**Complexity comparison between DMM4 and InterSDC with 2 segments**

1. **Block size**

We compares a 8x8 CU with 4 PUs coded as DMM4 *vs* a 8x8 CU coded as InterSDC with 2 segments.

1. **Operations**

We look into the number of additions, comparisons and multiplications. Shifts are not considered because (a+b+1)>>1 can be calculated as one instruction in many popular instruction sets such as MMX/SSE/NEON.

1. **Predictor padding**

As described in the WD, we have

(----------------------------------------------------

##### 8.4.4.2.1 General intra sample prediction

Inputs to this process are:

– a sample location ( xTbCmp, yTbCmp ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,

– a variable predModeIntra specifying the intra prediction mode,

– a variable nTbS specifying the transform block size,

– a variable cIdx specifying the colour component of the current block.

Output of this process is the predicted samples predSamples[ x ][ y ], with x, y = 0..nTbS − 1.

The nTbS \* 4 + 1 neighbouring samples p[ x ][ y ] that are constructed samples prior to the deblocking filter process, with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1, are derived as follows:

– The neighbouring location (xNbCmp, yNbCmp ) is specified by:

( xNbCmp, yNbCmp ) = ( xTbCmp + x, yTbCmp + y ) (8‑27)

* The current luma location ( xTbY, yTbY ) and the neighbouring luma location (xNbY, yNbY ) are derived as follows:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTbCmp, yTbCmp ) : ( xTbCmp  <<  1, yTbCmp  <<  1 ) (8‑28)

( xNbY, yNbY ) = ( cIdx  = =  0 ) ? ( xNbCmp, yNbCmp ) : ( xNbCmp  <<  1, yNbCmp  <<  1 ) (8‑29)

* The availability derivation process for a block in z-scan order as specified in subclause  is invoked with the current luma location ( xCurr, yCurr ) set equal to ( xTbY, yTbY ) and the neighbouring luma location ( xNbY, yNbY ) as inputs, and the output is assigned to availableN.

– Each sample p[ x ][ y ] is derived as follows:

* If one or more of the following conditions are true, the sample p[ x ][ y ] is marked as "not available for intra prediction":
  + The variable availableN is equal to FALSE.
  + CuPredMode[ xNbY ][ yNbY ] is not equal to MODE\_INTRA and constrained\_intra\_pred\_flag is equal to 1.
* Otherwise, the sample p[ x ][ y ] is marked as "available for intra prediction" and the sample at the location ( xNbCmp, yNbCmp ) is assigned to p[ x ][ y ].

When at least one sample p[ x ][ y ] with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1 is marked as "not available for intra prediction", the reference sample substitution process for intra sample prediction in subclause is invoked with the samples p[ x ][ y ] with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1, nTbS, and cIdx as inputs, and the modified samples p[ x ][ y ] with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1 as output.

--------------------------------------------------------------------------------------------)

So there should be 4 \* nTbS + 1 comparisons in the intra-predictor padding process. For a 4x4 block, the number is 17. In a 8x8 CU, the number is 17 x 4= 68

1. **Inter bi-prediction**

In the worst case, the 8x8 CU coded by Inter or segInter-SDC should apply the bi-prediction. 8x8 =64 additions are required.

1. **Segment threshold**

The four corner method is applied.

For DMM4, each PU needs 3 additions so there are 3 x4 = 12 additions for a 8x8 CU.

For InterSDC with 2 segments, 3 additions are used for a 8x8 CU.

1. **Classify**

For DMM4, each PU needs 4x4 =16 comparisons so there are 4 x 4 x 4 = 64 comparisons for a 8x8 CU.

For InterSDC with 2 segments, 8x8 = 64 comparisons are used for a 8x8 CU.

1. **Offset prediction**

As described in the WD, we have

(----------------------------------------------------

I.8.4.4.2.1.10 Depth partition value derivation and assignment process

Inputs to this process are:

* the neighbouring samples p[ x ][ y ], with x = −1, y = −1..nTbS \* 2 − 1 and x = 0..nTbS \* 2 − 1, y = −1,
* a binary array partitionPattern[ x ][ y ], with x, y =0..nTbS − 1, specifying a partitioning of the prediction block in a partition 0 and a partition 1,
* a sample location ( xTb, yTb ) specifying the top-left sample of the current block relative to the top‑left sample of the current picture,
* a variable nTbS specifying the transform block size.

Output of this process is:

* the predicted samples predSamples[ x ][ y ], with x, y = 0..nTbS − 1.

The variables vertEdgeFlag and horEdgeFlag are derived as specified in the following:

* 1. vertEdgeFlag = ( partitionPattern[ 0 ][ 0 ]  !=  partitionPattern[ nTbS − 1 ][ 0 ] ) (I‑61)
  2. horEdgeFlag = ( partitionPattern[ 0 ][ 0 ]  !=  partitionPattern[ 0 ][ nTbS − 1 ] ) (‑62)

The variables dcValBR and dcValLT are derived as specified in the following:

* If vertEdgeFlag is equal to horEdgeFlag, the following applies:
  + The variable dcValBR is derived as follows:
    - If horEdgeFlag is equal to 1, the following applies:
      * 1. dcValBR = ( ( p[ −1 ][ nTbS − 1 ] + p[ nTbS − 1 ][ −1 ] ) >> 1 ) (‑63)
    - Otherwise (horEdgeFlag is equal to 0), the following applies:
      * 1. vertAbsDiff = Abs( p[ −1 ][ 0 ] − p[ −1 ][ nTbS \* 2 − 1 ] ) (‑64)
        2. horAbsDiff = Abs( p[ 0 ][ −1 ] − p[ nTbS \* 2 − 1 ][ −1 ] ) (‑65)
        3. dcValBR = ( horAbsDiff > vertAbsDiff ) ? p[ nTbS \* 2 − 1 ][ −1 ] : p[ −1 ][ nTbS\*2 − 1 ] ) (‑66)
  + The variable dcValLT is derived as follows:

dcValLT = ( p[ −1 ][ 0 ] + p[ 0 ][ −1 ] ) >> 1 (‑67)

* Otherwise (horEdgeFlag is not equal to vertEdgeFlag), the following applies:

dcValBR = horEdgeFlag ? p[ −1 ][ nTbS − 1 ] : p[ nTbS − 1 ][ −1 ] (‑68)

dcValLT = horEdgeFlag ? p[ ( nTbS − 1 ) >> 1 ][ −1 ] : p[ −1 ][ ( nTbS − 1 ) >> 1 ] (‑69)

----------------------------------------------------)

So for a 4x4 block coded as DMM4, there should be 3 comparisons and 3 additions for a PU. That is 3x4 = 12 additions and 3x4=12 comparisons for a 8x8 CU totally.

For a 8x8 block coded as Inter-SDC with 2 segments, a sample need one addition and one comparison to find the majority value. So there are 8x8 = 64 additions and 8x8 = 64 comparisons.

1. **Scaling**

As described in the WD, we have

(----------------------------------------------------

### Scaling process for transform coefficients

Inputs to this process are:

– a luma location ( xTbY, yTbY ) specifying the top-left sample of the current luma transform block relative to the top‑left luma sample of the current picture,

– a variable nTbS specifying the size of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– a variable qP specifying the quantization parameter.

Output of this process is the (nTbS)x(nTbS) array d of scaled transform coefficients with elements d[ x ][ y ].

The variable bdShift is derived as follows:

– If cIdx is equal to 0,

bdShift = BitDepthY + Log2( nTbS ) − 5 (8‑270)

– Otherwise,

bdShift = BitDepthC + Log2( nTbS ) − 5 (8‑271)

The list levelScale[ ] is specified as levelScale[ k ] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.

For the derivation of the scaled transform coefficients d[ x ][ y ] with x = 0..nTbS − 1, y = 0..nTbS − 1, the following applies:

– The scaling factor m[ x ][ y ] is derived as follows:

– If scaling\_list\_enabled\_flag is equal to 0,

m[ x ][ y ] = 16 (8‑272)

– Otherwise (scaling\_list\_enabled\_flag is equal to 1),

m[ x ][ y ] = ScalingFactor[ sizeId ][ matrixId ][ x ][ y ] (8‑273)

Where sizeId is specified in Table 7‑3 for the size of the quantization matrix equal to (nTbS)x(nTbS) and matrixId is specified in Table 7‑4 for sizeId, CuPredMode[ xTbY ][ yTbY ], and cIdx, respectively.

– The scaled transform coefficient d[ x ][ y ] is derived as follows:

d[ x ][ y ] = Clip3( −32768, 32767, ( ( TransCoeffLevel[ xTbY ][ yTbY ][ cIdx ][ x ][ y ] \* m[ x ][ y ] \*   
 levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift )

----------------------------------------------------)

So for a 4x4 block coded as DMM4, there should be 16\*2 =32 muls for a PU. That is 32x4 = 128 muls for a 8x8 CU totally.

For a 8x8 block coded as Inter mode with TU size = 8x8, there should be 64 \* 2 = 128 muls.

1. **Transform**

As described in the WD, we have

(----------------------------------------------------

#### General

Inputs to this process are:

– a luma location ( xTbY, yTbY ) specifying the top-left sample of the current luma transform block relative to the top‑left luma sample of the current picture,

– a variable nTbS specifying the size of the current transform block,

– a variable cIdx specifying the colour component of the current block,

– an (nTbS)x(nTbS) array d of scaled transform coefficients with elements d[ x ][ y ].

Output of this process is the (nTbS)x(nTbS) array r of residual samples with elements r[ x ][ y ].

Depending on the values of CuPredMode[ xTbY ][ yTbY ], nTbS, and cIdx, the variable trType is derived as follows:

– If CuPredMode[ xTbY ][ yTbY ] is equal to MODE\_INTRA, nTbS is equal to 4, and cIdx is equal to 0, trType is set equal to 1.

– Otherwise, trType is set equal to 0.

The (nTbS)x(nTbS) array r of residual samples is derived as follows:

1. Each (vertical) column of scaled transform coefficients d[ x ][ y ] with x = 0..nTbS − 1, y = 0..nTbS − 1 is transformed to e[ x ][ y ] with x = 0..nTbS − 1, y = 0..nTbS − 1 by invoking the one-dimensional transformation process as specified in subclause for each column x = 0..nTbS − 1 with the size of the transform block nTbS, the list d[ x ][ y ] with y = 0..nTbS − 1, and the transform type variable trType as inputs, and the output is the list e[ x ][ y ] with y = 0..nTbS − 1.
2. The intermediate sample values g[ x ][ y ] with  x = 0..nTbS − 1, y = 0..nTbS − 1  are derived as follows:

g[ x ][ y ] = Clip3( −32768, 32767, ( e[ x ][ y ] + 64 )  >>  7 ) (8‑275)

1. Each (horizontal) row of the resulting array g[ x ][ y ] with x = 0..nTbS − 1, y = 0..nTbS − 1 is transformed to r[ x ][ y ] with x = 0..nTbS − 1, y = 0..nTbS − 1 by invoking the one-dimensional transformation process as specified in subclause for each row y = 0..nTbS − 1 with the size of the transform block nTbS, the list g[ x ][ y ] with x = 0..nTbS − 1, and the transform type variable trType as inputs, and the output is the list r[ x ][ y ] with x = 0..nTbS − 1.

#### Transformation process

Inputs to this process are:

– a variable nTbS specifying the sample size of scaled transform coefficients,

– a list of scaled transform coefficients x with elements x[ j ], with j = 0..nTbS − 1.

– a transform type variable trType

Output of this process is the list of transformed samples y with elements y[ i ], with i = 0..nTbS − 1.

Depending on the value of trType, the following applies:

– If trType is equal to 1, the following transform matrix multiplication applies:

y[ i ] =  with i = 0..nTbS − 1 (8‑276)

where the transform coefficient array transMatrix is specified as follows:

transMatrix = (8‑277)

{

{29 55 74 84}

{74 74 0 −74}

{84 −29 −74 55}

{55 −84 74 −29}

}

– Otherwise (trType is equal to 0), the following transform matrix multiplication applies:

y[ i ] =  with i = 0..nTbS − 1, (8‑278)

----------------------------------------------------)

So for a 4x4 block coded as DMM4, there should be 16\*2\*4 =128 muls and 16\*2\*3= 96 adds for a PU. That is 128\*4 = 512 muls and 96\*4=384 adds for a 8x8 CU totally.

For a 8x8 block coded as Inter mode with TU size = 8x8, there should be 64\*8\*2= 1024 muls and 64\*7\*2= 896 adds.

1. **Reconstruction**

As described in the WD, we have

(----------------------------------------------------

### Picture construction process prior to in-loop filter process

Inputs to this process are:

– a location ( xCurr, yCurr ) specifying the top-left sample of the current block relative to the top‑left sample of the current picture component,

– a variable nCurrS specifying the size of the current block,

– a variable cIdx specifying the colour component of the current block,

– an (nCurrS)x(nCurrS) array predSamples specifying the predicted samples of the current block,

– an (nCurrS)x(nCurrS) array resSamples specifying the residual samples of the current block.

Depending on the value of the colour component cIdx, the following assignments are made:

– If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array SL and the function clipCidx1 corresponds to Clip1Y.

– Otherwise, if cIdx is equal to 1, recSamples corresponds to the reconstructed chroma sample array SCb and the function clipCidx1 corresponds to Clip1C.

– Otherwise (cIdx is equal to 2), recSamples corresponds to the reconstructed chroma sample array SCr and the function clipCidx1 corresponds to Clip1C.

The (nCurrS)x(nCurrS) block of the reconstructed sample array recSamples at location ( xCurr, yCurr ) is derived as follows:

recSamples[ xCurr + i ][ yCurr + j ] = clipCidx1( predSamples[ i ][ j ] + resSamples[ i ][ j ] ) (8‑283)  
 with i = 0..nCurrS − 1, j = 0..nCurrS − 1

----------------------------------------------------)

So for a 4x4 block coded as DMM4, there should be 16 adds for PU. That is 16\*4 = 64 adds for a 8x8 CU totally.

For a 8x8 block coded as Inter mode with TU size = 8x8, there should be 64 adds.

For a 8x8 CU coded with segInter-SDC, 2 adds are applied to calculate the 2 reconstructed value for the 2 segments.