JCTVC-E243

Transform design for HEVC with 16 bit intermediate data representation

Arild Fuldseth	**Cisco**
Gisle Bjøntegaard	**Cisco**
Mangesh Sadafale	**TI**
Madhukar Budagavi	**TI**

Summary

- Unified transform design for HEVC (4x4,...,32x32)
- 16-bit intermediate storage
- 16-bit multipliers for internal processing
- Almost equal norm for all basis vectors
- Matrix multiplication or partial butterfly implementation
- BD-rate gains:
  - Normal and high QP range: 0.1% - 0.5%
  - Low QP range: 0.8% - 2.5%
- Proposed WD text is available

Unified design

- HM2.0 transforms:
  - 4x4 and 8x8: Same as AVC
  - 16x16 & 32x32: Based on Chen algorithm (DCT)

- Proposed transforms:
  - 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 partial butterfly

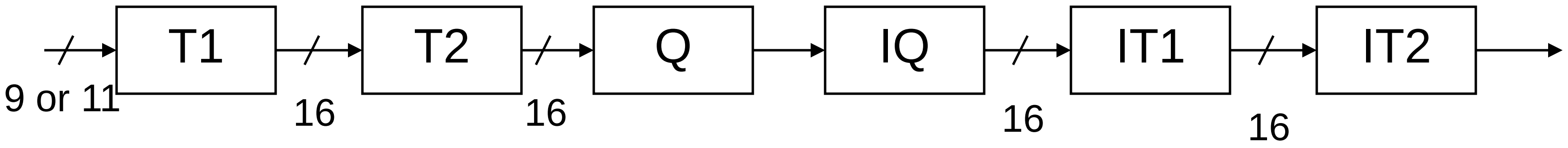
- Identical results
- Matrix multiplication:
  - Straightforward
  - Few code lines
  - Very SIMD friendly
- Partial ”butterfly” implementation:
  - Utilizes symmetry/anti-symmetry properties of basis vectors
  - Less multiplications/additions
  - Increased number of codelines

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

- HM2.0 transforms: >= 20 bit + 2 TPE/IBDI bits
- Proposed transforms: <= 16 bit



Internal data representation

$$a \mathrel{+}= c * d$$

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	16 bit	2	N/2
HM2.0	16+ bit	4	N/4

Intermediate scaling

- Forward NxN transform:
  - After first stage: $>>(\log_2(N)-1)$
  - After second stage: $>>(\log_2(N)+6)$

- Inverse NxN transform:
  - After first stage: >>7 (6)
  - After second stage: >>12 (13)

Quantization/dequantization

- Quantization:

  level = (coeff*Q + offset)>>(21 + QP/6 – log2(N))

- Dequantization:

  coeffQ = ((level*IQ << (QP/6)) + N/4)>>(log2(N)) -1)

  coeffQ = max(-32768, min(32767, 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 of transform properties

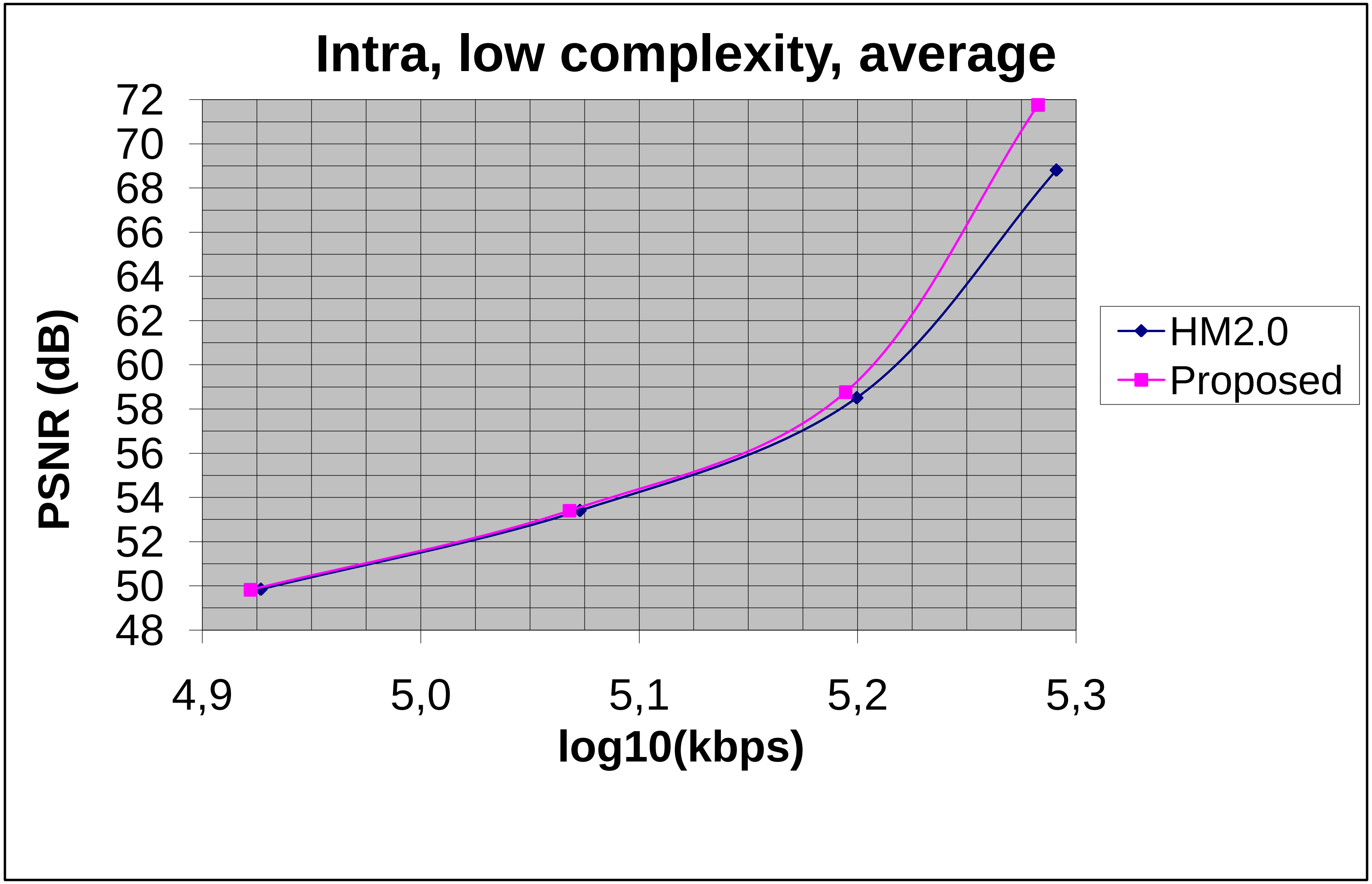
Feature	HM2.0	Proposed
Unified design	No	Yes
Intermediate bit width	>20	16
Multiplier bit width (software)	32	16
Equal norm of basis vectors	No	Almost
(De)quantization matrices	Yes	No
Matrix multiplication possible	No	Yes
TPE/IBDI value in simulations	2	0
Unified quantization scheme	No	Yes
Butterfly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

BD-rate results

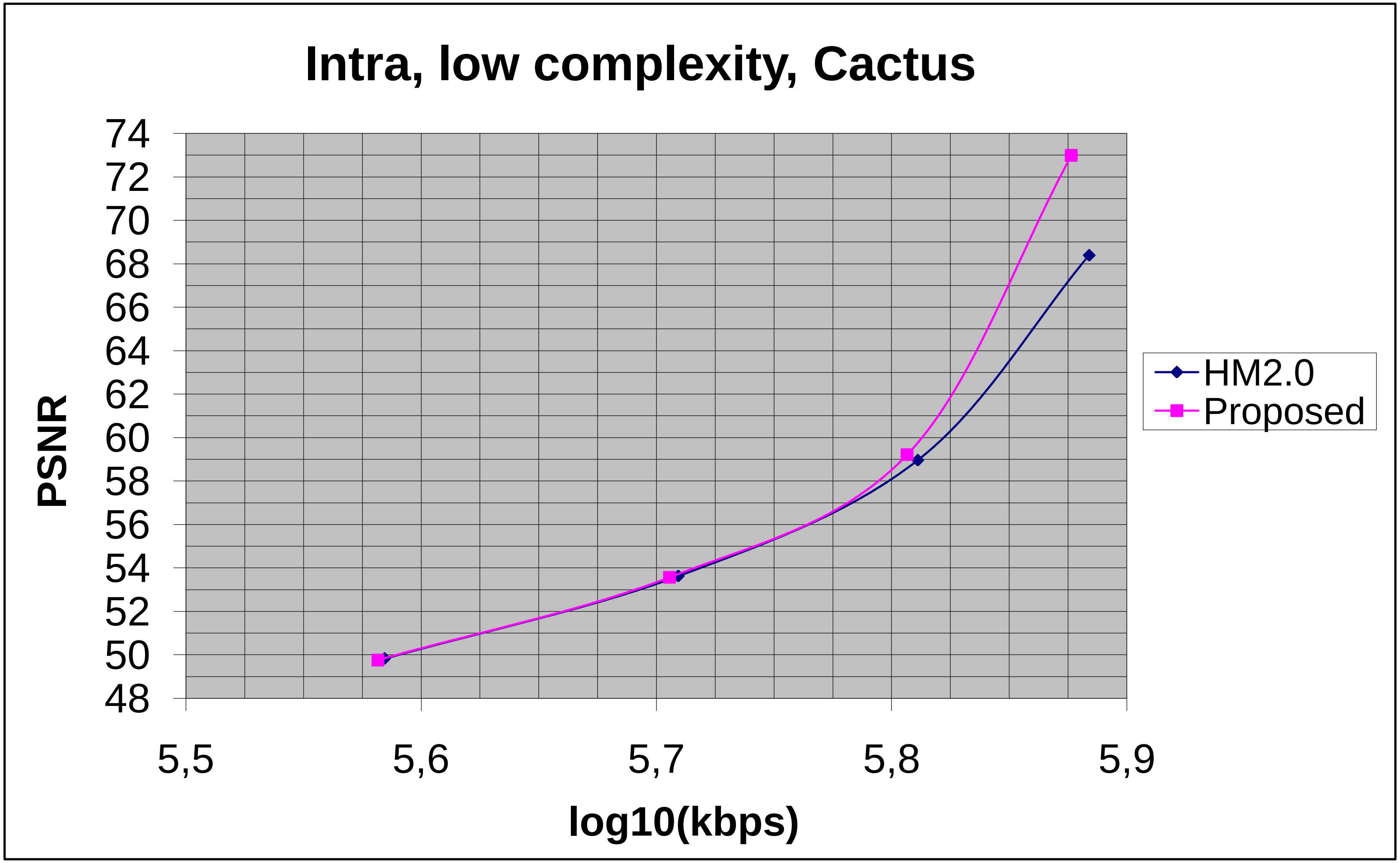
- Simulation conditions:
  - Anchor: TPE/IBDI=2
  - Proposed: TPE=0

BD-rate (%)	High Efficiency			Low Complexity		
	I	RA	LC	I	RA	LC
High QP (36,42,47,51)	-0,1	-0,2	-0,2	-0,1	-0,1	-0,2
Normal QP (22,27,32,37)	-0,3	-0,1	-0,2	-0,5	-0,3	-0,1
Low QP (1, 5, 9,13)	-1,1	-0,8	-0,9	-2,5	-1,6	-1,9

Performance at low QP values



Performance at low QP values



Conclusion

- Unified core transform design for HEVC
- Several advantages for efficient implementation
- Consistent BD-rate gain
- Proposal: To consider for adoption in HM

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;
}
```

Partial butterfly – forward 8x8

```
for (j=0; j<8; j++)
{
    O[0] = iRes[0] - iRes[7];
    O[1] = iRes[1] - iRes[6];
    O[2] = iRes[2] - iRes[5];
    O[3] = iRes[3] - iRes[4];

    E[0] = iRes[0] + iRes[7];
    E[1] = iRes[1] + iRes[6];
    E[2] = iRes[2] + iRes[5];
    E[3] = iRes[3] + iRes[4];

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

    tmp[0][j] = (c[0][0]*EE[0] + c[0][1]*EE[1] + offset)>>shift;
    tmp[4][j] = (c[4][0]*EE[0] + c[4][1]*EE[1] + offset)>>shift;

    tmp[2][j] = (c[2][0]*EO[0] + c[2][1]*EO[1] + offset)>>shift;
    tmp[6][j] = (c[6][0]*EO[0] + c[6][1]*EO[1] + offset)>>shift;

    tmp[1][j] = (c[1][0]*O[0] + c[1][1]*O[1] + c[1][2]*O[2] + c[1][3]*O[3] + offset)>>shift;
    tmp[3][j] = (c[3][0]*O[0] + c[3][1]*O[1] + c[3][2]*O[2] + c[3][3]*O[3] + offset)>>shift;
    tmp[5][j] = (c[5][0]*O[0] + c[5][1]*O[1] + c[5][2]*O[2] + c[5][3]*O[3] + offset)>>shift;
    tmp[7][j] = (c[7][0]*O[0] + c[7][1]*O[1] + c[7][2]*O[2] + c[7][3]*O[3] + offset)>>shift;
  }
```