|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Compression** | **ADD** | **CMP** | **AND** | **MUL** | **TBL** |
| **HTM12/J0035** | **No compress** | **0** | **0** | **0** | **0** | **0** |
| **HTM120/J0035** | **Run-length** | **MxN** | **2** | **0** | **0** | **0** |
| **J0025+J0035** | **Rotate+Run-Length** | **MxN+M** | **2** | **0** | **0** | **0** |
| **J0025** | **Rotate** | **5xMxN+3** | **5** | **0** | **0** | **MxN** |
| **J0021** | **Scale&Shift** | **2xMxN** | **2xMxN** |  |  | **MxN+1** |

**Run-length MxN Add+ 2 comparison**

for (y = 0; y < N; y++)

{

Pel = compressedPattern[y][0]

Run = compressedPattern[y][1]

for (x = 0; x < Run == 0 ? M - 1: Run; x++)

Pattern[x][y] = Pel

Pel = !Pel

for (;x < M - 1; x++)

Pattern[x][y] = Pel

}

Note: Substitute(“=”) is not add but it has similar complexity. Therefore we count “=” as add operations.

**Run-length+Rotate MxN+M Add+ 2 comparison**

for (x = M - 1; y >= 0; y--)

{

xx = M – 1 - x

Pel = compressedPattern[xx][0]

Run = compressedPattern[xx][1]

for (y = 0; y < Run == 0 ? N - 1 : Run; y++)

Pattern[x][y] = Pel

Pel = !Pel

for (;y < N - 1; y++)

Pattern[x][y] = Pel

}

Note: We found that rotate+run-length is the same computation with different loop direction. Therefore number of operation of rotate+run-length is equal to that of run-length.

**J0021(Scale+Shift)**2xMxN+2xMxN+2MxN

pelShiftFlag = wedge\_full\_tab\_idx[ xTb ][ yTb]>=NumWedgePattern[ 3 ] ? 1 : 0

ref\_wedge\_idx = pelShiftFlag ?   
wedge\_full\_tab\_idx – NumWedgePattern[3]: wedge\_full\_tab\_idx[ xTb ][ yTb ]

offsetX = pelShiftFlag ? scale : 0

offsetY = pelShiftFlag ? scale : 0

The variable wedgePattern[ x ][ y ] with x, y =0..nTbS − 1, specifying a binary partition pattern is derived as:

wedgePattern[ x ][ y ] = WedgePatternTable[ 3 ][ ref\_wedge\_idx ][ x2 ][ y2 ] ,

where x2 = clip3(0, nTbs-1, x + offsetX) >> scale, y2 = clip3(0, nTbs-1, y + offsetY) >> scale

Note: In real operation clipping(clip3) may not needed by more sophisticated looping.

**J0025 (rotate compression)**5xMxN+3+5+MxN

The values of variable wedgePattern[ x ][ y ] are derived as specified in the following:

cInc = wedgeOri == 2 ? − 1 : 0;

sInc = (wedgeOri == 1 ? 1 : −1) \* (1 + xInc);

xOffset = cInc + sInc < 0 ? patternSize − 1 ? 0

yOffset = −sInc + cInc < 0 ? patternSize − 1 ? 0

* + for( y = 0; y < patternSize; y++ )   
     for( x = 0; x < patternSize; x++ ) {
  + i = +iOffset + cInc \* x + sInc \* y
  + j = +jOffset − sInc \* x + cInc \* y

wedgePattern[ x ][ y ] = refPattern [ iOffset ][ jOffset ] ? refPattern [ i ][ j ] : 1 -refPattern [ i ][ j ]

}

Note: sInc and cInc is 0 or 1 or -1. Therefore multiply is not needed. When sInx is not equal to 0, cInc is equal to 0. Therefore it is sufficient to count either of sInc or cInc.