

JCTVC-F446

Core transform design for HEVC

Arild Fuldseth **Cisco**

Gisle Bjøntegaard **Cisco**

Mangesh Sadafale **TI**

Madhukar Budagavi **TI**

Summary

- Unified transform design (4x4,...,32x32)
- Adopted in HM3.0 software (JCVC-E243)
- 16-bit intermediate storage
- 16-bit multipliers for internal processing
- No quantization/dequantization matrices
- Matrix multiplication or partial butterfly implementation
- BDR-gain over HM2.0 transforms verified in HM3.0
- Performance results of optimized implementations

Unified design

- Unified design for all transform sizes
- Reuse of logic for smaller transform sizes

Examples

4x4 transform:

{64, 64, 64, 64}
{83, 36, -36, -83}
{64, -64, -64, 64}
{36, -83, 83, -36}

8x8 transform:

{64, 64, 64, 64, 64, 64, 64, 64}
{89, 75, 50, 18, -18, -50, -75, -89}
{83, 36, -36, -83, -83, -36, 36, 83}
{75, -18, -89, -50, 50, 89, 18, -75}
{64, -64, -64, 64, 64, -64, -64, 64}
{50, -89, 18, 75, -75, -18, 89, -50}
{36, -83, 83, -36, -36, 83, -83, 36}
{18, -50, 75, -89, 89, -75, 50, -18}

Basis vectors

- Close to DCT
- Symmetry/anti-symmetry properties of DCT
- Almost orthogonal
- Almost equal norm
- This provides:
 - Simplified quantization/dequantization
 - No quantization/dequantization matrices for norm equalization

Matrix multiplication or "butterfly"

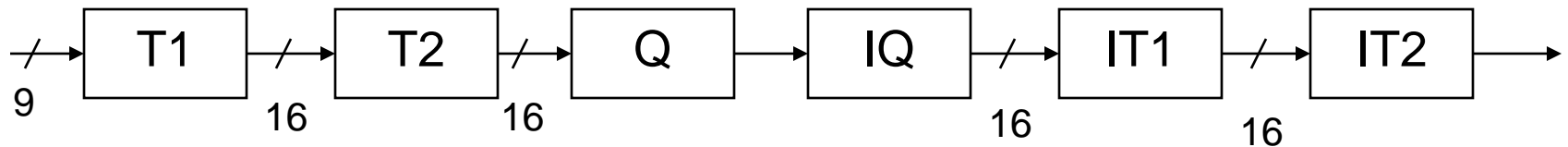
- Identical results
- Matrix multiplication:
 - Straightforward
 - Few code lines
 - Very SIMD friendly
- Partial "butterfly" implementation:
 - Utilizes symmetry/anisymmetry properties of basis vectors
 - Less multiplications/additions
 - Increased number of codelines

Matrix multiplication

```
for (i=0; i<uiTrSize; i++)
{
    for (j=0; j<uiPartialSize; j++)
    {
        iSum = 0;
        for (k=0; k<uiPartialSize; k++)
        {
            iSum += iIT[uiTrSize*k+i]*plCoef[k*uiTrSize+j];
        }
        tmp[i][j] = (iSum+32)>>6;
    }
}
for (i=0; i<uiTrSize; i++)
{
    for (j=0; j<uiTrSize; j++)
    {
        iSum = 0;
        for (k=0; k<uiPartialSize; k++)
        {
            iSum += iIT[uiTrSize*k+j]*tmp[i][k];
        }
        pResidual[i*uiStride+j] = (iSum+4096)>>13;
    }
}
```

16 bit intermediate storage

- 16 bit bit depth of all transform buffers,
- Independent of internal bit depth (8, 10, or 12 bit)



Internal data representation

$$a += c * d$$

- Matrix coefficients (c): 8 bit
- Data (d): 16 bit
- Accumulator (a): 32 bit

SIMD efficiency in software

Transform	Multipliers bit width	Bytes/multiplier	SIMD efficiency
Proposed (HM3.0)	16 bit	2	N/2
HM2.0	16+ bit	4	N/4

Intermediate scaling

- Forward transform:
 - First stage: $\gg (\log_2(N)-1)$
 - Second stage: $\gg (\log_2(N)+6)$
- Inverse transform:
 - First stage: $\gg 7$ (6)
 - Second stage: $\gg 12$ (13)

Quantization/dequantization

- Quantization:

$$\text{level} = (\text{coeff} * Q + \text{offset}) \gg (21 + QP/6 - \log_2(N))$$

- Dequantization:

$$\text{coeffQ} = (\text{level} * IQ \ll (QP/6) + N/4) \gg (\log_2(N)) - 1$$

$$\text{coeffQ} = \max(-32768, \min(32767, \text{coeffQ}))$$

- Same scheme for all transform sizes, N.
- No quantization matrices needed.
- Q and IQ are equal for all transform coefficients.

Decoder dynamic range control

- Assumption:
 - 16 bit dynamic range needs to be guaranteed
- After dequantizer:
 - Clipping to 16 bit
- After first stage of inverse transform:
 - Extra right shift – for reasonable quantizers,
or
 - Clipping to 16 bit – for randomized quantizers

Summary

Feature	HM2.0	Proposed (HM3.0)
Unified design	No	Yes
Intermediate bit width	>20	16
Multiplier bit width (software)	32	16
Equal norm of basis vectors	No	Almost
(De)quantization matrices	Yes	No
Matrix multiplication possible	No	Yes
TPE/IBDI value in simulations	2	0
Unified quantization scheme	No	Yes
Butterfly	Full	Partial

BD-rate results

- Simulation conditions:
 - Anchor: HM3.0 core transforms
 - Test: HM2.0 core transforms

	High efficiency			Low complexity		
	I	RA	LC	I	RA	LC
High QP (36,42,47,51)	0,2%	0,1%	0,2%	0,3%	0,1%	0,1%
Normal QP (22,27,32,37)	0,1%	0,0%	0,1%	0,2%	0,1%	0,0%
Low QP (1, 5, 9,13)	1,2%	0,9%	0,9%	2,0%	1,2%	1,4%

Optimized implementation

- Stand-alone test program with inverse transforms
- Optimized implementation for Intel Sandy Bridge
- Combined with transform usage statistics from HM3.0
- Estimated CPU load from inverse transforms alone

CPU load for Intel 3.5 GHz

Sequence	Bitrate (kbps)	CPU load	Bitrate (kbps)	CPU load
	QP=37		QP=22	
Kimono	560	0,6%	5325	2,5%
ParkScene	590	0,3%	8221	1,1%
Cactus	1308	0,8%	19693	3,9%
BasketballDrive	1572	1,2%	20049	5,6%
BQTerrace	799	0,4%	50211	4,6%

Conclusion

- Unified transform design for HEVC
- Several advantages for efficient implementation
- Small but consistent gains over HM2.0 transforms
- CPU load for 1080p decoding: 0.3% - 5,6%
- Proposal: To consider for adoption in WD

Partial butterfly – inverse (8x8)

```
for (j=0; j<8; j++){
    O[0] = c[1][0]*coef[8*j+1] + c[3][0]*coef[8*j+3] + c[5][0]*coef[8*j+5] + c[7][0]*coef[8*j+7];
    O[1] = c[1][1]*coef[8*j+1] + c[3][1]*coef[8*j+3] + c[5][1]*coef[8*j+5] + c[7][1]*coef[8*j+7];
    O[2] = c[1][2]*coef[8*j+1] + c[3][2]*coef[8*j+3] + c[5][2]*coef[8*j+5] + c[7][2]*coef[8*j+7];
    O[3] = c[1][3]*coef[8*j+1] + c[3][3]*coef[8*j+3] + c[5][3]*coef[8*j+5] + c[7][3]*coef[8*j+7];

    EE[0] = c[0][0]*coef[8*j+0] + c[4][0]*coef[8*j+4];
    EO[0] = c[2][0]*coef[8*j+2] + c[6][0]*coef[8*j+6];
    EE[1] = c[0][1]*coef[8*j+0] + c[4][1]*coef[8*j+4];
    EO[1] = c[2][1]*coef[8*j+2] + c[6][1]*coef[8*j+6];

    E[0] = EE[0] + EO[0];
    E[1] = EE[1] + EO[1];
    E[2] = EE[1] -EO[1];
    E[3] = EE[0] -EO[0];

    tmp[0][j] = (E[0] + O[0] + offset)>>shift;
    tmp[1][j] = (E[1] + O[1] + offset)>>shift;
    tmp[2][j] = (E[2] + O[2] + offset)>>shift;
    tmp[3][j] = (E[3] + O[3] + offset)>>shift;
    tmp[4][j] = (E[3] -O[3] + offset)>>shift;
    tmp[5][j] = (E[2] -O[2] + offset)>>shift;
    tmp[6][j] = (E[1] -O[1] + offset)>>shift;
    tmp[7][j] = (E[0] -O[0] + offset)>>shift;
}
```