A Hardware Design for Linear Equation System Solving of VVC Affine ME

Denis Maass, Marcello M. Muñoz, Murilo Perleberg, Marcelo Porto, Luciano Agostini
Graduate Program in Computing
Video Technology Research Group - ViTech
Federal University of Pelotas - UFPel, Pelotas, Brazil
denismaass7@gmail.com, {mmmunoz, mrperleberg, porto, agostini}@inf.ufpel.edu.br

***Abstract*—The Affine Motion Estimation, introduced by the Versatile Video Coding standard, contributes to the significantly enhances compression efficiency. On the other hand, it requires a huge computational effort that makes mandatory the use of dedicated hardware. In light of this, this work presents a hardware design that focuses on calculating the ΔMV values utilized in the Affine Motion Estimation process, specifically for refining the Affine Motion Vector. Synthesis results show that the developed architecture can reach an accuracy of 99.95% while dissipating 5.38 mW in an area of 222.84 Kgates, running at 55.08 MHz, the operational frequency required to process 120 frames per second of 4K resolution videos.**

***Index Terms*—VVC, Affine Prediction, AMVP mode, Hardware Design**

I. Introduction

The demand for digital videos has increased significantly in recent years. However, due to the substantial amount of data required to represent digital videos, it has mandatory to employ compression techniques to enable efficient storage and transmission of video content. Currently, the Versatile Video Coding (VVC) standard [1], released in 2020, is the state-of-the-art on video compression.

To achieve efficient video coding is essential to explore temporal redundancies, which refer to the similarity between neighboring frames. Motion Estimation (ME) is the main technique employed to deal with temporal redundancies. It is present in the Inter-Frame prediction process of almost all previous video coding standards. However, the VVC introduces a novelty tool in the Inter-frame process called Affine Motion Estimation [2]. Unlike conventional ME, which can only represent translational movements, Affine ME offers greater flexibility in Inter-Frame prediction by representing complex movements like rotation, scaling, and skew [1]. The substantial difference between the two versions of ME is that the conventional ME uses a single Motion Vector (MV) to describe the motion between two frames, while Affine ME has the capability to employ two or three MVs to represent the movement. [1].

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It was also financed in part by the Fundação de Amparo à pesquisa do Estado do Rio Grande do Sul – Brasil (FAPERGS), and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq).

Using Affine ME and other advanced tools, the VVC standard achieves a reduction of up to 44% when compared to its predecessor, the High-Efficiency Video Coding (HEVC) [3]. Previous evaluations conducted by our research group have demonstrated that disabling Affine ME results in a decrease of 2.14% in compression efficiency [4].

In order to obtain the set of MVs that describe the complex movement between two frames, an initial tuple of MVs is inherited from the motion information of previously processed tools or even from neighboring blocks. These inherited MVs may not precisely capture the motion details. Therefore, the VVC Test Model (VTM) [5], the reference software of the VVC Standard, employs an iterative algorithm that optimize and refine the set of MVs, seeking the best set of MVs that better describe the motion [6]. This iterative algorithm evaluates several sets of Affine MVs. After evaluating each set, the obtained reconstructed block are processed regarding its difference to the original block, which results in a ΔMV. When the ΔMV is accumulated to the last set of MV evaluated, it generates the new set of MVs to be evaluated, as it will be described in Section II.

Due to the considerable computational complexity involved in the iterative process of calculating the ΔMV, it becomes essential to employ a dedicated hardware design to handle it efficiently, particularly for real-time applications that target battery-powered devices. In this context, this work proposes a hardware design to calculate the ΔMV of Affine ME of the VVC standard, which can process 128×128 size blocks. The ASIC synthesis results for TSMC 40 nm show that for UHD 4K@120 fps, the architecture dissipates 5.38 mW with an area of 222.84 Kgates, reaching an accuracy of 99.95%.

II. Affine Motion Estimation

The Affine ME is employed to capture and represent the complex movement that occurs within digital videos. As in conventional ME, Affine ME searches for blocks in previously encoded frames that are similar to the block being currently encoded. The VVC standard supports Affine ME with four and six parameters. The Fig. 1 illustrates the different parameter configurations used in Affine Motion Estimation (AME). In (a), the Affine 4-parameters configuration is depicted, which utilizes two Motion Vectors (MVs) located at the top corners

of the block. On the other hand, (b) represents the Affine 6-parameters configuration, which employs three MVs, including the MV from the bottom-left corner of the block in addition to the two MVs at the top corners [7].

There are two modes to get the MVs to perform Affine ME: Affine merge mode and Affine Advanced Motion Vector Prediction (AMVP) mode [5]. In Affine merge mode, the MVs are obtained from neighboring blocks without requiring complex calculations. In contrast, the AMVP mode involves an iterative algorithm that requires several operations [6]. The AMVP algorithm will be detailed in the next subsection.

A. AMVP Algorithm

As previously mentioned, the AMVP Algorithm is responsible for defining the set of MVs that better describe the complex motion of the current block regarding the reference frame. It starts by evaluating the Affine with four parameters, and after it evaluates the six parameters [5]. The essence of this algorithm is to adjust the MVs aiming to minimize the mean square error (MSE) between the current block and the prediction block.

AMVP Algorithm starts by inheriting a set of MVs from an initial candidate. This initial candidate is used as the starting point for the gradient-based algorithm that calculates a ΔMV, which will be added to the previous MVs to generate a new set of MVs to be tested [2]. These sets of MVs may be inherited from the conventional ME tool previously applied over the current block, or even from neighboring blocks that were represented using either the conventional ME or the Affine ME.

To evaluate the quality of each candidate set of MVs, the AMVP reconstructs temporarily the current block using each set of MVs. After the reconstruction, an error is calculated by taking the difference between the current block and the reconstructed block. This error represents the discrepancies between the original block and the block generated using the candidate MVs.

Subsequently, Sobel filters are applied two times over the reconstructed block, producing a horizontal gradient block and a vertical gradient block. The (1) and (2) originate each sample of the horizontal gradient and vertical gradient, respectively. In (1) and (2), the RB represents the Reconstructed Block, while j e k represents the row and column of the sample being filtered, respectively.

$$\begin{aligned} gradH[j][k] =& (RB[j-1][k+1] - RB[j-1][k-1] + \\ & (2 * RB[j][k+1]) - (2 * RB[j][k-1]) + \\ & RB[j+1][k+1] - RB[j+1][k-1]) \end{aligned} \quad (1)$$

$$\begin{aligned} gradV[j][k] =& (RB[j+1][k-1] - RB[j-1][k-1] + \\ & (2 * RB[j+1][k]) - (2 * RB[j-1][k]) + \\ & RB[j+1][k+1] - RB[j-1][k+1]) \end{aligned} \quad (2)$$

The gradient blocks obtained from applying the Sobel filter are utilized in conjunction with the error value to calculate the ΔMV values. This process involves constructing a matrix with dimensions of 7×7 when using the 6-parameters model or a 5×5 matrix when using the 4-parameters model. The matrix coefficients are obtained using (3), where col and row represent each matrix position, ranging from 0 to 4 or 6, according to the number of Affine parameters. The iC values are given by the gradient values, as shown by (4) which givens the iC values for the processing of the 6-parameters model. Finally, the last matrix row is filled using (5).

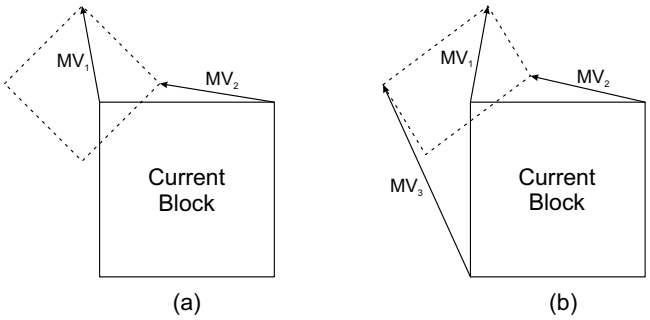

Fig. 1. Affine ME with (a) Affine 4-Parameter and (b) Affine 6-Parameter.

$$mtCoeff[col+1][row] = \sum_{j=0}^{CB_H} \sum_{k=0}^{CB_W} int(iC[col] * iC[row]) \quad (3)$$

$$\begin{aligned} iC_{6param} = [& gradH[j][k], \\ & (((k >> 2) << 2) + 2) * gradH[j][k], \\ & gradV[j][k], \\ & (((k >> 2) << 2) + 2) * gradV[j][k], \\ & (((j >> 2) << 2) + 2) * gradH[j][k], \\ & (((j >> 2) << 2) + 2) * gradV[j][k]] \end{aligned} \quad (4)$$

$$mtCoeff[col+1][lastRow] = \sum_{j=0}^{CB_H} \sum_{k=0}^{CB_W} int(iC[col] * error[j][k]) \quad (5)$$

The matrix of coefficients is actually a system of linear equations that represents the relationship between the gradient blocks and the error value. When solved, this matrix originates the ΔMV values [2], which allows the estimation of the optimal set of MVs. To solve this system, the Gaussian elimination with partial pivoting method [8] is employed in VTM software [5], as is explained in the next subsection.

B. Gaussian Elimination with Partial Pivoting

Gaussian Elimination with Partial Pivoting is a numerical method used to solve systems of linear equations. In this application, the process starts in column zero and row one, then is performed the steps as follows:

1) Find the element whit the largest absolute value in the current column, considering the rows below the current position. This element is called a pivot.
2) If the pivot element is not in the current row, swap the current row with the row that contains the pivot element.
3) Perform row operations to make all the elements below the pivot equal zero. To do this, each coefficient below the pivot's row is recalculated according to (6).
4) Move to the next column and next row and repeat steps 1 to 3 until the system of equations is transformed into an upper triangular form.

5) Solve the upper triangular system of equations by backward substitution, starting from the last row and working upwards.

$$\begin{aligned} newCoeff[col][row] = & mtCoeff[col][row] - \\ & (mtCoeff[col][pivotRow] * \\ & mtCoeff[pivotCol][row] / \\ & pivotValue) \end{aligned} \quad (6)$$

The result of this process is an array whit four or six elements, depending on the Affine parameters. Observing the reference software of the VVC standard [5] can be seen that there is a final processing that involves the size of the current block to be performed over this array. The result of this process is, finally, the ΔMV array.

Since that ΔMVs gives the difference between the sets of MVs for evaluation, they are accumulated to the last set of MVs, thus generating the new set of MVs for evaluation. The process is repeated until the new MVs be identical to the last one, in other words, until ΔMV equals zero, or until the process reaches a limit of evaluated MVs [5]. According to empirical studies, six to eight iterations should be enough to find the optimal set of MVs [2] [5].

III. Proposed Hardware Design

This section presents the architecture proposed to obtain the ΔMV, required to process the Affine ME as performed in VVC reference software. This architecture solves a linear equation system by applying the Gaussian elimination with partial pivoting. Thus, the developed hardware inputs are a 32-bit coefficient matrix of size 7×7, that can be used as a 5×5 matrix just ignoring two rows and columns. The matrix's size depends on the 1-bit input Affine type, that defines the Affine model used (4-parameters or 6-parameters). The architecture also receives two 8-bit values that represent the width and height of the current block. As output, the developed design delivers a 19-bit ΔMV array with 6 or 4 values.

The complete architecture is shown in Fig. 2. As can be seen, it is composed of four modules inside the Gaussian Elimination with Partial Pivoting architecture, where each one performs a step of the Gaussian elimination with partial pivoting, plus one additional module to perform the Final Processing according to the Current Block size [5]. The *Identify Pivot* module receives the elements of the current column and compares them in order to find the highest element in the column, which will be the pivot element. The *Identify Pivot* compares two elements at each clock cycle. The comparison output is a 1-bit signal that informs which input has the higher absolute value, and this bit is stored to be used for defining the elements in the next comparisons on the next cycle. Once the pivot is found, and if it is different from zero, the module delivers the index of the line that contains it, otherwise, the process stops and the ΔMV is considered a null array.

The index of the pivot element and the coefficient matrix is the inputs of *Swap Rows* module. This structure is a set of multiplexers that rearranges the matrix to ensure that the pivot element is in the current row. This process is purely combinational, then the output matrix goes to the *Row Operations* module, where the elements below the pivot are turned to zero. This is performed by applying (6) to each element below the pivot row and to the right of the pivot column. This module takes one clock cycle to calculate each new coefficient, therefore, up to 30 cycles can be necessary to calculate the new elements.

The three processing from *Identify Pivot*, *Swap Rows* and *Row Operations* modules must be repeated until all elements below the main diagonal be zero, forming an upper triangular matrix. That means processing these modules five times for Affine 6-parameters and three times for Affine 4-parameters. So, the upper triangular system follows to the *Solve System* module to be solved by a backward substitution. This process consists of first solving the last row and using its result to solve the above row, working upwards until solving all equations. Finally, the values obtained are submitted to the *Final Processing* module. This module takes into account the block size information and also converts the elements' values to integers, thus delivering the ΔMV.

All the modules in the architecture are synchronized by the control module. The control module receives the Affine type information and incorporates a cycle counter that helps in coordinating and timing the flow of data between different modules. To perform all the described operations and obtain the ΔMV array, the proposed architecture may take up to 103 clock cycles to.

IV. Accuracy Analysis

Although the architecture's inputs and outputs are integer values, fractional values come up during the process of divider operations. In addition, the several operations performed between fractional values can introduce accumulative errors, depending on the precision adopted by the system. So, the accuracy of the fractional values depends on the resolution adopted, which refers to the number of bits used to represent the fractional components. Higher resolutions offer a more precise representation of fractional values but also require more area and power resources to be implemented.

Taking that into consideration, the proposed architecture was implemented in VHDL using two different versions. One version utilizes a 16-bit fractional resolution, while the other version employs an 8-bit fractional resolution. Implementing the architecture in different resolutions allows for a comparison of their respective trade-offs in terms of accuracy versus resource requirements and power consumption.

The accuracy analysis was performed using ModelSim Tool and real data related to 100k ΔMV operations from the RaceHorses sequence [9]. Since each ΔMV operation may generate four or six output values, the 100k operations originate 434626 output values. Table I shows the accuracy results of the two different versions. As can be seen, the 16-bit version of the proposed architecture exhibits a high level of accuracy that reaches 99.95% of the output values. Its performance demonstrates precise and reliable functionality, making it a suitable choice for demanding applications that

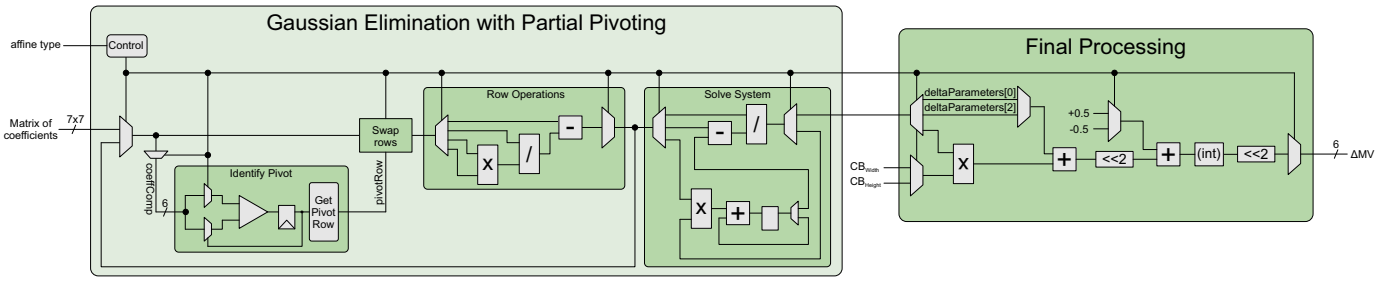

Fig. 2. The proposed architecture to get the ΔMV values.

require accurate video processing. On the other hand, the 8-bit version also showcases a relatively good level of accuracy, achieving a success rate of nearly 90%.

TABLE I
ACCURACY ARCHITECTURE WITH DIFFERENT RESOLUTION

Fractional Resolution	8	16
Correct Outputs	388386	434390
Accuracy	**89.36%**	**99.95%**

V. SYNTHESIS RESULTS

As said, the proposed architecture was implemented in VHDL. The two versions of the proposed architecture were synthesized to an Application-Specific Integrated Circuit (ASIC) using the Cadence RTL Compiler [10]. The synthesis process was performed considering the 40 nm standard-cell library provided by TSMC [11]. The target frequency for the ASIC design was set at 55.08 MHz, ensuring the capability to process blocks of size 128×128 samples in Ultra High Definition (UHD) 4K (3840×2160) resolution at 120 frames per second.

Table II shows the synthesis results of the proposed architecture, where the area results consider the number of equivalent gates, considering the area of a NAND2 (0.9408 μm^2). As expected, the 16-bit requires more resources in terms of both area and power consumption compared to the 8-bit version. The 16-bit version exhibits a significant increase of 56.70% in terms of area utilization and a 47.72% increase in power consumption when compared to the 8-bit version. On the other hand, as presented in Table I, the 16-bit version reaches more than 99.9% accuracy in the elements of ΔMV, while only 89.3% accuracy was obtained with 8-bit precision.

TABLE II
SYNTHESIS RESULTS OF THE TWO VERSIONS OF THE PROPOSED ARCHITECTURE, RUNNING AT 55.08MHZ

Fractional Resolution	8	16
Leakage Power (mW)	0.169	0.256
Dynamic Power (mW)	3.477	5.130
Total Power (mW)	**3.646**	**5.386**
Cell Area (Kgates)	**142.20**	**222.84**

VI. CONCLUSIONS

This work presented dedicated hardware to calculate the ΔMV values, necessary to process the Affine ME as performed in VVC reference software. Two different versions of architecture, taking into account the fractional resolution, were evaluated. The analysis and synthesis results show that the 16-bit version provides higher accuracy, but it comes at the expense of increased resource utilization and power consumption. On the other hand, the 8-bit version offers a more resource-efficient solution with slightly reduced accuracy.

Future works aim to improve this architecture, especially in paralleling operations to reduce the number of clock cycles required to process each matrix of coefficients and thus increase the throughput, while also preparing this architecture to be embedded in a hardware design to perform the complete Affine ME.

REFERENCES

[1] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[2] L. Li, H. Li, D. Liu, Z. Li, H. Yang, S. Lin, H. Chen, and F. Wu, "An efficient four-parameter affine motion model for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1934–1948, 2018.

[3] Siqueira, G. Correa, and M. Grellert, "Rate-distortion and complexity comparison of hevc and vvc video encoders," in *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, 2020, pp. 1–4.

[4] P. H. R. Gonçalves, "Um esquema rápido baseado em aprendizado de máquina para a predição interquadros do codificador de vídeo vvc," Master's thesis, Universidade Federal de Pelotas, 2021.

[5] A. Browne, Y. Ye, and S. Kim, "Algorithm description for versatile video coding and test model 17 (vtm 17)," JVET-Z2002, July 2022.

[6] M. Martina, "Simplified affine motion estimation algorithm and architecture for the versatile video coding standard," Ph.D. dissertation, Politecnico di Torino, 2022.

[7] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Versatile video coding editorial refinements on draft 10," JVET-T2001, October 2020.

[8] W. Ford, "Chapter 11 - gaussian elimination and the lu decomposition," in *Numerical Linear Algebra with Applications*, W. Ford, Ed. Boston: Academic Press, 2015, pp. 205–239. [Online]. Available: www.sciencedirect.com/science/article/pii/B9780123944351000119

[9] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, "Vtm common test conditions and software reference configurations for sdr video," JVET-T2010, Teleconferência, October 2020.

[10] CADENCE. (2020) Cadence rtl compiler. [Online]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/training/all-courses/84441.html

[11] TSMC. (2008) 40nm technology. [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_40nm