

Matrix multiplication specification for HEVC transforms

(JCTVC-D036)

Mangesh Sadafale (*) and Madhukar Budagavi ()**

*** Texas Instruments India**

**** Texas Instruments Dallas**

**Joint Collaborative Team on Video Coding (JCT-VC)
of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**

4th Meeting: Daegu, KR, 20-28 January, 2011

Introduction

- Continuation of JCTVC-C266: Matrix multiplication DCT
- JCTVC-226 proposed specification of transform using matrix multiplication
 - 16x16 and 32x32 transform matrices were fixed-point approximations of DCT/IDCT
 - 4x4 and 8x8 transform matrices were retained from AVC
 - Quantization matrices used for norm correction were eliminated
- This contribution:
 - 16x16 and 32x32 transform matrices are 6-bit approximation of DCT/IDCT
 - DCT implemented using even-odd decomposition for achieving speed-up
 - Software released to Motorola and Qualcomm for cross-verification.
Thanks to Motorola and Qualcomm for cross-verifying.
- Related contribution:
 - JCTVC-D224 (Cisco)

Implementation complexity

- Large size transforms provide coding gain but increase implementation complexity significantly
- Implementation complexity of large size transforms needs to be studied carefully
 - Important to study both *hardware* and software implementation complexity
- Hardware codecs are expected to play an increasingly important role in deployment of HEVC solutions since HEVC is expected to be used for high definition (HD) and above video resolutions
 - Need for HD has already led to hardware acceleration being used for AVC video coding in desktop, mobile, and portable devices (which are traditionally thought of as software implementation platforms)
- In software, HEVC codecs can be expected to run on processors that support extensive SIMD operations
 - Already, 8-way SIMD architectures are becoming commonplace

IDCT specification

```
MAX_TSIZE = 32; // Maximum transform size
IDCTMatrix is of size [MAX_TSIZE][MAX_TSIZE]; // cos() values of DCT
TransposeBuffer is of size [MAX_TSIZE][MAX_TSIZE];
uiDctOffset = MAX_TSIZE/uiSize; // subsampling factor for DCTMatrix
pSrc is input data, pDst is output data
uiSize is transform block size
```

```
// D'*Input
for(i=0;i<uiSize;i++), for(j=0;j<uiSize;j++)
    sum = 0;
    for(k=0;k<uiSize;k++)
        sum += IDCTMatrix[k*uiDctOffset][i] * pSrc[k*uiSize+j];
    TransposeBuffer[i][j] = sum;
```

IDCTMatrix[32][32]
gets reused for all
DCT sizes

```
// (D'*Input)*D
for(i=0;i<uiSize;i++), for(j=0;j<uiSize;j++)
    sum = 0;
    for(k=0;k<uiSize;k++)
        sum += TransposeBuffer[i][k] * IDCTMatrix[k*uiDctOffset][j];
    sum = sum*uiDctScale;
    pDst[i*uiStride+j] = sum;
```

Quantization

- Existing HM 1.0
 - Quantization of 32x32 block: `UInt g_aiQuantCoef1024[6][1024];`
 - Inverse quantization of 32x32 block: `UInt g_aiDeQuantCoef1024[6][1024];`
 - Quantization of 16x16 block: `UInt g_aiQuantCoef256[6][256];`
 - Inverse quantization of 16x16 block: `UInt g_aiDeQuantCoef256[6][256];`
- Our proposal:
 - Quantization of 16x16/32x32 block:
 - `UInt g_aiQuantCoef256_s[6] = {205,186, 158, 146, 128, 114};`
 - `UInt g_aiQuantCoef1024_s[6] = {102, 93, 79, 73 ,64, 57};`
 - Inverse quantization of 16x16/32x32 block:
 - `UInt g_aiDeQuantCoef[6] = {10, 11, 13, 14, 16, 18};`
- Memory required for Quant/dequant matrices goes down from 12.5KB to 6-12 bytes

Intra

	Intra			Intra LoCo		
	Y BD-rate	U BD-rate	V BD-rate	Y BD-rate	U BD-rate	V BD-rate
Class A	0.0	-0.1	-0.2	-0.1	0.5	0.5
Class B	0.0	0.0	-0.1	0.0	0.0	0.1
Class C	0.0	0.0	-0.1	0.0	0.1	0.1
Class D	0.0	0.0	0.0	0.0	0.1	0.1
Class E	-0.1	0.0	-0.2	0.0	0.3	0.3
All	0.0	0.0	-0.1	0.0	0.1	0.2
Enc Time[%]	100%			107%		
Dec Time[%]	100%			99%		

Encoder simulations were run on Linux cluster. Decoding of bitstreams was carried out sequentially on a PC with Intel(R) Core(TM)2 CPU 6600 @ 2.4GHz, 3.25 GB of RAM.

Random access

	Random access			Random access LoCo		
	Y BD-rate	U BD-rate	V BD-rate	Y BD-rate	U BD-rate	V BD-rate
Class A	0.0	0.1	0.0	-0.1	0.5	0.3
Class B	-0.1	-0.1	0.0	-0.1	0.2	0.0
Class C	0.0	-0.2	-0.1	0.0	0.3	0.2
Class D	0.0	0.1	0.2	0.0	-0.2	0.3
Class E						
All	0.0	0.0	0.0	-0.1	0.1	0.1
Enc Time[%]	95%			96%		
Dec Time[%]	100%			100%		

Encoder simulations were run on Linux cluster. Decoding of bitstreams was carried out sequentially on a PC with Intel(R) Core(TM)2 CPU 6600 @ 2.4GHz, 3.25 GB of RAM.

Low delay

	Low delay			Low delay LoCo		
	Y BD-rate	U BD-rate	V BD-rate	Y BD-rate	U BD-rate	V BD-rate
Class A						
Class B	0.0	0.1	-0.3	-0.1	0.2	0.0
Class C	0.0	0.0	0.0	0.0	0.0	0.2
Class D	0.1	-0.3	-0.1	-0.1	0.1	-0.1
Class E	-0.2	-0.1	-0.6	-0.3	-0.6	0.1
All	0.0	0.0	-0.3	-0.1	0.0	0.0
Enc Time[%]	93%			96%		
Dec Time[%]	100%			100%		

Encoder simulations were run on Linux cluster. Decoding of bitstreams was carried out sequentially on a PC with Intel(R) Core(TM)2 CPU 6600 @ 2.4GHz, 3.25 GB of RAM.

Complexity analysis

- Multiplication/accumulation bit-width sizes
- Buffer sizes
- Throughput
- Total number of multiplications and additions
- Area

Complexity analysis - Multiplication/accumulation bit-width sizes

	IBDI Off			
	Chen's		Matmult	
	Input of Inverse transform	Intermediate values after first transform stage	Input of Inverse transform	Intermediate values after first transform stage
16x16	21 bits	24 bits	15 bits	15 bits
32x32	20 bits	24 bits	15 bits	15 bits
	IBDI On			
	Chen's		Matmult	
	Input of Inverse transform	Intermediate values after first transform stage	Input of Inverse transform	Intermediate values after first transform stage
16x16	25 bits	24 bits	19 bits	19 bits
32x32	24 bits	24 bits	19 bits	19 bits

Complexity analysis – Buffer sizes

- Hardware implementation complexity for IDCT
 - Compute logic (~50%) + Transpose buffer (~50%)
- Reduction in transpose buffer size leads to direct area savings
- Transpose buffer element size for 16x16, 32x32 transform for IBDI-off
 - HM 1.0: 21 bits
 - Matrix mult: 15 bits
 - ~30% savings in area for transpose buffer in hardware
 - Similar savings for IBDI-on
- For IBDI-off in software, number of cycles for fetching intermediate data goes down by a factor of 2
 - 32b data fetch for Chen v/s 16 bit data fetch for matrix multiplication

Complexity analysis - Throughput

- Chen's algorithm uses multiple stages of butterfly-type of structure
 - Introduces serial dependency and leads to multipliers getting cascaded one after the other
 - Leads to increased delay in hardware implementation and limits the maximum frequency at which the IDCT block can be run

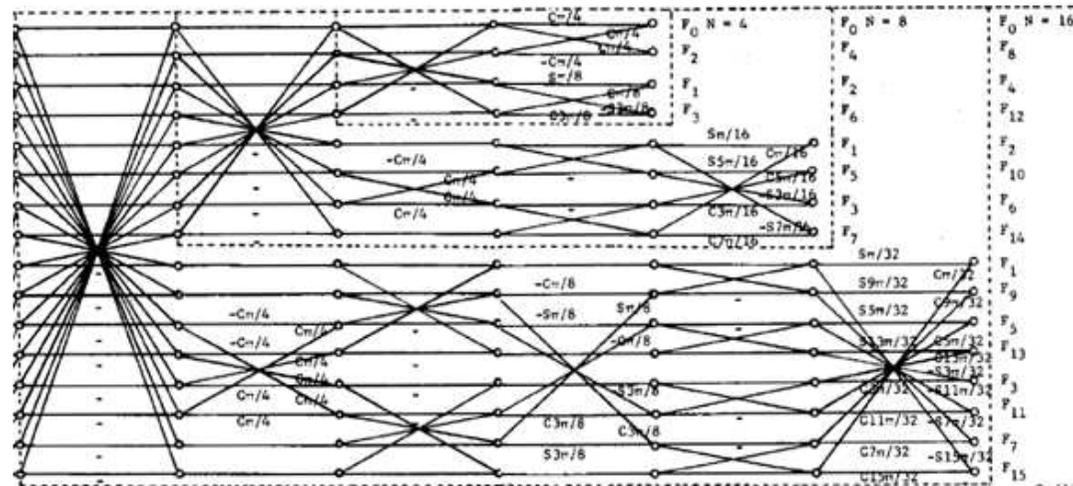


Figure from Ref [1]

	TMuC-0.9 Chen's IDCT	Matrix multiplication IDCT
Number of cascaded multipliers	4 (for 32x32) 3 for (16x16)	0 for all
Max frequency in Low-power 45nm (with NO Pipeline)	100MHz	275MHz

1/22/2

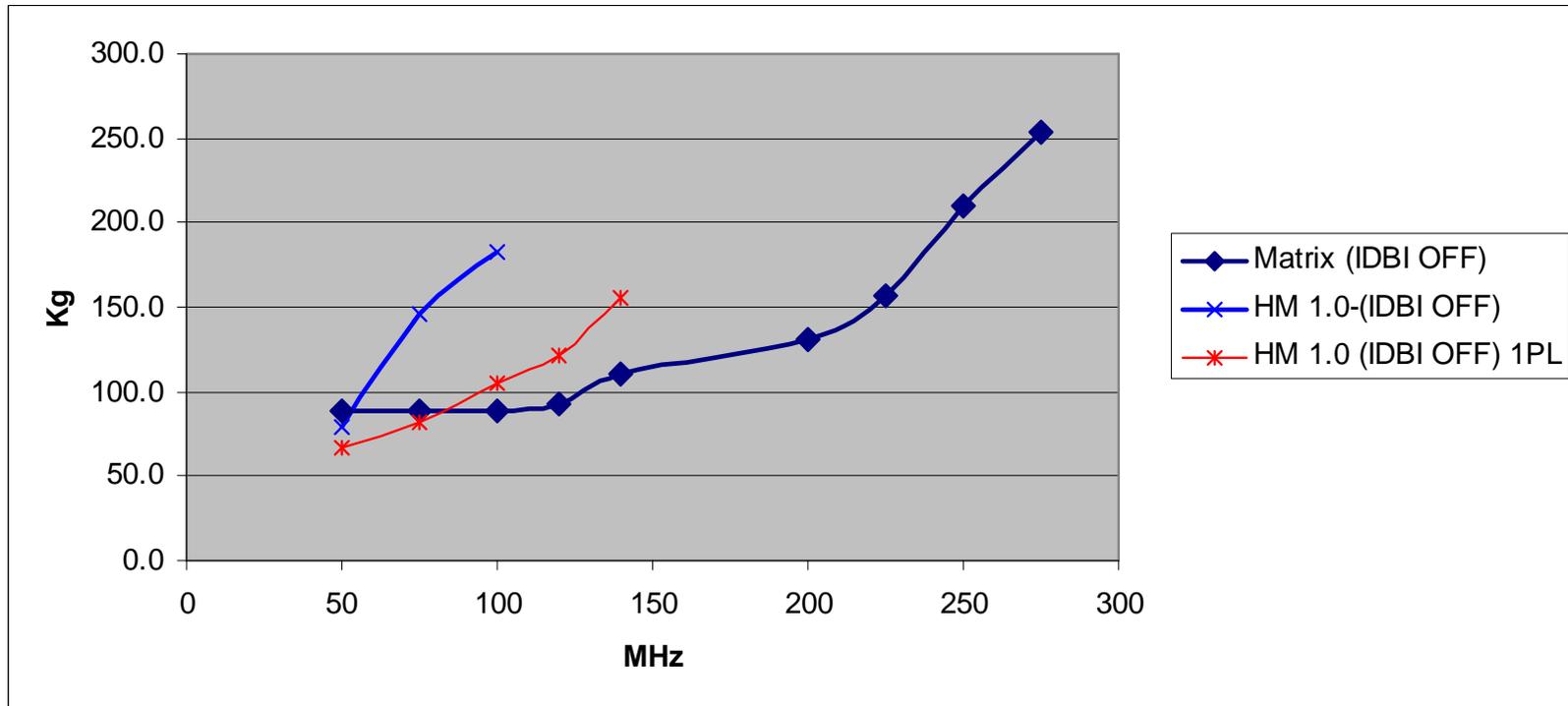
12

Number of multiplications and additions

	1D 32-point transform			1D 16-point transform		
	HM 1.0	MatMult in SW	MatMult in HW	HM 1.0	MatMult in SW	MatMult in HW
MULTs	116	342	300	44	82	-
ADDs	210	404	424	82	116	-

- Bitwidth impacts cycles in software e.g. 32-bit multiplication v/s 16-bit multiplication

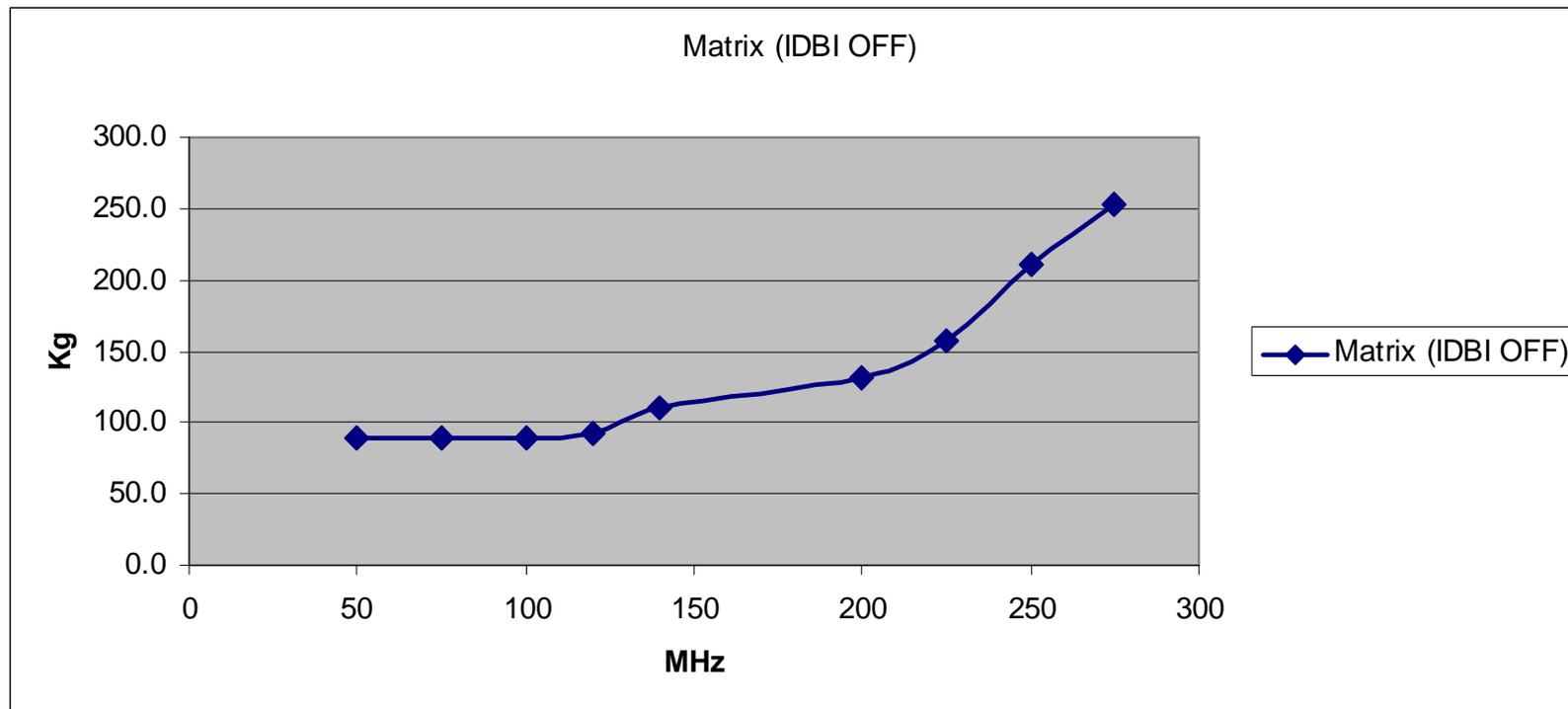
Complexity analysis – Hardware Area



- Area
 - Example area numbers for 1D 32x32 transform implementation in RTL

Hardware considerations - Transistor sizing

- For area: Number of computations matter but number of cascaded operations (critical path delay) in circuit matters too
- The more the number of cascaded operations (critical path delay) one needs to finish in a cycle time, the bigger the transistors become



1/22/2011

15

Conclusions

- Important to study both *hardware* and software implementation complexity
 - Something to think about: Are “Fast” DCTs really fast on today’s architecture?
- Contribution proposes specifying transforms using matrix multiplication
 - Also achieves HEVC standard text compression 😊
- Contribution eliminates norm correction quantization matrices
- Our approach is highly parallel, good fixed-point behavior
 - Good for hardware implementation
 - Good for SIMD and VLIW architecture
- Recommend adoption of matrix multiplication transform specification and scalar quantization into HM 2.0

DCT implementation by direct matrix multiplication

$$\begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} c_4 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\ c_4 & c_3 & c_6 & -c_7 & -c_4 & -c_1 & -c_2 & -c_5 \\ c_4 & c_5 & -c_6 & -c_1 & -c_4 & c_7 & c_2 & c_3 \\ c_4 & c_7 & -c_2 & -c_5 & c_4 & c_3 & -c_6 & -c_1 \\ c_4 & -c_7 & -c_2 & c_5 & c_4 & -c_3 & -c_6 & c_1 \\ c_4 & -c_5 & -c_6 & c_1 & -c_4 & -c_7 & c_2 & -c_3 \\ c_4 & -c_3 & c_6 & c_7 & -c_4 & c_1 & -c_2 & c_5 \\ c_4 & -c_1 & c_2 & -c_3 & c_4 & -c_5 & c_6 & -c_7 \end{pmatrix} \begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{pmatrix},$$

DCT implementation using full even-odd decomposition

$$\begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} c_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_2 & c_6 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_6 & -c_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_3 & c_5 & c_7 \\ 0 & 0 & 0 & 0 & c_3 & -c_7 & -c_1 & -c_5 \\ 0 & 0 & 0 & 0 & c_5 & -c_1 & c_7 & c_3 \\ 0 & 0 & 0 & 0 & c_7 & -c_5 & c_3 & -c_1 \end{pmatrix} \begin{pmatrix} X(0) \\ X(4) \\ X(2) \\ X(6) \\ X(1) \\ X(3) \\ X(5) \\ X(7) \end{pmatrix}$$