RS, Brazil
Email: husemann@univates.br

Antônio Augusto Fröhlich
Federal University of Santa Catarina – UFSC
Laboratory for Software and Hardware Integration – LISHA
PO Box 476 - 88049-900 - Florianópolis, SC, Brazil
Email: guto@lisha.ufsc.br

Mateus Krepsky Ludwich
Federal University of Santa Catarina – UFSC
Laboratory for Software and Hardware Integration – LISHA
PO Box 476 - 88049-900 - Florianópolis, SC, Brazil
Email: mateus@lisha.ufsc.br

Valter Roesler
Federal University of Rio Grande do Sul – UFRGS
Av. Bento Gonçalves, 9500
Porto Alegre, RS, Brazil
Email: roesler@inf.ufrgs.br

***Abstract*—Over 60% of the total encoding time of raw video into H.264 is spent in block-matching, a stage of Motion Estimation. In this paper we introduce two block-matching optimizations that yield significant performance improvements on Motion Estimation: truncation of the two less significant bits per sample and 4:1 subsampling by macroblock. When applied to the JM Reference Encoder, our strategy showed an average speedup of 2.64 times in total encoding time with a small loss of quality (less than 0.5 dB). We also discuss the implementation of the proposed strategy in hardware, evaluating it in terms of memory consumption, clock cycles, frequency of operation, and chip area. In comparison to available data from related work, this implementation is the fastest one and demands about 72,000 logic gates.**

I. Introduction

Motion Estimation (ME) is a technique used to explore temporal redundancy in multimedia data during compression. Temporal redundancy arises from the fact that neighbor frames in a video stream very often share similar pixel blocks. Therefore the goal of Motion Estimation is to *estimate* the shifting of such similar regions across neighbor frames, thus enabling them to be differentially encoded. Standards like ISO MPEG series and ITU-T H26x are examples of encoders that use ME to improve compression ratios in output video streams [1].

In this article, we explore optimization opportunities in block-based Motion Estimation algorithms. More precisely, we focus on *block-matching* algorithms, since our experiments with the JM H.264 Reference Encoder [2] demonstrated that this stage accounts for over 60% of the total encoding time. Furthermore, a previous implementation of a high definition scalable MPEG-2 hardware encoding engine by one of the authors showed that ME takes up 53% of the chip area [3]. Block-based algorithms were preferred over pixel and region-based ones mostly because we intended our optimizations to be efficiently implemented in hardware and the regular nature of macroblocks (i.e. fixed-size, square regions) is very convenient in that scenario. For the same reason, Sum of Absolute Differences (SAD) was taken as matching criterion in detriment of other traditional criteria such as Sum of Squared Errors (SSE), Mean Squared Error (MSE), and Mean Absolute Difference (MAD).

In order to cope with the computational cost of block-matching, two major alternatives to Full-Search have been considered in this work: sparse search and multi-resolution. Sparse search algorithms avoid scanning the whole search window by deploying heuristics to identify the most probable regions in which similar macroblocks can be found. PMVFast [4], Three-Step search (TSS) [5], and Four-Step Search (4SS) [6] are examples of sparse search algorithms. An alternative proposal, known as Diamond Search (DS), was presented by Zhu [7], and was then improved by Sung to support different H.264 modes [8]. The second approach, multi-resolution, has the goal of simplifying ME through the reduction of image resolution. The computational complexity is reduced because the motion information computed in lower resolution can be used to speed up the motion detection in higher resolutions. The Wavelet Method [9], the Two-Stage Mechanism [10], and the Pyramidal Method [11], [12] fall into this category of block-matching algorithms. From these options, PMVFast has the best performance and shows a Peak Signal to Noise Ratio (PSNR) similar to that of Full-Search and thus was taken as a promising algorithm for the optimizations we had in mind.

Our proposal for an optimized Motion Estimation engine builds on the PMVFast block-matching algorithm and the SAD matching criterion to explore two major strategies: truncation of the two least significant bits of each sample and 4:1 subsampling by macroblock. The engine was implemented in the JM Reference Encoder for quality assessment purposes and subsequently on a FPGA for performance and size evaluation.

The next sections of this article are organized as follow: section II discusses the related work that inspired our proposal and also work on hardware optimization; section III presents the proposed ME optimizations; sections IV and V describe the implementation of the proposed optimization respectively for the JM H.264 Reference Encoder and as dedicated hardware. Section VI is dedicated to evaluate the proposal and its implementation while comparing it to related work. The final considerations are presented in section VII.

II. Related Work

Liu [13] performed a series of experiments with 1:1 subsampling (which is the canonical form and does not imply in losses), 2:1, and 4:1 using two different video sequences: *Tennis*, and *Football*. Liu used 8x8 blocks, and the pixel choice to subsample was based on an algorithm that chooses up to four different groups asymmetrically positioned in the block. Zhong-Li [14] performed a series of experiments with 8-bit samples and adaptive truncation of 0, 1, 2, 3, 4, 5, 6, and 7 bits. His experiments were performed using five video sequences with different animation characteristics: *Miss America*, *Salesman*, *Clarie*, *Carphone*, and *Foreman*. These works were the major inspiration to the approach presented in section III.

Lee [15] describes hardware optimizations to ME based on resolution reduction to minimize the computational cost and chip area. The Successive Elimination Algorithm (SEA) [16] is another approach used to speed up the Sum of Absolute Differences (SAD) calculation proposed by Brunig. SEA avoids unnecessary SAD calculations, comparing the minimum SAD to the absolute difference of the sum of all current macroblock pixels and the sum of all candidate macroblock pixels in the reference image. The total execution time is decreased, because when the sum of a candidate macroblock surpasses the minimum, it can be eliminated. The Global Elimination Algorithm (GEA) is an improvement of SEA in hardware [17]. GEA modifies SEA to achieve a regular flow, with a constant number of clock cycles, and without the necessity of an initial estimation. We compare our hardware implementation to GEA in section V.

III. Proposed Optimizations

We envisioned two major opportunities to optimize matching operations within block-based Motion Estimation algorithms: subsampling and truncation. Both can be applied to the computation of matching criteria to speedup the whole process of motion estimation.

A. Subsampling

Subsampling aims at reducing the time needed to compute a matching criterion, like SAD, by applying the corresponding operation only to a subset of the pixels in a macroblock. The optimization relies on the assumption that neighbor pixels in a macroblock should have similar values and thus computing the matching criterion from a regular subset should yield comparable results. This assumption is confirmed by the experiments presented in section IV. Figure 1 illustrates the process of 4:1 subsampling in a 16x16 macroblock. The dark pixels are the pixels taken into consideration. Instead of comparing 256 pixels (16x16), the algorithm compares only 64 pixels (8x8), speeding up the process.

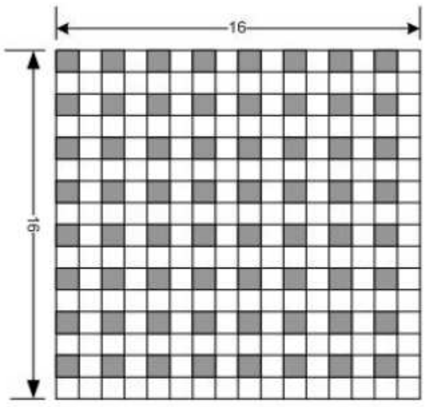

Fig. 1. Illustration of 4:1 subsampling.

B. Truncation

Similarly to subsampling, truncation also aims at reducing the computational cost of block matching by eliminating redundant data. As basis for this optimization we took the assumption that variations in the least significant bits of pixels in a macroblock are more likely to originate from texture nuances than from motion. Edges and shapes usually considered in motion detection are more likely to cause variations in most significant bits of pixels.

Truncating bits of pixels has a more direct impact for Motion Estimation implemented in hardware, since the optimization drives registers and memory words to shrink, reducing the overall chip area and boosting memory access. Nonetheless, our experiments with the JM H.264 Reference Encoder demonstrate that truncation can also improve software-implemented algorithms, because the Full-Search block-matching algorithm in JM features an early termination mechanism. The motion cost in H.264 is calculated using a Lagrangian rate-distortion cost function [18]. The Full-Search block-matching algorithm in JM first computes the rate component of the motion cost for a specific position in the search window. If the rate component value is grater or equal than the previous minimum motion cost (cost of a previous compared block), the distortion component of motion cost (given by the SAD function) is not computed, and the motion cost computation is abbreviated for that block. Truncating bits, numbers that were slightly different from each other, become equal. Because of that, the number of occurrences that the previous minimum cost and the rate component of the current cost are equal increase, and the SAD do not need to be computed. The speedup is then caused by the reduction in the frequency of calls to the SAD function.

IV. Evaluation in the JM H.264 Reference Encoder

We implemented the optimizations described in the previous section in the JM H.264 Reference Encoder [2] to assess their

Sequence	Resolution	Frame rate (fps)	Duration (s)
Foreman	QCIF (176 x 144)	30	10
Mobile	CIF (352 x 288)	30	10
City	4CIF (704 x 576)	60	10
Stockholm	720p (1280 x 720)	60	10

TABLE I
VIDEO SEQUENCES USED IN THE EXPERIMENTS.

impact on the final video quality. Being the reference software implementation for the H.264 standard, JM includes both encoding and decoding functionality. Our experiments were carried out with JM version 14.2 and consisted in encoding a set of reference video sequences with the original encoder and subsequently with the modified one, therefore enabling us to measure variations in the Peak Signal to Noise Ratio (PSNR). Modifications to JM were implemented so that subsampling and truncation could be evaluated both in separation and together, thus rendering more detailed results.

For inter macroblock modes in H.264 (i.e. modes related to the Motion Estimation), the motion cost for chrominance components derives from the motion cost for luminance components [18]. Consequently the PSNR for chrominance components derives from the PSNR for luminance components. For this reason, in this paper we focus on the PSNR variation of the luminance component.

The video sequences chosen for the experiments encompass different resolutions and frame rates: Foreman, Mobile & Calendar, City 4CIF, and Stockholm 720p. These are all raw video sequences in YUV format available at *http://media.xiph.org/video/derf/*. Their key features are summarized in table I.

Subsampling was implemented in JM as proposed in section III. Samples are taken from macroblocs following a regular, symmetrical pattern. For instance, a 4:1 subsample is obtained by having the iteration loop to skip columns and rows as show in Figure 1. Truncation was done in JM by assigning zero to the least significant bits of each sample, since actual register and memory truncation are meaningless to software implementations. Both techniques, subsampling and truncation, were implemented during the SAD computation. The block-matching algorithm used was Full-Search.

Our results show that subsampling and LSB truncation both improve video encoding performance while degrading quality, either deployed separately or in combination. Both parameters, performance and quality, vary proportionally to the extend of the applied optimizations as can be seen in figures 2 to 5. Figure 2 shows the performance improvement for the chosen video sequences as a function of a growing subsampling factor, while Figure 3 shows the corresponding quality loss measured as a decrease in PSNR. Figures 4 and 5 illustrate the facts for truncation.

From the obtained data, we noticed that a combination of 4:1 subsampling and 2 LSB truncation would result in considerable speedups at a relatively small quality loss. Indeed, we looked into the obtained data for a combination that would

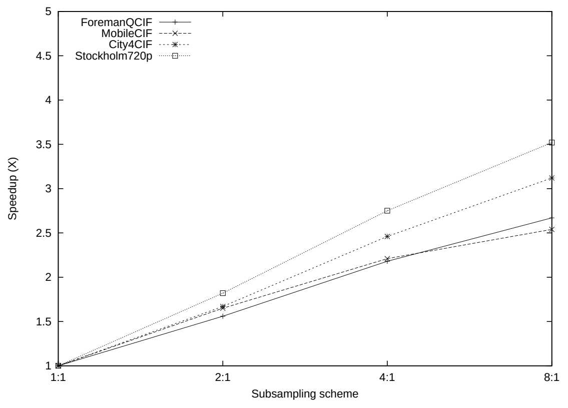

Fig. 2. Impact of subsampling on JM performance.

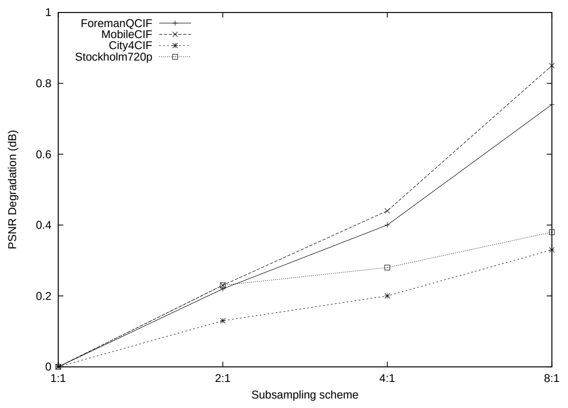

Fig. 3. Impact of subsampling on JM quality.

cause no more than 0.5db decrease in PSNR and yet had good potential for a hardware implementation. Truncation of 2 LSB incurs in an average PSNR reduction of less than 0.1dB, while 4:1 subsampling has a higher toll on quality, about 0.4db. Subsampling, however, is the major speedup source for JM, since $3/4$ of the data in each macroblock is simply ignored[1]. This leads to speedups of up to 3.18 times (for the Stockholm sequence).

The combination of 4:1 subsampling and 2 LSB truncation was further evaluated, yielding the charts in Figures 6 and 7. The average speedup for the combined approach was of 2.64 times, with an average impact on PSNR of less than 0.5db. Notice that higher resolution sequences suffer less quality loss from subsampling. This is because neighbor pixels of high resolution sequences tend to be more similar than those in lower resolution sequences.

V. IMPLEMENTATION IN HARDWARE

Implementing the Motion Estimation optimizations proposed in section III in hardware brings about additional benefits as regards performance. A motion estimation engine implemented in hardware can easily exploit data and process

[1]As explained in section III, speedups from truncation on software ME implementations arise from early termination mechanisms.

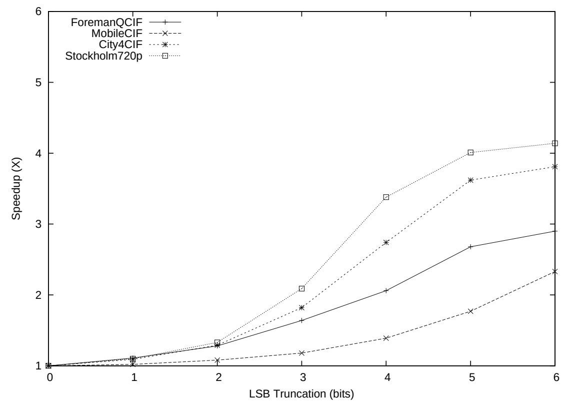

Fig. 4. Impact of truncation on JM performance.

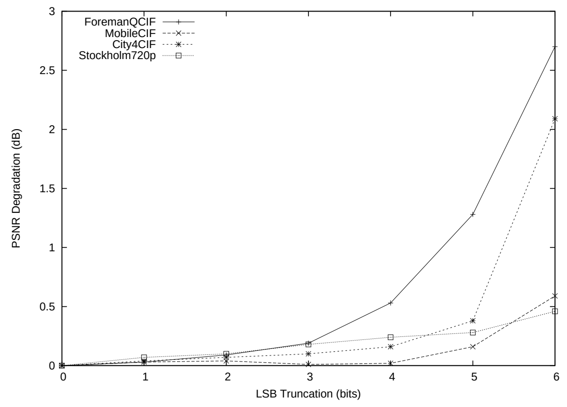

Fig. 5. Impact of truncation on JM quality.

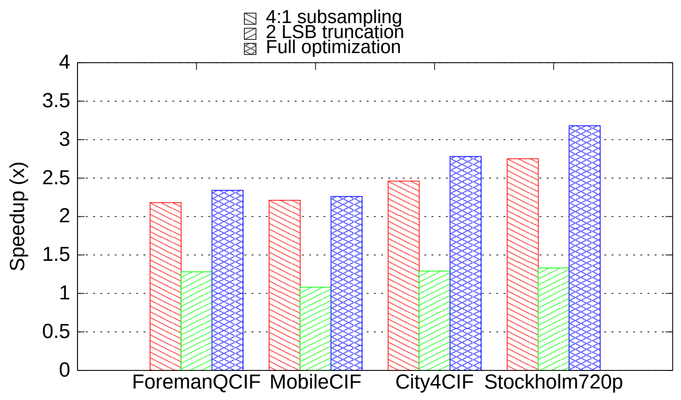

Fig. 6. Optimizations speedup on JM.

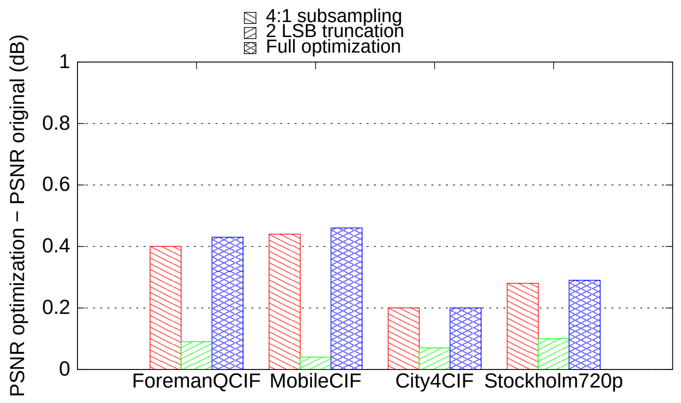

Fig. 7. PSNR degradation due to optimization on JM.

parallelism to achieve considerable speedups relative to sequential implementations. This has already been demonstrated by other groups [19], [15], [17].

As explained earlier in this article, the optimizations proposed here were implemented around the SAD computation within the PMVFast block-matching algorithm. They focus on reducing the comparison time between target and reference image macroblock. Basically, the implemented algorithm gets the target macroblock and walks through the reference image in a diamond topology, searching for the minimum SAD. In the first loop, the algorithm compares the central macroblock of the reference image, the central macroblock shifted one pixel up, shifted one pixel right, one pixel down, and one pixel left.

The first optimization, subsampling, when applied in the 4:1 proportion, immediately yields a 75% reduction in the number of operations needed to calculate the SAD. Additionally, providing alignment of pixel data to the width of the main memory bus further reduces the number of accesses to the memory holding the reference macroblock. With 4:1 subsampling, for example, when each line in the macroblock contains 8 pixels, the PMVFast algorithm needs 10 pixels (8 pixels of central macroblock and two more pixels of the adjacent macroblocks) to shift the main macroblock in each one of four directions (up, down, left and right). The memory cost varies according to the number of bits per pixel. Considering 10 8-bit pixels, the bus width would be 80 bits. The hardware implementation used a 64-bit memory bus, so the ideal sample size is 6 bits, allowing a complete line to be read in only one memory access.

An important issue in our hardware design was to limit contention on the system's main memory. Therefore, after a frame is fetched from system memory and subjected to truncation and subsampling, it is locally stored in two fast access memories, called supplying memories, as illustrated in Figure 8. The first memory, called H-Memory, stores pixel sequences arranged in horizontal lines, which simplifies up and down shifts. The second one, called V-Memory, contains pixel sequences arranged in vertical columns, facilitating lateral pixel manipulation. Each supplying memory holds a copy of the whole reference frame after truncation and subsampling, allowing for parallel access within the iteration loops in the algorithm.

Diamond searching is implemented through a dedicated matrix of registers to increase parallelism. According to the PMVFast algorithm, the register matrix is initially filled with pixel data which represent the central macroblock and their neighboring pixels. After that, each clock cycle triggers a sequence of operations to compare current SAD values with

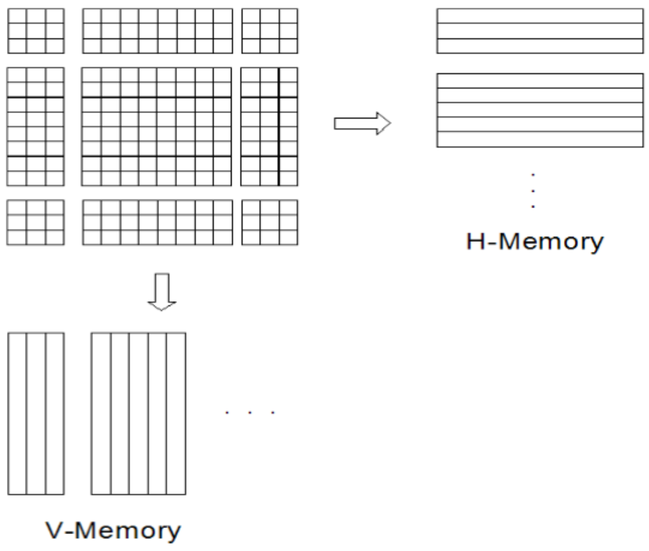

Fig. 8. H and V supplying memories.

those for the central macroblock and then shift the macroblocks by one pixel to all directions (above, below, right, left, and diagonal). The result of the nine SAD computations determine the next step of the algorithm as follow:

- If the central SAD has the lowest result of all, then the value for the motion vector is determined;
- If the top or bottom SAD has the lowest result of all, then the register matrix is taken from the corresponding line of H-Memory (top or bottom memory shift);
- If the right or left SAD has the lowest result of all, then the register matrix is taken from the corresponding column of V-Memory (lateral memory shift);
- If the SAD of some diagonal has the lowest result of all, then the register matrix is taken from the corresponding line of H-Memory and by the corresponding column of V-Memory.

Data from the supplying memory is continually inserted into the register matrix to define the current searching matrix. For example, if a top line is inserted into the searching matrix, pixel data is shifted from top to bottom. The last inferior line is then discarded. This procedure (i.e. data shifting into the register matrix) is executed in only one clock cycle.

VI. Result Analysis

The optimized Motion Estimated engine described above was implemented in VHDL and simulated with Modelsim. The implementation was evaluated according to the following metrics:

- Number of clock cycles spent in the motion estimation module;
- Chip area, measured in gate number;
- Minimum frequency operation;
- Demanded memory.

These metrics are the same used in Huang's work [17] about the Global Elimination Algorithm (GEA) mentioned in section II. Since the encoder for which the Motion Estimation engine was designed is still under development, it was not possible to measure PSNR directly during the evaluation essays. Nevertheless, we can presume the impact of the proposed optimization on PSNR for the hardware module to be the same as that of the JM implementation, since all other hardware optimizations targeted performance improvements that did not incur in algorithm modifications. PMVFast was performed with small diamond search, taking 64 values in each macroblock (obtained through 4:1 sub-sampling). The search area was defined as the whole image. The half-pixel resolution algorithm does not use subsampling.

The performance analysis was carried out with the Foreman sequence at CIF resolution. The original table II was extracted from Huang's work [17]. Huang implemented all referred architectures besides his own (GEA). The architectures signaled with "*" where no fully implemented by Huang (they miss some shift-registers), therefore the number of gates shown in the table for those implementations must be arbitrarily extrapolated for comparison purposes. The evaluation of the engine proposed here is appended to the table.

All Huang's implementations use Full-Search limited to a search window with p = 16 or a total area of 32 pixels. This is too small for a CIF image. Our implementation uses PMVFast without a search window, that is searching the whole image (352 x 288 pixels for the used CIF sequence). So, the proposed approach uses a faster navigation algorithm and a bigger search window.

In the following subsections we compare our approach with the others presented in table II.

1) Memory Utilization: From all presented architectures, the one that demands more memory is the Tree architecture, which needs 4096 bits. The one that demands less memory needs only 8 bits. The implemented approach needs an intermediary memory of 64 bits. The GEA architecture demands 256 bits, which is 4 times more than our proposal.

2) Clock Cycles: Our approach demands 130 clock cycles for type B (bidirectional) motion vectors, segmented according to the following composition:

- 27 cycles to previous reference image analysis;
- 27 cycles to subsequent reference image analysis
- 76 cycles to half-pixel motion estimation calculation.

From all the architectures in table II, the fastest is the one proposed here. This is particularly important for the project in which the Motion Estimation engine will be used, since it targets real-time, 30 fps, HD encoding.

3) Minimum Operating Frequency: As a direct consequence of the small number of clock cycles demanded, the proposed implementation has the smallest operation frequency of all algorithms. With a clock of only 1.55 MHz, it is possible to encode, in real time (30 fps), all motion estimation of the Foreman video sequence in CIF resolution. This puts our algorithm as the best one in this metric.

Considering the results (1.55 MHz to CIF resolution), it is possible to estimate the engine's performance for high definition images. Taking a 1920 x 1080 pixel image, we can

Architecture	Description	Memory (bits)	Clock cycles	Min. freq (MHz)	Gate number (k)
Yang	1-D semi-systolic	24	8,192	97.32	44.7
AB1	1-D systolic	256	24,064	285.88	14.4
AB2	2-D systolic	128	1,504	17.87	98.2
Hsieh*	2-D systolic	8	2,209	26.24	100.1
Tree	Tree based	4,096	1,024	12.17	58.7
Yeo	2-D semi-systolic	24	256	3.04	436.6
Lai	1-D semi-systolic	24	256	3.04	384.8
SA*	2-D systolic	16	1,024	12.17	127.0
SSA*	2-D semi-systolic	16	1,024	12.17	110.6
GEA	GEA based	256	1,635	19.42	23.1
Proposed Architecture	4:1 Subsampling and 2 LSB truncation	64	130	1.55	71.8

TABLE II
COMPARISON OF DIFFERENT MOTION ESTIMATION OPTIMIZED ARCHITECTURES

conclude that it would only be necessary to have a typical board frequency of 30 MHz to accomplish a 30 frame per second motion estimation.

4) Chip Area: Concerning the chip area in hardware, normally there is a direct relation between performance increase and the involved parallelism, i.e., increasing the parallelism and performance increases the chip area. This can also be seen in table II, where the proposed implementation shows the best performance, and also a high gate number (about 72,000) when implemented in a Xilinx Virtex II PRO FPGA. A careful design, however, helped to limit the impact on area, keeping our proposal in pair with others considered. Actually, only Yang, AB1, Tree, and GEA implementations are smaller than our proposal, at the onus of a worse performance.

5) Reasoning about the Comparisons: The processing capacity of 1-D based architectures is not big enough to perform motion estimation in larger search areas, reaching unacceptable frequency requirements. The processing capacity of 2-D based architectures is higher than 1-D, however, the required gate number increases, demanding larger memory use.

GEA architecture uses less gates, however it demands more memory, has a higher minimal operating frequency, and needs more clock cycles per operation.

VII. CONCLUSIONS

In this article, we introduce two complementary optimizations for block-based Motion Estimation in multimedia encoding: truncation of the two less significant bits per sample and 4:1 subsampling by macroblock. The proposed optimizations were developed around the PMVFast block-matching algorithm and the SAD matching criterion, yielding a block-matching engine that was implemented in the JM Reference Encoder and also on a FPGA.

The evaluation of the proposed optimizations in the JM H.264 Reference Encoder showed a quality loss of less than 0.5dB for all considered sequences, and an average speedup of 2.64. The main purpose of this evaluation was to assess ME quality, since good performance gains were foreseeable for the parallel hardware implementation of the proposed optimizations. The positive results encouraged us to advanced with the hardware implementation.

The hardware implementation was compared to different strategies and it was clear by the presented results that the proposed method is the most efficient one, requiring only 130 clock cycles at 1.55 MHz, and requiring only 64 bits of memory. The chip area, with about 72,000 gates, is also very competitive.

Therefore, we conclude that there is a real advantage in using the approach proposed in this paper, mainly for encoders targeted at high-resolution, real-time streaming.

REFERENCES

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, 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, August 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.815165

[2] "H.264/avc jm reference software," 2009. [Online]. Available: http://iphome.hhi.de/suehring/tml/

[3] R. Husemann, A. de Souza Jr, T. Tome, and V. Roesler, "Análise da implementação de algoritmos de codificação e decodificação de vídeo mpeg-2 hd escalável em hardware," in *SEMISH 2006*, 2006, pp. 1–10.

[4] A. M. Tourapis, O. C. Au, and M. L. Liou, "Predictive motion vector field adaptive search technique (pmvfast) -enhancing block based motion estimation," 2001.

[5] T. K. et al, "Motion-compensated interframe coding for video conferencing," 1981.

[6] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," vol. 6, no. 3, pp. 313–317, June 1996.

[7] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 9, no. 2, pp. 287–290, Feb 2000.

[8] S. Kim, J. Han, and J. Kim, "Efficient motion estimation algorithm for mpeg-4 to h.264 transcoder," 2005, pp. III: 656–659.

[9] Y. Zhang and S. Zafar, "Motion-compensated wavelet transform coding for color video compression," vol. 2, no. 3, pp. 285–296, September 1992.

[10] S. K. et al., "Interframe coding using two-stage variable block-size multiresolution motion estimation and wavelet decomposition," 1998.

[11] K. Nam, J. Kim, R. Park, and Y. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," vol. 5, no. 4, pp. 344–351, August 1995.

[12] M. S. Zan, J. Ahmad, "Pyramidal motion estimation techniques exploiting intra-level motion correlation," 2003.

[13] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," vol. 3, no. 2, pp. 148–157, April 1993.

[14] Z.-L. He, C.-Y. Tsui, K.-K. Chan, and M. L. Liou, "Low-power vlsi design for motion estimation using adaptive pixel truncation," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 10, no. 5, pp. 669–678, 2000.

[15] S. Lee, J. Kim, and S. Chae, "New motion estimation algorithm using adaptively quantized low bit-resolution image and its vlsi architecture for mpeg2 video encoding," vol. 8, no. 6, p. 734, October 1998.

[16] M. Brünig and W. Niehsen, “Fast full-search block matching,” 2001.
[17] Y. Huang, S. Chien, B. Hsieh, and L. Chen, “Global elimination algorithm and architecture design for fast block matching motion estimation,” vol. 14, no. 6, pp. 898–907, June 2004.
[18] T. D. H. Du, “Macroblock mode decision for h.264,” in *MIR ’05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*. New York, NY, USA: ACM, 2005, pp. 167–172.
[19] I. Nousias, “Optimized hardware/software mpeg video player,” 2002.
[20] ISO/IEC, *Optimized Reference Software for Coding of Audio-visual Objects*, 2002.
[21] ——, *International Standard ISO/IEC 13818-5 (Software Simulation)*, 1997.
[22] Z. He and M. Liou, “Reducing hardware complexity of motion estimation algorithms using truncated pixels,” in *Circuits and Systems, 1997. ISCAS ’97., Proceedings of 1997 IEEE International Symposium on*, vol. 4, Jun 1997, pp. 2809–2812 vol.4.