Software Solution for Hardware Optimization of 1-D DCT of the HEVC

Ruhan Conceição
José Cláudio Sousa Júnior
Universidade Federal de Pelotas
CDTec - PET Computação
Pelotas, Brazil
{radconceicao, jcdsouza}
@inf.ufpel.edu.br

Ricardo Jeske
Universidade Federal de Pelotas
CDTEC – PPGC
Pelotas, Brazil
rgjeske@inf.ufpel.edu.br

Luciano Agostini
Julio C. B. Mattos
Universidade Federal de Pelotas
CDTEC
Pelotas, Brazil
{julius,agostini}@inf.ufpel.edu.br

ABSTRACT

This paper presents a software solution for hardware optimization of 1-D DCT 32 points used in the emerging video coding standard HEVC – High Efficiency Video Coding. The 1-D DCT is used by the 32x32 2-D DCT of the HEVC standard. The transforms stage is one of the innovations proposed by HEVC, not only because of the variable size (from 4x4 to 32x32) but also because higher dimension transforms other than the traditional 4x4 and 8x8 are used. The software presented in this work is design to test more than five billions combinations of hardware for 1-DCT 32 points operations, in order to get the maximum sharing of operations. Thus, through the result obtained by the software, it will be able to implements an efficient and optimized hardware of the 1-D DCT 32 points. Since a huge number of possibilities, this paper presents partial results generated by the software.

Categories and Subject Descriptors

B.7. [INTEGRATED CIRCUITS]: Types and Design Styles – *Algorithms implemented in hardware.*

General Terms

Algorithms, Performance.

Keywords

Hardware Optimization, DCT, HEVC, Video coding.

1. INTRODUCTION

Nowadays, the resolution and the quality of digital videos have been improving in a fast and steady manner. Additionally, such videos are becoming supported by an increasing number of electronic devices. Thus, the improvement of video encoders/decoders in an extremely relevant activity in the current scenario, since the many devices that process digital videos, must be capable of processing high-resolution videos in real time. For this reason, topics such as compression rate, video quality, computational complexity and energy consumption must be improved, hence they are thoroughly investigated this area.

Video coding is imperative in applications that handle digital videos, since an uncompressed video requires a large volume of bits to be represented [1]. H.264/AVC [2] is the latest video coding standard available, presenting significant gains in compressions when compared to the MPEG-2 standard [3]. On January 2010, the JCT-VC (Joint Collaborative Team – Video Coding) was created, composed of experts from ITU-T and ISO/IEC, to start the development of a new video coding standard called HEVC – High Efficiency Video Coding [4]. The goal of the JCT-VC is to increase video compression in 50% while maintaining the same computational complexity. HEVC is still under development, but some important modifications related to H.264/AVC were already defined.

On HEVC, each frame is divided into a sequence of square units called treeblocks, which hold the information of chrominance and luminance. The chrominance blocks dimensions depend on the color sampling used. The current version of HEVC defines treeblocks as areas containing 64x64 luminance samples and their corresponding chrominance samples [5] [6].

Each treeblock is composed of one or more basic Coding Units (CU). A CU can be recursively divided into four blocks of the same size starting from the treeblock and going all the way down to a minimum of 8x8 samples. This recursive process forms a quadtree composed of CU blocks, assuming dimensions that vary from 8x8 pixels to the size of the treeblock itself, in other words, 64x64. Fig. 1 illustrates one possible partitioning of a treeblock, forming a CU quadtree.

A generic video encoder can be represented as a sequence of stages, where each stage is responsible for part of the coding process. Among the different stages, the transforms hold an important position. Normally by utilizing a DCT – Discrete Cosine Transform, the purpose of the transforms stage is to concentrate the energy of an image in just a few numerical coefficients. In doing so, the following stages (quantization and entropy coding) can be performed in a much more efficient way.

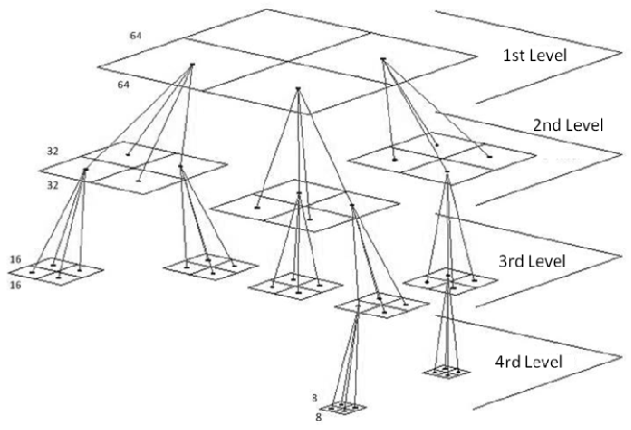

Figure 1. Treeblock partitioning example

On HEVC, the basic units for the transform and quantization operations are called Transform Units (TU). Their format is always square and their dimensions can vary from 4x4 to 32x32 samples. As occurs with the CUs, TUs can be structured with quadtrees. Each CU can contain one or more TUs.

The computation of DCT requires large number of operations, such as sums, subtractions and multiplications. As it is known, multipliers are very expensive in terms of hardware consumption, making it necessary the use of sums/subtractions and shifts instead these operations. By this replacement, it is possible to share some operations among the equations used in the 1-D DCT 32 points.

The aim of this paper is to present a software that find, among five billions possibilities, the best combinations of operations used in the processing of 1-D DCT 32 points, in order to get the maximum sharing of operations. Therefore, it will be possible to implement an efficient and optimized hardware after, with a less energy and area consumption.

Currently on version 4.0, the HEVC Model (HM) [7] was used as a golden model for this project. This way, algorithmic validation was performed by using data extracted from this software, increasing the reliability of achieved results.

The paper is organized as follows. Section 2 discusses Discrete Cosine Transform used in the emerging video coding standard HEVC. Section 3 presents an overview of the software under development. This software is organized in different modules. Section 4 presents the partial results produced by the software. Section 5 concludes and points out directions for further research.

2. DISCRETE COSINE TRANSFORM

HEVC perform four sizes of 2-D DCT: 4x4, 8x8, 16x16 and 32x32. In order to calculate this transform, it is necessary to perform the 1-D DCT two times as follows. First, given an input matrix, it is calculated the 1-D DCT for each line, and the result is stored column by column in an intermediated matrix. After all lines have been calculated, this process is done again, calculating the 1-D DCT line by line from the intermediated matrix, and the result is stored column by column in the output matrix. Thus, the 2-D DCT is performed from the input matrix to output matrix.

Since this work is focused on the 1-D DCT 32 points, it is explained the processing of this size, although the other ones follows the same idea. Each input of the 1-D DCT 32 points will be called in this paper as W_n, which n represents the index of the input (from 0 to 31). Sums and subtractions are performed among the input, generating results denominated in this paper as a_n. The sums and subtractions are done as follows in Table 1.

Table 1. First Operations on the Algorithm

Stage "a"	Inputs
a_0	$W_0 + W_{31}$
a_1	$W_0 – W_{31}$
a_2	$W_1 + W_{30}$
a_3	$W_1 – W_{30}$
…	…
a_{31}	$W_{15} – W_{16}$

All a's indexed by odd numbers are straight used in the final equation, where they are multiplied by the constants. The other ones are operated among each other following the same idea of the inputs. The focus of this work is the odd a's, since the even a's could be treated as 1-D DCT 16 points [8].

Table 2 shows some equations that use the results of the odd a's in their processing. Every X_n represents the position of which equation in the output vector indexed by the respective n.

Table 2. Some equations used in the computation of 1-D DCT 32 points

X_n	Final equation
X_1	$90*a_1 + 90*a_3 + 88*a_5 + 85*a_7 + 82*a_9 + 78*a_{11} +$ $+73*a_{13} + 67*a_{15} + 61*a_{17} + 54*a_{19} + 46*a_{21} +$ $+38*a_{23} + 31*a_{25} + 22*a_{27} + 13*a_{29} + 4*a_{31}$
X_3	$90*a_1 + 82*a_3 + 67*a_5 + 46*a_7 + 22*a_9 - 4*a_{11} +$ $-31*a_{13} - 54*a_{15} - 73*a_{17} - 85*a_{19} - 90*a_{21} +$ $-88*a_{23} - 78*a_{25} - 61*a_{27} - 38*a_{29} - 13*a_{31}$
…	…
X_{31}	$4*a_1 - 13*a_3 + 22*a_5 -31*a_7 + 38*a_9 - 46*a_{11} +$ $+54*a_{13} + 61*a_{15} + 67*a_{17} - 73*a_{19} + 78*a_{21} +$ $-82*a_{23} + 85*a_{25} - 88*a_{27} + 90*a_{29} - 90*a_{31}$

Every constants used in each equation, is the same in the other ones, although some of them can be negated. In the next topic it will be explained how the software works, searching for the best sharing of operations through the final equations.

3. THE SOFTWARE

The software was developed in *C* programming language. It is divided in seven modules, which each one is responsible for a part of the process to find the greater number of operations shared among all sixteen equations.

3.1 Module 1 – Sum-and-shift generation

For each constant that is used in the final equations, the module 1 generates sums and shifts that can be used instead the multiplications. Module 1 purposes only solutions that used up to 4 sums and shifts until six bits.

Table 3 shows an example of module 1 result from the constant 13. Each column represents the shifts and sums that can be used instead the multiplications, and each line shows all these possibilities.

Table 3. An example of module 1 results for the constant 13

	Number of shifted bits							=
	6	**5**	**4**	**3**	**2**	**1**	**+1**	
13	0	0	0	+	+	0	+	<<3 + <<2 + 1
13	0	0	0	+	+	+	-	<<3 + <<2 + <<1 – 1
13	0	0	+	-	+	0	+	<<4 - <<3 + <<2 + 1
13	0	0	+	0	-	0	+	<<4 - <<2 + <<1 + 1
13	0	0	+	0	-	+	-	<<4 - <<2 + <<1 - 1
13	0	0	+	0	0	-	-	<<4 - << 2 - 1
13	0	+	-	0	-	0	+	<<5 - <<4 - <<2 + 1
13	0	+	-	0	0	-	-	<<5 - <<4 - <<1 - 1

The result generated from module 1 to each constant is used as an input on the module 2.

3.2 Module 2 – Sum-and-shift combination

There are sixteen constants and the module 1 generates for each one all possibilities to swap the multiplications to sums/subtractions and shifts. The module 2 basically combines all possibilities generated for each constant in module 1 with all possibilities of the other constants. Altogether are generated more than 5 x 10^9 possibilities. Each combination is used as an input to the module 3.

This number is obtained (five billions of possibilities) through multiply of the number of possibilities generated for each constant from module 1, this computation is shown in equation 1. Table 4 shows the number of sum-and-shift possibilities generated for each constant.

Table 4. Number of sum-and-shift possibilities to be used instead multiplications for each constant

Constants	Number of combinations	Constants	Number of combinations
90	3	**61**	4
90	3	**54**	6
88	3	**46**	6
85	1	**38**	7
82	3	**31**	5
78	4	**22**	8
73	3	**13**	8
67	4	**4**	4

$$N\_possibilities = 3x3x3x1x3x4x3x4x4x6x6x7x5x8x8x4$$
$$N\_possibilities = 5{,}016{,}453{,}120 \qquad (1)$$

If the module 1 did not restrict the number of sums up to four, would be generated almost 2 x 10^{15} possibilities, becoming unfeasible the software performing.

3.3 Module 3 – Adaptation for all equations

Table 2 shows that the order and the signal of the constants change among the sixteen equations. Therefore, the module 3 adapts the generated combination on the module 2 for each of the sixteen equations. Moreover, this module organizes the equations by the shifts, and no more by the constants multiplications.

For example, the equation X_1 can be represented as Equation 2.

$$\begin{aligned} X_1 = {} & (a_1+a_3+a_5+a_7+a_9+a_{11}+a_{13}+a_{15}+a_{17}) >>6 + \\ & + (a_{19}+a_{21}+a_{23}+a_{25}) >>5 + \\ & + (a_1+a_3+a_5+a_7+a_9+a_{19}+a_{27}) >>4 + \\ & + (a_1+a_3+a_5+a_{11}+a_{13}-a_{17}+a_{21}+a_{29}) >>3 + \\ & + (a_7+a_{11}+a_{17}+a_{19}+a_{21}+a_{23}-a_{25}+a_{27}+a_{29}+a_{31}) >>2 + \\ & + (a_1+a_3+a_9+a_{11}+a_{15}+a_{19}+a_{21}+a_{23}+a_{25}+a_{27}) >>1 + \\ & + (a_7+a_{13}+a_{15}+a_{17}+a_{25}+a_{29}) \qquad (2) \end{aligned}$$

This type of result is generated for all sixteen equations and used as an input in the module 4.

3.4 Module 4 – B-operations generation

Module 4 basically groups the *a*'s in pairs, generating sums and subtractions operations called *b*. For each operation *b* created, the *a*'s pair is replaced by the respective *b* everywhere they are found among the sixteen equations.

Two methods are done to choose which operation would be created. These methods are called *higher occurrence* and *pair search* and they will be explained below.

3.4.1 Higher Occurrence

This method searches which pair of *a*'s has higher occurrence among all equations. After it is found, the software replaces the *a* pair by a *b* operation. Then, the software begins to find again other *a*'s which is the pair that has higher occurrence. This process is finished when there is not more than one '*a*' in each line.

For example, if the sum "$a_1 + a_3$" were elected as the operation with the higher occurrence, the Equation 2 will be transformed in Equation 3.

$$\begin{aligned} X_1 = {} & (\boldsymbol{b_0}+a_5+a_7+a_9+a_{11}+a_{13}+a_{15}+a_{17}) >>6 + \\ & + (a_{19}+a_{21}+a_{23}+a_{25}) >>5 + \\ & + (\boldsymbol{b_0}+a_5+a_7+a_9+a_{19}+a_{27}) >>4 + \\ & + (\boldsymbol{b_0}+a_5+a_{11}+a_{13}-a_{17}+a_{21}+a_{29}) >>3 + \\ & + (a_7+a_{11}+a_{17}+a_{19}+a_{21}+a_{23}-a_{25}+a_{27}+a_{29}+a_{31}) >>2 + \\ & + (\boldsymbol{b_0}+a_9+a_{11}+a_{15}+a_{19}+a_{21}+a_{23}+a_{25}+a_{27}) >>1 + \\ & + (a_7+a_{13}+a_{15}+a_{17}+a_{25}+a_{29}) \qquad (3) \end{aligned}$$

3.4.2 Pair Search

This method searches only two *a*'s in each line. If the software find it, then this pair is called as an operation b_n and every occurrence is replaced by the respective operation *b*.

If it is not found any pair, the software performs the higher occurrence methods.

3.5 Module 5, 6 and 7 – C, D and E-operations generation

The next modules follow the same idea of the module 4. The module 5 perform the *higher occurrence* and the *pair search* among the operations *b* and the remaining *a*'s generating the operations called *d*. The same is done by the modules 6 and 7, generating respectively *d* and *e* operations.

3.6 Higher Occurrence or Pair Search

For each combination generated in the module 2 and adapted by the module 3, the next modules perform two methods in order to replace the reminiscent operations, higher occurrence and pair search. The software tests all possibilities, using these two methods in every module. Thus, for each of the five billion combinations, it is done sixteen combinations through the use of higher occurrence and pair search methods.

The software selects as the best combination the output produced by the Equation 4.

$$N\_bits = 10 * nb + 11 * nc + 12 * nd + 13 * ne \qquad (4)$$

N_bits is the number of bits used considering all adders. For example, the sum *b* needs 10-bit adders, and *nb* represents the number of *b* operations used by the current combination. Thus, considering that the operations *c, d* and *e* need respectively 11-bit, 12-bit and 13-bits adders, it is possible to compute the number of bits (*N_bits*).

4. RESULTS

The results of this paper are partial results because until the present moment, the software does not process all combinations, since there are more than five billons possibilities of them. However it is possible to show some results.

The software could find a combination that generates only 37 *b* operations; meanwhile manually it was found a combination that resulted in 104 *b* operations at least. The manually way follows the same technique used by Jeske [9], although it was developed for the 1-D DCT 16 points.

Table 5 shows comparative results between the software and the manual optimization. The better result shows the less number of *b* operations found by the module 4, from the configuration generated by the module 2. In the other hand, the worse result shows the opposite, the higher number of *b* operations.

Table 5. Comparative between the software and the manual optimization

Operation	Result	Number of b's
Manual	Single	106
Software	Better	37
Software	Worse	143

The result generated by the software was validated using the HM as a golden model. The main goal of the software has been achieving, due it is possible to generate a validated hardware configuration extremely faster.

5. CONCLUSIONS

The emerging video coding standard – HEVC - is being developed in order to fulfill the demand for high-resolution videos, which are supported by an increasing variety of devices and applications. Henceforth, algorithmic optimization and architectural design investigations for the coding tools of this standard is an activity of utmost importance in the current scenario.

The aim of this work was to develop a software that is able to find the maximum share of operations in order to produce an optimized hardware. This work is dedicated to the 1-D DCT 32 points of the HEVC, which is part of the 32x32 2-D DCT.

The paper shows that there are more than five billion possibilities to combine all the operations, demonstrating the importance of this work. The partial results show that this work has been achieving its goals.

As a future work, it is planned to design the software for others DCT sizes.

6. ACKNOWLEDGMENTS

This work was supported by grants from the following sponsor agencies: CAPES – Programa PET, CNPq and Fapergs.

7. REFERENCES

[1] Agostini, L. Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas a Compressão de Vídeo Segundo o Padrão H.264/AVC. 2007. 172f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[2] International Telecommunication Union. "ITU-T Recommendation H.264/AVC (03/05): advanced video coding for generic audiovisual services". 2005.

[3] International Telecommunication Union. "ITU-T Recommendation H.262 (11/94): generic coding of moving pictures and associated audio information – part 2: video". 1994.

[4] Joint Collaborative Team on Video Coding (JCT-VC). Available at: http://www.itu.int/en/ITU-T/studygroups /com16/video/Pages/jctvc.aspx

[5] Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 – "HM3: High Efficiency Video Coding (HEVC) Test Model 3 Encoder Description", 5th Meeting: Geneva, CH, 16-23 March, 2011.

[6] W. Han, et al. "Improved video compression efficiency through Flexible Unit Representation and Corresponding Extension of Coding Tools", IEEE Transactions on Circuits and Systems for Video Technology. Vol. 20, pp. 1709-1720. December 2010.

[7] ISO/IEC-JTC1/SC29/WG11, "HEVC Reference Software Manual", ed. Geneva, Switzerland, 2011.

[8] Hecktheuer, B., Conceição, R., Souza, J., Jeske, R., Agostini, L., and Mattos, J. 2012. Reconfigurable Architecture Implementation for the 1-D Discrete Cosine Transform. In *Southern Programmable Logic Design Forum* (Bento Gonçalves, Brazil, March 20 – 23, 2012)

[9] Jeske, R., Souza, J., Wrege, G., Conceição, R., Grellert, M., Mattos, J., Agostini, L. 2012. Low Cost and High throughput Multiplierless Design of a 16 Point 1-D DCT of the new HEVC Video Coding Standard. In *Southern Programmable Logic* (Bento Gonçalves, Brazil, March 20 – 23, 2012)