ANALYSIS AND HARDWARE ARCHITECTURE FOR GLOBAL MOTION ESTIMATION IN MPEG-4 ADVANCED SIMPLE PROFILE

Shao-Yi Chien, Ching-Yeh Chen, Wei-Min Chao, Yu-Wen Huang, and Liang-Gee Chen

DSP/IC Design Lab
Graduate Institute of Electronics Engineering and Department of Electrical Engineering
National Taiwan University
1, Sec. 4, Roosevelt Rd., Taipei 106, Taiwan
{shoayi, cychen, hydra, yuwen, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

Global motion estimation (GME) and compensation is one of the key modules in MPEG-4 Advanced Simple Profile (ASP). However, there are no hardware architectures for GME since existing algorithms are not suitable for hardware implementation. In this paper, GME in MPEG-4 ASP is analyzed, and a hardware-oriented GME algorithm is proposed according to the analysis results, which is a combination of a feature points based algorithm and a novel iterative sprite point matching algorithm. The associated hardware architecture is also proposed. Simulation results show that the performance of the proposed algorithm is the same as that of MPEG-4 Verification Model. The implementation results show the proposed hardware architecture can achieve the requirements of MPEG-4 ASP@L3 with 66K gates and 31Kb internal memory at 100 MHz working frequency.

1. INTRODUCTION

Advanced Simple Profile (ASP) is defined in Streaming Video Profile in Amendent 4 of MPEG-4 [1]. Compared with MPEG-4 Simple Profile (SP), ASP is defined for devices with high processing power, and more than 50% of bits can be saved [2]. The most important compression tools contained in MPEG-4 ASP are B-frame, quater-pel motion estimation/compensation (QME/QMC), and global motion estimation/compensation (GME/GMC). The state-of-art ASP hardware encoders do not implement GME since the computation complexity is too high, and there are no algorithms suitable for hardware implementation.

Many global motion estimation algorithms have been proposed. They can be classified into three types: frame matching, differential technique, and feature points based algorithms [3]. Frame matching algorithm matches the whole frame with the candidate motion parameters to find the global motion vector [4]. The differential method employs Taylor series to expand a criterion function into polynomial equations, such as frame difference of the current frame and the global motion compensated frame [5]. Both of these two kinds of algorithms have large computation complexity. The feature points based algorithms [6] first find the motion vectors of the feature points. The global motion vector can be then derived with regression. The computation complexity of this kind of algorithms is much smaller than those of the former two kinds, but the motion vectors are not as accurate as theirs.

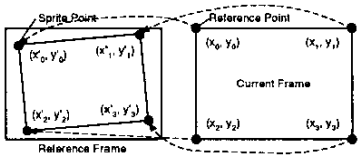

Figure 1: In MPEG-4, GMC parameters are transmitted with the positions of the sprite points.

Instead of transmission the global motion parameters directly, in MPEG-4, the global motion information is transmitted with the trajectories of the reference points in order to handle the quantization error during compression. In ASP, the reference points are set as the four corners of a frame, as shown in Fig. 1. One, two, three, or four reference points is used for translation, isotropic, affine, or perspective model. For every frame, the corresponding positions of the reference points in the reference frame, which are denoted as sprite points in Fig. 1, are transmitted. In MPEG-4, only half pixel precision is allowed for the sprite points. We think the positions of sprite points should be considered for GME in MPEG-4 ASP to avoid redundant computation for precision higher than half pixel, and GME can be applied directly in the sprite point domain rather than in motion parameter domain.

In this paper, we first analyze GME in MPEG-4 ASP to find the optimal way to implement GME. Based on the analysis, in Section 3 and 4, a new algorithm suitable for hardware implementation and the associated hardware architecture are proposed. The simulation and implementation results are shown in Section 5. Finally, Section 6 concludes this paper.

2. ANALYSIS OF GLOBAL MOTION ESTIMATION AND COMPENSATION IN MPEG-4

In this section, GME is analyzed in MPEG-4 ASP to show its coding efficiency and find the parameters for implementation.

Several standard sequences are tested. From the rate-PSNR curve, we find that GME sometimes has almost no coding gain as the results of [2]; however, GME has benefits for some sequences when global motion of zooming and rotation are involved. One of the typical case of zooming are shown in Fig. 2, where the first 30 frames of *Table Tennis* in CIF format are used as test sequence.

0-7803-7761-3/03/$17.00 ©2003 IEEE

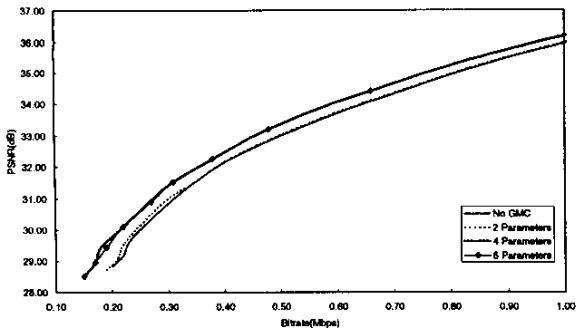

Figure 2: Rate-PSNR curve of sequence *Table Tennis*, where curves of no GMC and GMC with two, four, and six parameters motion models are shown.

When ASP Level 3 encoder is considered, it requires about 750 Kbps to achieve 35 dB in PSNR with GMC and requires 810 Kbps without GMC. It is shown that GME can reduce 7.4% in bitstream size. Besides, the global motion information can also support other high level video signal processing applications, such as global motion descriptor in MPEG-7, scene change detection, and video segmentation. Therefore, GME should be included in the implementation of the MPEG-4 ASP encoder. On the other hand, the runtime profile shows that GME takes 34.8% computation time. The large amount of runtime comes from the complex gradient-descent GME algorithm, many iterations, and differential operations with respect to all the motion parameters in each iteration. Hence hardware implementation of GME is necessary.

We also examine the performance of ASP with two, four, and six parameters motion models. The results show that the performance of four and six parameters are similar, as shown in Fig. 2. The reason may be that in most cases, only panning, tilting, rotation, and zooming camera motions occur between two successive frames, and four parameters are suitable for most applications. On the other hand, although in some cases, the performance of two parameters are similar to those of four and six parameters, in most cases, the performance is similar to ASP without GMC, as shown in Fig. 2. We think it is because that the DPCM compression scheme of local motion vectors can achieve the same performance of GMC when only translation camera motion occurs. For that reason, the number of motion parameters should be more than or equal to four, and four motion parameters (isotropic model) are selected in this paper.

3. PROPOSED HARDWARE-ORIENTED ALGORITHM

The three types of GME algorithms, including differential technique, feature points based algorithms, and frame matching, all have some drawbacks for hardware implementation. The differential algorithm (DA) contains many floating point operations in the differential value calculation, inverse matrix operations are involved in the linear regression, and the amount of memory access is large for the iterations. In feature points based algorithms (FPBA), the computation load is much lower; however, floating points operations and inverse matrix operations are also contained in the linear regression, a large sorting element is required for feature points selection, and the motion parameters are not as accurate as those of the other algorithms. In frame matching algorithms (FMA), the computation are more regular, and the floating point calculation in GMC can be avoided with a predefined GMC precision [5]. Although FMA is more suitable for hardware implementation, the drawback is that the search step size is hard to decide, and the computation and the memory access amount is enormous.

We proposed a new GME algorithm named as sprite point matching algorithm (SPMA). Since MPEG-4 transmits global motion parameters with the locations of sprite points, as describe in Section 1, the optimal sprite points are searched directly instead of finding the optimal motion parameters. For isotropic model, two sprite points, (x'_0, y'_0) and (x'_1, y'_1), are used, and the operations can be presented as the following equations.

$$SAD_{\{t'_0,t'_1\}} = \sum_{t \in F} |CF(t) - RF(W(t, t'_0, t'_1))|, \quad (1)$$

$$\{t'_0, t'_1\} = \arg \min_{\{t'_0,t'_1\}} SAD_{\{t'_0,t'_1\}}, \quad (2)$$

$$t'_0 \in \text{search range}, t'_1 \in \text{search range},$$

where CF denotes current frame, RF denotes reference frame, $t = (x, y)$, $t'_0 = (x'_0, y'_0)$, $t'_1 = (x'_1, y'_1)$, F is a set of all the pixels in the current frame, and $W(t, t'_0, t'_1)$ denotes the warped location of location t according to the sprite points.

Since the computation complexity of SPMA is too large for hardware implementation because a GMC operation needs to be applied in whole frame for each candidate sprite point, a fast algorithm is proposed, which is named as iterative sprite point matching algorithm (ISPMA). In order to reduce the search range, FPBA with two motion parameters is applied first to give predicted sprite points.

In the two-parameters FPBA, the feature points are first found in current frame. The feature points are the points with higher Hessian values [6]:

$$\frac{d^2CF(x,y)}{dx^2}\frac{d^2CF(x,y)}{dy^2} - \left(\frac{d^2CF(x,y)}{dxdy}\right)^2. \quad (3)$$

In local motion estimation, a cross matching algorithm in an L-neighborhood (typically L=4) around the feature point is applied [6]. The feature points with higher sum of absolute difference values are dropped. After that, the two global motion parameters are just the average motion vectors, and the predicted locations of sprite points can be derived.

The operations of ISPMA is similar as (2) except for the way to search sprite points. The search scheme of ISPMA can be shown in Fig. 3. Figure 3(a) shows the predicted sprite points A and B. We fix point A and move point B in a predefined search range and find the optimal location B', as shown in Fig. 3(b). Then we fix point B' and find the optimal location A', as shown in Fig. 3(c). Next, the procedure is applied again with A' and B' as new predicted sprite points, and the search range can be reduced. After we find the optimal locations of sprite points in the integer pixel precision, a full search algorithm is employed to find the optimal locations in half pixel precision, as shown in Fig. 4, where $9 \times 9 = 81$ candidate locations are searched. In many experiments, we find that 52 feature points and the search range of (3×3)-(3×3)-(1×1) can give accurate locations of sprite points.

4. HARDWARE ARCHITECTURE

Based on the proposed algorithm, a hardware architecture for GME is proposed in this section. The block diagram of the proposed ar-

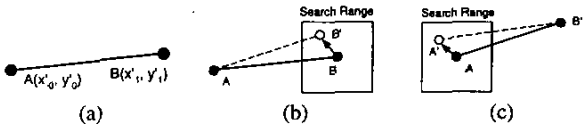

Figure 3: Iterative sprite point matching algorithm.

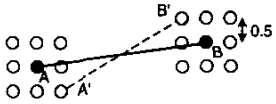

Figure 4: Full search sprite points in the range [-0.5, 0.5].

chitecture is shown in Fig. 5. FPBA is executed in *Hessian*, *Sorting*, *LME*, and *Regression* to generate predicted global motion parameters. Then *SPM PE Array* (Sprite Point Matching Processing Element Array) can execute ISPMA GME to refine the predicted locations of sprite points and output the final results. An off-chip frame memory is required in this system to store the information of current frame and reference frame. The detailed architectures of these modules are described in the following subsections.

4.1. FPBA Part

Hessian calculates the Hessian value of each pixel in the current frame. The detailed architecture is shown in Fig. 6, where a delay line architecture is used. Note that in order to spread feature points in the frame, the frame is divided into four parts, and 13 feature points are chosen in each part. So only delay lines with $W/2$ delay elements are required. Besides, most of the delay elements are implemented with two-port RAM to reduce the hardware cost. *Sorting* module then finds the pixels with the largest 13 Hessian values in each quarter frame as feature points.

Since local motion estimation will be applied at only 52 feature points, only one processing element is sufficient. The architecture is similar to other conventional motion estimation architectures. The motion vectors are then accumulated in *Regression*, and a multi-cycle-shift-and-subtract divider can find the average motion vector as the global motion vector.

4.2. Sprite Point Matching PE Array

Figure 7 shows the detailed architecture of *SPM PE Array*. The SAD values of different locations of sprite points are calculated in different PEs at the same time. In each PE, GMC frame is generated, and the SAD between GMC frame and current frame is calculated. In Fig. 7, *GMC* can generate the corresponding location of current pixel in the reference frame. The operation can be

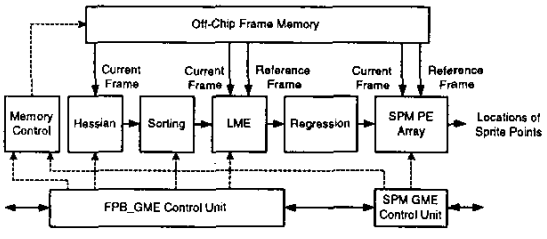

Figure 5: Overview of the proposed hardware architecture.

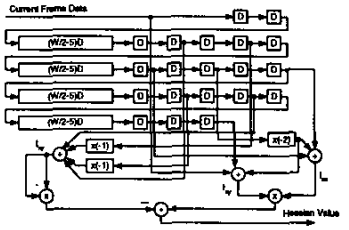

Figure 6: Architecture of Hessian value calculation.

described in the following equations [5]:

$$u(x,y) = C_0 + (C_1 \times x + C_2 \times y)/C_6, \quad (4)$$

$$v(x,y) = C_3 + (C_4 \times x + C_5 \times y)/C_6, \quad (5)$$

$$x' = \lceil u(x,y) \rceil / s, \quad (6)$$

$$y' = \lceil v(x,y) \rceil / s, \quad (7)$$

where C_0, C_1, C_2, C_3, C_4, C_5, C_6 can be directly derived from the locations of sprite points and are constants in whole frame, $C_1 = C_5$ and $C_2 = -C_4$ for GMC with four parameters, s is the precision of warping and can be 2, 4, 8, or 16, C_6 is always power of two, (x, y) is current pixel, and (x', y') is the corresponding pixel in the reference frame. Since the divisors are all power of two, no divider is needed in *GMC*. Besides, the multiplication can be further replaced with addition by the following equations [7].

$$u(0,0) = C_0, \quad (8)$$

$$u(x+1,y) = u(x,y) + C_1/C_6, \quad (9)$$

$$u(x,y+1) = u(x,y) + C_2/C_6. \quad (10)$$

For the function $v(.)$, the same multiplication-free technique can also be used. *Set Constant* first generates the constants C_0, C_3, C_1/C_6, and C_2/C_6 for each sprite point location. *GMC* then finds the corresponding location (x', y') of current point (x, y). Next, address generator (*AG*) derives the address to access the local memory (*LM*), which contains four banks and can output four values at the same time. Bi-linear interpolation is executed in *Interpolation* to find the pixel value. The SAD value between current frame and GMC frame is stored in *SAD Register*. Finally, *Compare Tree* can find the minimum SAD values and get the optimal sprite point location. Note that an on-chip *Reference Frame Data Buffer* is required for data-reuse, and the content of *LM* of all PEs are the same in order to simplify the memory access.

5. SIMULATION RESULTS

5.1. Proposed Algorithm

The PSNR curve of the GMC frames are shown in Fig. 8, where GME in VM, FPBA with 52 and 800 feature points, and the proposed hybrid algorithm (FPBA+ISPMA) are compared. In Fig. 8(a), sequence *Stefan* in CIF format is tested, which contains panning and zooming camera motion. It shows that the performance of VM GME and the proposed one are very similar and hard to be distinguished. The performance of FPBA is not stable and not robust. The PSNR has vibration, which can be observed when playing back the global motion compensated sequence. The results of sequence *Table Tennis* is presented in Fig. 8(b), where a scene change occurs at frame 132, and a large zooming motion occurs at

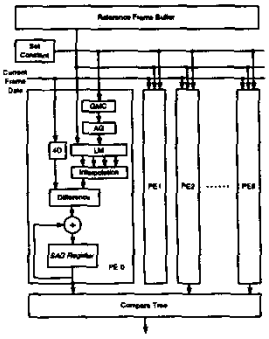

Figure 7: Architecture of sprite point matching global motion estimation PE array.

Table 1: Result of hardware implementation of GME for MPEG-4 ASP@L3.

Unit	Gate count	Internal memory size(bit)
Hessian	3328	5632
Sorting	9918	0
LME	856	2016
Regression	999	0
SPM PE Array		
9 PEs	45819	3528
Set Constant	3612	0
Compare Tree	1584	0
Frame Buffer	0	19712
Total	66116	30888

the first 30 frames. It is shown that the proposed algorithm has the same performance as VM GME and is robust to scene change.

5.2. Hardware Implementation

The results of hardware implementation of the proposed hardware architecture for GME is shown in Table 1. The target is ASP@L3, that is, the input frame is in CIF format, The hardware architecture is implemented and simulated with Verilog-HDL and synthesized with SYNOPSYS Design Compiler. We adopt AVANT! 0.35 μm cell library, and the target working frequency is 100 MHz. Nine PEs are required in *SPM PE Array* in this clock rate. In Table 1, the dominant module is *SPM PE Array*, and the total gate count is about 66K and the required internal memory size is about 31Kb, which is feasible in today's technology and is reasonable to be integrated into MPEG-4 ASP encoders.

6. CONCLUSION

In this paper, the hardware implementation issues of global motion estimation (GME) in MPEG-4 Advanced Simple Profile is discussed. Combining a feature points based algorithm and a novel iterative sprite point matching algorithm, a hardware-oriented GME algorithm is proposed. The associated hardware architecture is also proposed in this paper. For MPEG-4 ASP@L3, the gate count is 66K, the internal memory size is 31Kb, and the working frequency is 100 MHz. This architecture is suitable to be integrated into MPEG-4 ASP encoder to provide GME/GMC ability.

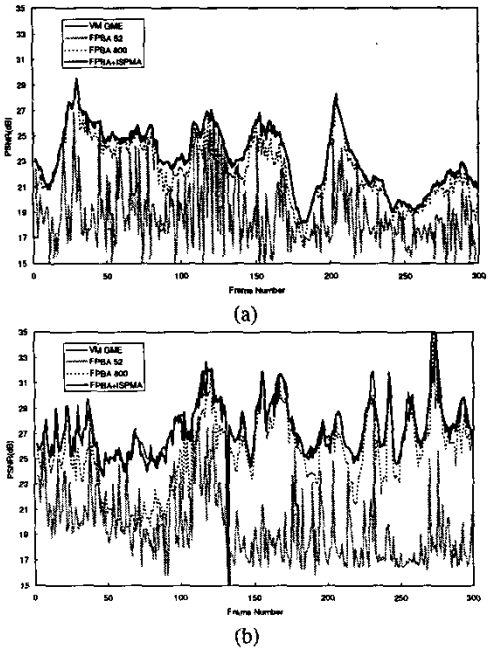

Figure 8: PSNR curve of VM GME, feature point based GME, and the proposed GME algorithms. (a) *Stefan*. (b) *Table Tennis*

7. REFERENCES

[1] MPEG Video Group, *AMENDMENT 4: Streaming Video Profile*, ISO/IEC JTC 1/SC 29/WG11 N3904, 2001.

[2] A. Luthra, R. Gandhi, K. Panusopone, K. Mckoen, D. Baylon, and L. Wang, "Performance of MPEG-4 profiles used for streaming video," in *Proc. of 2002 Workshop and Exhibition on MPEG-4*, 2002, pp. 103 –106.

[3] F. Moscheni, F. Dufaux, and M. Kunt, "A new two-stage global/local motion estimation based on a background/foreground segmentation," in *Proc. of International Conference on Acoustics, Speech, and Signal Processing 1995*, 1995, pp. 2261–2264.

[4] D. Adolph and R. Buschmann, "1.15Mbit/s coding of video signals including global motion compensation," *Signal Processing: Image Communication*, vol. 3, no. 2, 1991.

[5] MPEG Video Group, *The MPEG-4 Video Standard Verification Model version 18.0*, ISO/IEC JTC 1/SC 29/WG11 N3908, 2001.

[6] A. Smolic, T. Sikora, and J.-R. Ohm, "Long-term global motion estimation and its application for sprite coding, content description, and segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 8, pp. 1227–1242, Dec. 1999.

[7] W. Badawy and M. Bayoumi, "A multiplication-free algorithm and a parallel architecture for affine transformation," *Journal of VLSI Signal Processing*, vol. 31, no. 2, pp. 173–184, June 2002.