Optimization of 4x4 Integer DCT in H.264/AVC Encoder

Charles S. Lubobya[1], Mqele M. Dlodlo[1], Gerhard De. Jager[1] and Keith L.Ferguson[2]
Department of Electrical Engineering
University of Cape Town[1], Private Bag X3, Cape Town, 7700
Tel: +27 0788648756
and Meraka institute
Council for Scientific and Industrial Research [2]
Email: {smart.lubobya, mqele.dlodlo,gerhard.dejagar}@uct.ac.za[1]; kferguson@csir.co.za[2]

Abstract- **This paper gives the computation time speed-up improvements obtained for the forward and inverse 4x4 integer Discrete Cosine Transforms (DCT) in H.264/AVC video encoders at constant Peak Signal to Noise Ratio (PSNR). The H.264/AVC specifies use of Integer DCT transforms for decomposing the video signal from the spatial domain to the frequency domain. We implemented the integer DCT using the two-dimension (2-D) matrix multiplications and optimized our implementation using the one-dimension (1-D) butterfly methods in software. All conditions being equal, results show that the two 1-D methods outperforms the 2-D method by an average computation time speed-up of 1.7x to 1.9x. At these speed-ups our optimized implementation can be used in real-time video broadcasting/internet applications and in low bandwidth environment.**

Index Terms— H.264/AVC, Integer DCT, Computation time and SIMD instructions.

I. INTRODUCTION

Transform coding in image and video compression has over the years experienced significant development. The discrete cosine transform has been used in the JPEG, MPEG-1, MPEG-2, H.261 and H.263 Video compression standards [1]-[5]. However, DCT have the weakness of inverse mismatch problems between encoder and decoder, limited block size scalability and complicated hardware implementation [2] [5]. The current standard H.264/ MPEG-4 part 10 or Advanced Video Coding (AVC) also defined by the ITU-T as H.264/AVC has three profiles [2]; baseline, main and extended. Each of these profiles uses the integer DCT to decompose the video signal from the special domain to the frequency domain [4]. Redundant coefficients can then be subjectively removed via quantisation step within the encoder.

The core 4x4 matrix of the integer DCT can be implemented as a direct 2-D matrix multiplication or as 1-D using additions, subtractions and shift operations along the rows and then down the columns, a technique commonly referred to as 'butterfly' [1]-[6].

In this paper, we show the computation speed up improvements of the core forward and inverse 4x4 Integer DCT in software. A comparison between the direct 2-D matrix multiplication method using C++ programming and the optimized 1-D butterfly algorithm method using intrinsic SIMD and assembly SIMD instructions are provided.

The rest of the paper is organized as follows: the next chapter gives the motivation for the optimization methods; Section III gives related work on integer DCT. The H.264/AVC encoder diagram and its description is given in section IV. Section V outlines the experimental methods used in achieving results of section VI and the conclusion is given in VII.

II. MOTIVATION

In recent years some studies have been conducted to reduce the hardware complexities of integer DCT [7]. Most of these were tailored to suit the various block sizes supported by the H.264/AVC standard. These have not, however, adequately addressed the flexibility concerns in the integer DCT algorithm. Providing hand codes for the integer DCT algorithm using loop optimization or low level assembly language coding is one solution. The second is the use of SIMD instructions to optimize the algorithm part on which additions, subtractions and shifts occurs. The Intel‘s Pentium three (3) and four (4), AMD's Athlon XP and K6-2 (and above) and AltiVec's PowerPC®RISC processors have SIMD capabilities [8] [9]. The SIMD instructions in the above stated processors, allows parallel manipulation of matrix elements which represents pixels. Hence, the encoder operations in compressing the video can be speeded up and can therefore be adopted in real time applications. Additionally, the compressed video can be transmitted low bandwidths environment that characterize most developing countries or stored in limited storage capacities.

III. RELATED WORK

Research on Implementation of the integer DCT from both the software and hardware perspectives have been done but never exhausted. Most of these researches are aimed at minimising the complex multiplications and optimising adds and shift operations as a solution. Richardson [2] states that the original proposal for the 4x4 H.264/AVC forward and inverse transform processes describe alternate implementation methods using either a series of add and shift, a flow graph algorithm or matrix multiplication. Malvar *et al* [3] as well as [4][6][10] suggest that we can treat two-dimension (2-D) 4X4 integer DCT as row-column application of one dimension (1-D) methods, that is, separately implement the 1-D four dot integer in each of the rows and columns.

Muhammed *et al* [11] compared the performance of 4x4 Integer DCT with the conventional 8x8 DCT on the basis of computational time and PSNR. The 4x4 integer DCT was 0.30ms times faster than the conventional 8x8 DCT. Jianhong *et al* [6] suggested a fast and efficient parallel implementation of H.264 Integer transforms using SIMD instructions based on 8x8 block size. Xueming *et al* [12] outlined a method of using SIMD instructions and applied the method for the integer and Hadamard transforms. Chen *et al* [13] designed a SIMD matrix multiplication scheme for both integer and Hadamard transforms. They also analysed different multi-threading schemes that have different quality/performance and proposed a scheme with good scalability and quality. Hassaballah *et al* [8] outlined various SIMD optimisation techniques for various processors and their application in the field of engineering. This paper gives the computation speed-up obtained for forward integer DCT, quantisation and backward re-quantisation and inverse integer DCT process based on the 4x4 block size.

IV. H.264/AVC ENCODER BLOCK DIAGRAM

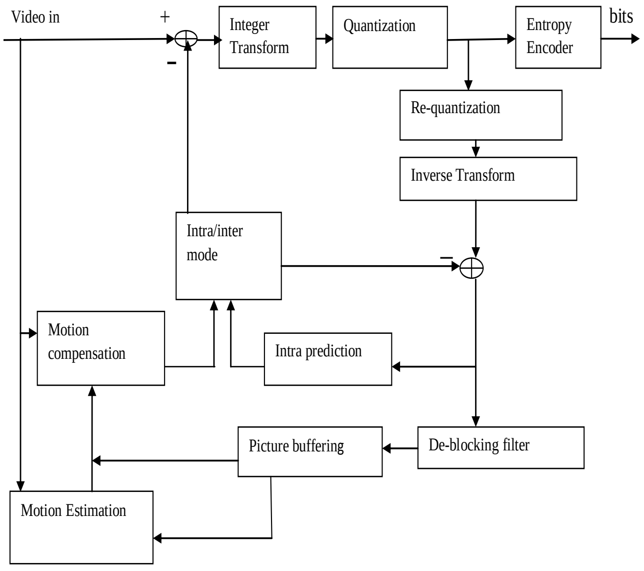


Figure 1: Block Diagram of the Video Encoder [9]

Figure 1 is a block diagram showing the various modules that constitutes the H.264/AVC encoder. A brief description of the major modules and their functions are discussed in this section.

A. *Motion compensation and estimation*

The Motion estimation module is used to identify and eliminate the temporal redundancies that exist between individual frames [3] [4] [5]. It involves use of motion vectors that describes the transformation of the video /image from one dimension to the next. Motion vectors may be applied to the whole image in which case we have global motion estimation or on parts of the image in which it becomes local motion estimation or even per pixel Motion Compensation (MC) will decode the image that is encoded by Motion Estimation [3] [4]. Three types of motion compensation exists depending on which reference frame is chosen: forward MC which uses previous frames as reference, backward MC which uses future frames as reference and bi-directional MC which uses both previous and future frames as reference. This combined motion estimation and compensation accounts for most of computation time in a video encoder [13].

B. *Intra and inter prediction*

An input frame or field is processed in units of a macroblock. Each macroblock is encoded in intra or inter mode and, for each block in the macroblock, a prediction is formed based on reconstructed picture samples [2]. In Intra mode, prediction is formed from samples in the current slice that have previously encoded, decoded and reconstructed. In Inter mode, prediction is formed by motion-compensated prediction from one or two reference picture(s) selected from the set of list 0 and/or list 1 reference pictures [2].

C. *Forward and Inverse Integer DCT transforms*

The forward integer DCT transform is used to convert the spatial domain signal into its equivalent frequency domain [4]. This module essentially, removes temporal redundancies in the residual signal. Mathematically, integer DCT can be derived from discrete cosine transforms and is described as [2] [5] [6]:

$$Y_{ij} = \left(C_f . X . C_f^T\right) \otimes E_f \tag{1}$$

Where by:

$$C_f = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}, \quad E_f = \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}$$

And: $a = \frac{1}{2}$, $b = \sqrt{\frac{2}{5}}$.

Matrices X and Y_{ij} are the input and output video signal respectively. E_f is the post-scaling matrix which is incorporated in the quantisation process [6] while C_f is the transform matrix and C_f^T is its transpose. The operator $\otimes$ means multiplying the corresponding elements of two matrices [2] [6]. From equation (1), the core 4x4 matrix of the integer DCT can be written as:

$$W = C_f . X . C_f^T \tag{2}$$

The inverse integer DCT converts a frequency domain signal back to special domain [13]. Mathematically, It can also be written in similar manner as the forward integer DCT [2] [5].

$$X = C_i^T \left[Y \otimes E_i\right] C_i \tag{3}$$

Where by:

$$C_i = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix}, E_i = \begin{pmatrix} p & q & p & q \\ q & r & q & r \\ p & q & p & q \\ q & r & q & r \end{pmatrix}$$

And: $p = 1/4,\ q = 1/\sqrt{10},\ r = 2/5$

From which the core inverse matrix is [6]:

$$W' = C_i^T . Y . C_i \qquad (4)$$

D. Quantization

The integer transform coefficients are quantised to remove unimportant values, so that only significant coefficients are left for representing the residual frame. The transform coefficients are then quantified to the quantization value associated with the intervals within which the respective coefficients reside [2] [4]. Thus, further reduction of the data representation is achieved. There are two types of quantisers: scalar and vector quantiser. A scalar quantiser maps one sample of the input signal into one quantised output value (for example the process of rounding a fractional number to the nearest integer) and a vector quantiser maps a group of input samples to a group of quantised values [2] [5]. The H.264/AVC adopts the scalar quantiser [2]. The basic quantization operation (Z) for forward integer DCT can be defined as follows [2] [5]:

$$Z = round\left(Y_{ij} / Q_{step}\right) \qquad (5)$$

There are 52 values of Q_{step} supported by H.264/AVC and each has a Quantization Parameter (QP) associated with it [2]. To integrate the scalar multiplication by matrix E_f or E_i as shown above, and to avoid any division operations, the equation is changed to [1] [2] [5]:

$$Z = round\left(W * \left(PF / Q_{step}\right)\right) \qquad (6)$$

Where:

$$PF / Q_{step} = M_f / 2qbits$$

$$qbits = 15 + floor((QP)/6)$$

PF is the value from matrix E_f or E_i depending on the location of Y_{ij}

Equation (6) now becomes [1] [2]:

$$|z_{ij}| = ((|W| * M_f + f)) >> qbits \qquad (7)$$

E. Entropy Coder

Entropy coder module converts the quantized series of symbols representing elements of a video sequences into compressed bitstreams suitable for transmission and / or storage. There are two types of entropy coding techniques: Huffman and arithmetic coding [7]. The H.264/AVC uses an advanced arithmetic coding called context adaptive variable length coding (CAVLC) and context adaptive binary arithmetic coding (CABAC) [2] [7]. The entropy encoder also compresses motion vector and header information. This removes statistical redundancy in the compressed bit stream

F. De-blocking filter

The de-blocking filter reduces the blocking artifacts in the block boundary and prevents the propagation of accumulated coded noise [14]. It is used in both the decoder and encoder.

V. EXPERIMENTAL SET UP

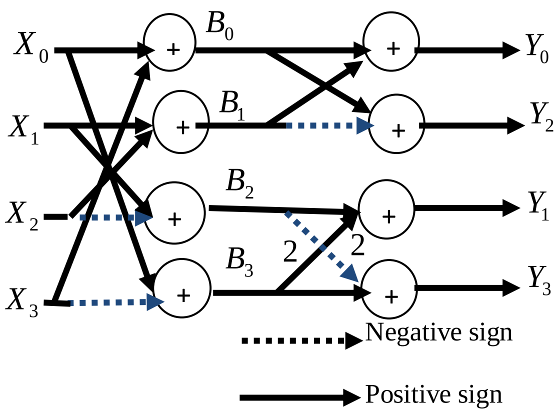
To investigate the efficiency and merits of the proposed optimization methods, we used one personal computer consisting of Intel Pentium IV processor. It is a dual core with 1.98 GM memory. Microsoft Visual studio 2008 studio equipped with a C++ compiler was installed as programming software. Three standard test video sequences, each of 300 frames (CIF 352x288): ***Foreman***, ***Soccer*** and ***Mobile & calendar*** were used as input to our experiments. We then determined the computation time speed-up for each of the video sequence while keeping PSNR constant in both methods. The speed up S_{up} was calculated from:

$$S_{up} = \frac{T_{reference}}{T_{optimised}} \qquad (8)$$

Where: $T_{reference}$ and $T_{optimised}$ are the average computation times in milliseconds for the reference and optimized codes or methods respectively in which thirty (30) experimental trials have been done for each video sequence in each method.

The 2-D matrix multiplication and 1-D butterfly algorithm method were implemented for the forward and inverse integer DCT transforms according to [15] [16]. In the forward integer DCT matrix multiplication method, the core transform matrix C_f was first multiplied with the input matrix X . The resultant was again multiplied by the transposed transform matrix C_f^T as derived in equation (2). For the inverse integer DCT the core matrix of equation (3) was multiplied in a similar manner as the forward integer DCT.As the input matrix was declared as **short** integer data type, the transform matrix was equally declared as a short data type. However, since the inverse transform matrix and its transpose consists of **floats** (0.5), they were declared as **double** data types.

On the other hand, in the 1-D butterfly method, we avoid use of the transform matrices and their transpose and yet achieving the same signal quality as in the matrix multiplication method. The method ideally, concentrate on the input matrix block (X) in which its rows and columns are added, subtracted and right or left shifted. Two stages are followed in the butterfly method: 1-D forward transform in the horizontal (along the rows) direction and another in the vertical direction (down the columns). This technique is illustrated in Figure 2(a) and (b) for the forward integer DCT and a similar approach was adopted for the inverse integer DCT. The merits of this method are that it avoids the complicated multiplication process as it uses only additions, subtractions and shift operations and avoids the use of floats or doubles and therefore rounding errors as it deals with integer only.



(a)

Stage 1 stage 2

$$B_0 = X_0 + X_3 \qquad Y_0 = B_0 + B_1$$
$$B_1 = X_1 + X_2 \qquad Y_2 = B_0 - B_1$$
$$B_2 = X_1 - X_2 \qquad Y_1 = B_2 + B_3 << 1$$
$$B_3 = X_0 - X_3 \qquad Y_3 = B_3 - B_2 << 1$$

(b)

Figure 2: (a) Forward integer DCT butterfly algorithm (b) Pseudo codes for the forward integer DCT [3] [6]

The Intel SIMD intrinsic instructions can be used to add, subtract, compare, set, shift, transpose and store data in parallel [8] [9] [17]. The Pentium IV architecture supports 128-bit floating point using SSE, 128-bit integer using SSE2 and 64-bit integer data types using MMX technology. In our experiments we needed to add, subtract or shift four 16-bit pixel define as short integer data type. We, therefore, chose the MMX Technology instruction which can load four *shorts* without any overheads.

To achieve a complete forward integer DCT using MMX SIMD instructions, four main stages as illustrated in Figure 3 were followed according to [6] [18]. Since the coefficients are kept in one packed word in one register, the 4x4 matrix is first transposed before applying the butterfly algorithm [18].

a) ***Loading and Transposing*** the 4x4 matrix of the input residual video signal. This is achieved by making use of four MMX conversion instructions [17]:
_mm_unpacklo_pi16(row1,row2]),
_mm_unpacklo_pi32(row0,row1),
_mm_unpackhi_pi16(row1,row2]),
_mm_unpackhi_pi32(row0,row1)

b) ***Butterfly a****lgorithm* to the transposed input video signal in b), using MMX instructions:
_mm_slli_pi16(row3, 1) ,
_mm_add_pi16 (row0, row3) and
_mm_sub_pi16 (row0, row1)

c) ***Transpose*** of the resulting matrix in b) as in a)

d) ***Butterfly and store*** algorithm to the matrix in c) as in b). Storage of pixels is combined within the butterfly process

The reverse of the above steps can be implemented for the inverse integer DCT transform

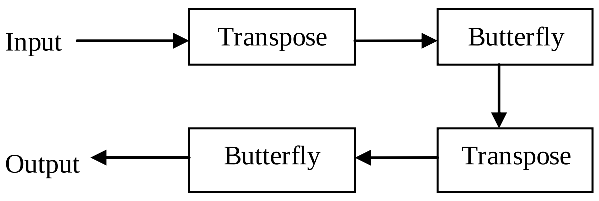


Figure 3: SIMD MMX implementation of the forward integer DCT butterfly method

In the second butterfly experiments, we converted the SIMD MMX intrinsic codes to the SIMD assembly MMX Technology instructions. The complete forward integer transform process is same as Figure 3: Loading of input block of pixel into MMX registers (using *movq* SIMD instructions), transposing of the loaded block of pixel (using SIMD instructions *punpckldqd, punpckhdqd, punpcklwd, punpckhwd),* butterfly algorithm to the transposed block (using *paddw, psubw, psllw* SIMDM instructions). The transpose and butterfly processes are repeated followed by the storage of pixels in MMX register. The *movq* instructions used in loading is also used for storing of pixels in registers. By substituting the *movq mm0, mm1*, with a *pshufw mm0, mm1*, 0xE4 instructions, we can speed up the **loading** of pixels in the MMX registers. The *pshufw* instruction is three times faster than the *movq* instruction [19].

Again the reverse of the above steps can be implemented for the inverse integer DCT transform.

VI. RESULTS AND DISCUSSION

Table 1 shows the results of the computation time in milliseconds taken to encode and decode each of the three standard video sequences.

The 1-D butterfly method with SIMD intrinsic takes in excess of 15000 ms less time to encode and decode each of the video sequences representing a 1.7x speed-up when compared to the 2-D matrix multiplication method. This speed-up is good but can be improved further. A closer check at the assembly generated by the intrinsic reveals the following weakness: firstly, of the eight (8) MMX registers mm0 to mm7 only two (2) are used. This means that in spite of the advantage of small code size; the intrinsic codes do not make efficient use of the registers. Secondly, the input matrix needs to be changed to __m64 data type by using type cast operation. The type cast operation is very expensive and tends to slow the codes.

To mitigate the above two weakness, we converted the intrinsic into SIMD assembly. This approach brings in the following advantages: No need of type casting, efficient use of MMX registers and Inherent assembly language speed up advantages

All measurement parameters being equal, an extra 2000ms representing a 1.9x speed-up was achieved when using SIMD assembly instructions.

The butterfly method achieved a high computational time speed-up largely due to low complexity introduced in the encoder. Less complex additions, subtractions and shifts were used instead of the more complicated multiplication obtained in the matrix multiplication method. Additionally, SIMD intrinsic and assembly instructions eliminates the use of iterative statement (loops) such as the '*for statement'* which tends to slow the codes. With SIMD instructions pixels are manipulated in parallel. Differences in computation speed up and PSNR between sequences are largely due to varying sequence details, bitrate and varying video motion in each sequence. However all these factors being the same the 1-D butterfly method outperforms the two dimension matrix multiplication method.

TABLE 1: AVERAGE COMPUTATION TIME FOR THE FORWARD AND INVERSE INTGER DCT PAIRS

Standard video sequences	Average computation time in milliseconds(ms)		
	Matrix multiplication	Butterfly algorithm	
	C++ only	SIMD Intrinsics	SIMD Assembly
Foreman	37534	21948	18956
Soccer	36187	20863	18196
Mobile & Calendar	42972	26059	24038
Speed up	1 x	1.7 x	1.9 x

VII. CONCLUSION

This paper has given the computation time speed-up improvement of 4x4 Integer DCT in a H.264 video encoder. The 1-D butterfly algorithm method gave superior speed gain compared to the 2-D matrix multiplication. We achieved an average 1-D butterfly SIMD computational time gains of 15000ms for the forward and inverse integer DCT pairs. This represents some speed-up of 1.7x to 1.9x in comparison to the matrix multiplication method. When the two butterfly results are compared there is a 10% time gain for the SIMD assembly results over the SIMD intrinsic. At these time gains and/or speed up the 1-D butterfly method can be used in real-time applications such as video broadcasting, internet, mobile communication and the like including low bandwidth environment. The 2- D method can also be used in applications were speed is less emphasized such as video compression for storage in Digital Versatile Disc (DVD), Digital Signal Processing (DSP) and for education purposes.

ACKNOWLEDGEMENT

This work is supported by the Council for Scientific and Industrial Research (CSIR) through the DST-innovation fund on project no. T70026: Bandwidth Adaptive Real-Time video broadcasting over internet.

REFERECES

[1] H.264: International Telecommunication Union, "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," *ITU-T*, 2003.

[2] I.E.G Richardson, "H.264 and MPEG-4 Video Compression", published by John Wiley and sons, West Sussex, UK, 2003

[3] H.S.Marver, A.Hallapuro, M.Karczewicz and L.kerofsky, "Low Complexity Transform and Quantisation in H.264/AVC" *IEEE Transactions on circuits and systems for video technology*, vol. 13, No 7, pp. 560-576, July, 2003.

[4] H. Kalva and J.B.Lee, "The VC-1 and H.264 Video Compression Standards for Broadband Video Services", Springer, New York, USA, 2008.

[5] I. E.G Richardson, (2009, April). *4x4 Transform and Quantization in H.264/AVC*. [On-Line]. Available: http://www.vcodex.com/h264transform4x4.html. [August 20, 2009].

[6] Y. Jianhong, L. Jilin, "Fast Parallel Implementation Of H.264/AVC Transform Exploiting SIMD Instructions" *International Symposium On Intelligent Signal Processing and Communication Systems*, November, 2007 pp. 870-873.

[7] Chil-Peng Fan, "Cost Effective Hardware Sharing Architectures of Fast 8x8 And 4x4 Integer Transforms For H.264/AVC" IEEE 2006.

[8] M. Hassaballah, Saleh Omran, Youssef B. Mahdy, "A Review of SIMD Multimedia Extensions and their Usage in Scientific and Engineering Applications" *The Computer Journal*, vol.51, issue 6, pp. 630-649, 2008.

[9] F. Franchetti, S. Kral, J. Lorenz, C. Ueberhuber, "Efficient Utilization of SIMD Extensions"*Proceedings of the IEEE*, Vol. 93, No. 2, pp 409-425, 2005

[10] C.P.Fan, "Fast 2-dimentional 4x4 forward integer transform implementation for H.264/AVC", *IEEE Transactions on circuits and systems II: Express briefs*, volume 53, Issue 3, pp. 174-177, March, 2006.

[11] Y. Muhammd K.E Khan, M.S Beg, "Performance Evaluation Of 4x4 DCT Algorithms for Low Power Wireless Applications" *First International Conference On Emerging Trends In Engineering and Technology*, 2008.

[12] Xueming Li, Fang WEI, "An Improved Practical Efficient Implementation of ICT Used in H.264", *IEEE International Conference on Multimedia*, 2004, pp. 1163-1166.

[13] Y. K. Chen, E. Q. Li, X.I. Zhou, Steven Ge, " Implementation of H.264 Encoder and Decoder On Personal Computers," *Journal of Visual Communication and Image Representation*, 2006, pp 509-532.

[14] Soon-kak Kwon, A. Tamhankar and K.R. Rao, "Overview of H.264/MPEG-4 part 10" *Journal of Visual Communication and Image Representation* ,Vol.17, Issue 2, pp. 186-216. 2006

[15] B. Stroustrup, "The C++ Programming Language", Addison-Wesley, special edition, New Jersey, USA

[16] S.Meyers," Effective C++:55 Ways to Improve Your Programs and Designs", 3rd edition, Addison-Wesley, New Jersey, USA.

[17] Intel®, (1999). *Architecture Optimization reference Manual*. [On-line], (245127-001). Available: http://download.intel.com/design/PentiumII/manuals/24512701.pdf [October 21, 2010]

[18] .J. Xiuhua, Z. Caiming, and W. Yanling , "Fast Algorithm of the 2-D 4x4 Inverse Integer Transform for H.264/AVC," *IEEE Innovative Computing, Information and Control International Conference(ICICIC)*, 2007, pp. 144-148.

[19] K. Nguyen, "Optimisation of the Intel Pentium 4 Processor using Assembly Languages" Intel Corporation, 20th October, 2008.

Charles Smart Lubobya received his undergraduate degree in 2009 from the Copperbelt University in Zambia and is currently studying towards his Master of Science degree at the University of Cape Town. His research interests include video compression and streaming, wireless communication and data networks.