

Low-complexity, configurable transform architecture for HEVC (JCTVC-C226/m18266)

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
3rd Meeting: Guangzhou, CN, 7-15 October, 2010**

Motivation (1)

- TMuC transforms types
 - Large size DCT (16x16, 32x32, 64x64) implemented using Chen's algorithm
 - 4x4 and 8x8 integer AVC "DCT"
 - Directional transforms (ROT and MDDT) implemented using Matrix multiplication
- 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

Motivation (2)

- 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
- Hence transforms and architectures that are efficient to implement in both *hardware* and in software on SIMD machines are needed

Proposal

- We propose matrix multiplication based architecture for transform implementation
- Advantages of matrix multiplication architecture
 - Friendly to parallel processing with minimal dependency and control logic
 - Can be efficiently implemented on SIMD processors
 - Results in high throughput architecture in hardware
 - Results in low-area implementation in hardware since architecture is configurable and can be re-used across various DCT transform block sizes for HEVC
 - Good fixed-point behavior
 - No need to use existing quant/dequant matrices in TMuC
 - Memory requirement for storing dequantization matrices in the TMuC decoder goes down from 7.5 KB to 12 bytes in both encoder and decoder
 - Unifying architecture in the sense that it is flexible enough to support other transforms being considered in HEVC
- Focus of this contribution is on DCT/IDCT implementation

10/12/2010

4

Evaluation of coding gain impact

- TMuC-0.7.3 used as reference
- JCTVC-B300 common conditions
- Two main modifications done to software
 - Direct matrix multiplication implementation of DCT/IDCT
 - Elimination of quant/dequant matrices

Direct matrix multiplication

```
MAX_TSIZE = 64; // Maximum transform size
DCTMatrix 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 += DCTMatrix[k*uiDctOffset][i] * pSrc[k*uiSize+j];
    TransposeBuffer[i][j] = sum;
```

DCTMatrix[64][64]
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] * DCTMatrix[k*uiDctOffset][j];
    sum = sum*uiDctScale;
    pDst[i*uiStride+j] = sum;
```

Quant/Dequant matrices elimination (1)

- Following are sizes of quant/dequant matrices used in TMuC:
 - 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];
 - Matrix entries vary slightly depending on position of transform coeff
- Presumably used in TMuC to offset fixed-point effects of Chen's DCT/IDCT
 - Leads to degradation of 0.1%-0.3% if scalar values are used instead
- Total memory used about **12.5 KB** in encoder and decoder

Quant/Dequant matrices elimination (2)

- Our implementation uses the following:
 - Quantization of 32x32 block: UInt g_aiQuantCoef1024_s [6];
 - Inverse quantization of 32x32 block: UInt g_aiDeQuantCoef1024_s [6];
 - Quantization of 16x16 block: UInt g_aiQuantCoef256_s [6];
 - Inverse quantization of 16x16 block: UInt g_aiDeQuantCoef256_s [6];
 - One entry per Qstep ($\text{mod}(\text{QP}, 6)$)
- Memory required for Quant/dequant matrices goes down from **12.5KB to 12 bytes** for both decoder and encoder

Coding gain results – 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	0.0	-0.1	-0.1	0.0	-0.1
Class B	0.0	0.0	0.0	0.0	0.0	0.0
Class C	0.0	0.0	0.0	0.0	0.0	0.0
Class D	0.0	0.0	0.0	0.0	0.0	0.0
Class E	0.0	0.0	0.0	0.0	-0.1	0.0
All	0.0	0.0	0.0	0.0	0.0	0.0

Coding gain results – 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	0.0	-0.1	0.0	0.0	0.0
Class B	0.0	-0.1	-0.1	0.0	0.1	0.0
Class C	0.0	-0.1	0.0	0.0	0.0	0.0
Class D	0.0	0.0	0.1	0.0	0.1	-0.2
Class E						
All	0.0	-0.1	0.0	0.0	0.0	-0.1

Coding gain results – 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	0.1	0.0	0.0	-0.1
Class C	-0.1	-0.1	0.1	0.0	0.0	0.1
Class D	0.0	-0.1	-0.3	0.0	0.1	-0.1
Class E	0.0	0.3	0.3	0.0	-0.1	0.3
All	0.0	0.0	0.0	0.0	0.0	0.0

Complexity analysis

- Throughput
- Multiplication/accumulation bit-width sizes
- Buffer sizes
- Area

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

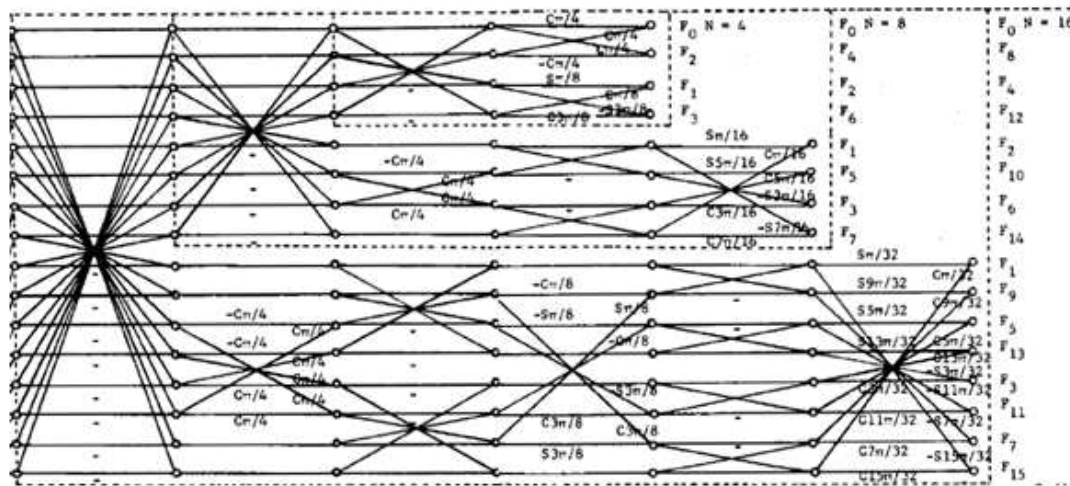


Figure from Ref [1]

	TMuC-0.7.3 Chen's IDCT	Matrix multiplication IDCT
Number of cascaded multipliers	5 for (64x64) 4 (for 32x32) 3 for (16x16)	0 for all
Max frequency in Low-power 45nm (with NO Pipeline)	115MHz	230MHz

Complexity analysis - Multiplication/accumulation bit-width sizes

- IBDI off

TU Size	Inverse transform input bit-width (after transform, quantization, inverse quantization)		Inverse transform transpose buffer bit-width (after horizontal OR vertical inverse transform)	
	TMuC-0.7.3 Chen's IDCT	Matrix mult	TMuC-0.7.3 Chen's IDCT	Matrix mult
64x64	19	16	19	16
32x32	20	16	20	16
16x16	20	16	20	16
8x8	15	16	16	16
4x4	15	16	17	16

- Software implication (IBDI off)

	TMuC-0.7.3 Chen's IDCT	Matrix multiplication IDCT
CPU multiplier	32-bit instruction	16-bit instruction
CPU adder/sub	64-bit instruction	32-bit instruction

10/12/2010

14

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
 - TMuC-0.7.3: 20 bits
 - Matrix mult: 16 bits
 - 20% savings in area for transpose buffer in hardware
 - Similar savings for IBDI-on
- For IBDI-off in software, number of cycles for fetching data goes down by a factor of 2 (32b data fetch for Chen v/s 16 bit data fetch for matrix multiplication))

Complexity analysis – Hardware sharing and Area

- Hardware sharing

	TMuC-0.7.3 Chen's IDCT	Matrix multiplication IDCT
HW Sharing between different transform sizes	Limited as multiplier size and constants are different	High degree of reuse

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

	TMuC-0.7.3 Chen's IDCT	Matrix multiplication IDCT
Area @ 115MHz	1X	0.8X

- 20% savings in area

Conclusions

- Important to study both *hardware* and software implementation complexity
 - Something to think about: Are “Fast” DCTs really fast on today’s architecture?
- Matrix multiplication is an attractive architecture, propose that matrix multiplication be made the default DCT/IDCT implementation in TM after cross-verification
 - Future optimizations if any can be built on top of it
 - Request core experiment be started on matrix multiplication DCT/IDCT
- Propose that we remove quantization matrices in TM
- Request AhG on Efficient implementation of IDCT in hardware and software