# Optimisation des transformées inverses de HEVC

# Transformée inverse 4x4 (isDST= false)

Transpose4x4\_0 et Transpose4x4\_1 sont des transpositions de matrices 4x4. IT4x4\_1D est une transformée 1D appliquée sur chaque ligne de la matrice 4x4.

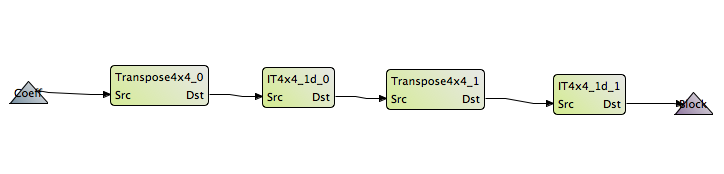


Figure 1: Diagramme de la transformée 4x4

Le pseudo associé à la transformée 4x4 est le suivant :

**action** Src:[ src ] **repeat** 4 // Coefficient matrices are processed column order, least significant first.

==>

Dst:[ [ (dst[i] + rouding\_factor) >> shift : **for** **int** i **in** 0 ..3] ] **repeat** 4

**var**

**List**(**type**:**int**(**size**=25), **size**=2 ) even, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=25), **size**=2 ) odd, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) dst, // one addition of two 25-bit summands = 26 bits

**int** rouding\_factor := 1 << (shift - 1)

**do**

even := [( src[0] \* g\_aiT4[0][0] ) + ( src[2] \* g\_aiT4[2][0]),

( src[0] \* g\_aiT4[0][1] ) + ( src[2] \* g\_aiT4[2][1])

];

odd := [( src[1] \* g\_aiT4[1][0] ) + ( src[3] \* g\_aiT4[3][0] ),

( src[1] \* g\_aiT4[1][1] ) + ( src[3] \* g\_aiT4[3][1] )

];

dst := [ even[0] + odd[0],

even[1] + odd[1],

even[1] - odd[1],

even[0] - odd[0]

];

**end**

# Transformée inverse 8x8

Transpose8x8\_0 et Transpose8x8\_1 sont des transpositions de matrices 8x8. IT8x8\_1D est une transformée 1D appliquée sur chaque ligne de la matrice 8x8.

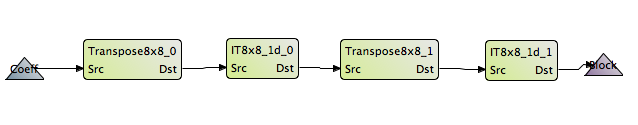


Figure 2: Diagramme de la transformée 8x8

Le pseudo associé à la transformée 8x8 est le suivant :

**action** Src:[ src ] **repeat** 8 // Coefficient matrices are processed column order, least significant first.

==>

Dst:[ [ (dst[i]+ rouding\_factor) >> shift : **for** **int** i **in** 0 .. 7] ] **repeat** 8

**var**

**List**(**type**:**int**(**size**=25), **size**=2 ) evenEven, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=25), **size**=2 ) evenOdd, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) even, // one addition of two 25-bit summands = 26 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) odd, // three additions of four (16-bit + 8-bit) summands = 26 bits

**List**(**type**:**int**(**size**=27), **size**=8 ) dst, // one addition of two 26-bit summands = 27 bits

**int** rouding\_factor := 1 << (shift - 1)

**do**

evenEven := [( src[0] \* g\_aiT8[0][0] ) + ( src[4] \* g\_aiT8[4][0] ),

( src[0] \* g\_aiT8[0][1] ) + ( src[4] \* g\_aiT8[4][1] )

];

evenOdd := [( src[2] \* g\_aiT8[2][0] ) + ( src[6] \* g\_aiT8[6][0] ),

( src[2] \* g\_aiT8[2][1] ) + ( src[6] \* g\_aiT8[6][1] )

];

even := [ evenEven[0] + evenOdd[0],

evenEven[1] + evenOdd[1],

evenEven[1] - evenOdd[1],

evenEven[0] - evenOdd[0]

];

odd := [( src[1] \* g\_aiT8[1][0] ) + ( src[3] \* g\_aiT8[3][0] ) + ( src[5] \* g\_aiT8[5][0] ) + ( src[7] \* g\_aiT8[7][0] ),

( src[1] \* g\_aiT8[1][1] ) + ( src[3] \* g\_aiT8[3][1] ) + ( src[5] \* g\_aiT8[5][1] ) + ( src[7] \* g\_aiT8[7][1] ),

( src[1] \* g\_aiT8[1][2] ) + ( src[3] \* g\_aiT8[3][2] ) + ( src[5] \* g\_aiT8[5][2] ) + ( src[7] \* g\_aiT8[7][2] ),

( src[1] \* g\_aiT8[1][3] ) + ( src[3] \* g\_aiT8[3][3] ) + ( src[5] \* g\_aiT8[5][3] ) + ( src[7] \* g\_aiT8[7][3] )

];

dst := [ even[0] + odd[0],

even[1] + odd[1],

even[2] + odd[2],

even[3] + odd[3],

even[3] - odd[3],

even[2] - odd[2],

even[1] - odd[1],

even[0] - odd[0]

];

**end**

# Transformée inverse 16x16

Transpose16x16\_0 et Transpose16x16\_1 sont des transpositions de matrices 16x16. IT16x16\_1D est une transformée 1D appliquée sur chaque ligne de la matrice 16x16.

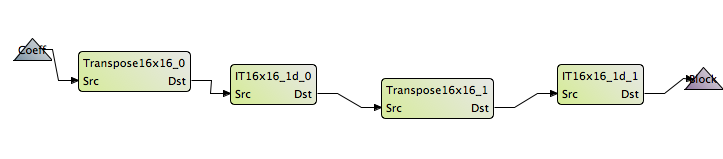


Figure 3: Diagramme de la transformée 16x16

Le pseudo associé à la transformée 16x16 est le suivant :

**action** Src:[ src ] **repeat** 16 // Coefficient matrices are processed column order, least significant first.

==>

Dst:[ [ (dst[i] + rouding\_factor) >> shift : **for** **int** i **in** 0 .. 15] ] **repeat** 16

**var**

**List**(**type**:**int**(**size**=25), **size**=2 ) evenEvenEven, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=25), **size**=2 ) evenEvenOdd, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) evenEven, // one addition of two 25-bit summands = 26 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) evenOdd, // three additions of four (16-bit + 8-bit) summands = 26 bits

**List**(**type**:**int**(**size**=27), **size**=8 ) even, // one addition of two 26-bit summands = 27 bits

**List**(**type**:**int**(**size**=27), **size**=8 ) odd, // seven additions of eight (16-bit + 8-bit) summands = 27 bits

**List**(**type**:**int**(**size**=28), **size**=16 ) dst, // one addition of two 27-bit summands = 28 bits

**int** rouding\_factor := 1 << (shift - 1)

**do**

evenEvenEven := [( src[0] \* g\_aiT16[0][0] ) + ( src[8] \* g\_aiT16[8][0] ),

( src[0] \* g\_aiT16[0][1] ) + ( src[8] \* g\_aiT16[8][1] )

];

evenEvenOdd := [( src[4] \* g\_aiT16[4][0] ) + ( src[12] \* g\_aiT16[12][0] ),

( src[4] \* g\_aiT16[4][1] ) + ( src[12] \* g\_aiT16[12][1] )

];

evenEven := [ evenEvenEven[0] + evenEvenOdd[0],

evenEvenEven[1] + evenEvenOdd[1],

evenEvenEven[1] - evenEvenOdd[1],

evenEvenEven[0] - evenEvenOdd[0]

];

evenOdd := [( src[2] \* g\_aiT16[2][0] ) + ( src[6] \* g\_aiT16[6][0] ) + ( src[10] \* g\_aiT16[10][0] ) + ( src[14] \* g\_aiT16[14][0] ),

( src[2] \* g\_aiT16[2][1] ) + ( src[6] \* g\_aiT16[6][1] ) + ( src[10] \* g\_aiT16[10][1] ) + ( src[14] \* g\_aiT16[14][1] ),

( src[2] \* g\_aiT16[2][2] ) + ( src[6] \* g\_aiT16[6][2] ) + ( src[10] \* g\_aiT16[10][2] ) + ( src[14] \* g\_aiT16[14][2] ),

( src[2] \* g\_aiT16[2][3] ) + ( src[6] \* g\_aiT16[6][3] ) + ( src[10] \* g\_aiT16[10][3] ) + ( src[14] \* g\_aiT16[14][3] )

];

even := [ evenEven[0] + evenOdd[0],

evenEven[1] + evenOdd[1],

evenEven[2] + evenOdd[2],

evenEven[3] + evenOdd[3],

evenEven[3] - evenOdd[3],

evenEven[2] - evenOdd[2],

evenEven[1] - evenOdd[1],

evenEven[0] - evenOdd[0]

];

odd := [( src[1] \* g\_aiT16[1][0] ) + ( src[3] \* g\_aiT16[3] [0] ) + ( src[5] \* g\_aiT16[5] [0] ) + ( src[7] \* g\_aiT16[7] [0] ) +

( src[9] \* g\_aiT16[9][0] ) + ( src[11] \* g\_aiT16[11][0] ) + ( src[13] \* g\_aiT16[13][0] ) + ( src[15] \* g\_aiT16[15][0] ),

( src[1] \* g\_aiT16[1][1] ) + ( src[3] \* g\_aiT16[3] [1] ) + ( src[5] \* g\_aiT16[5] [1] ) + ( src[7] \* g\_aiT16[7] [1] ) +

( src[9] \* g\_aiT16[9][1] ) + ( src[11] \* g\_aiT16[11][1] ) + ( src[13] \* g\_aiT16[13][1] ) + ( src[15] \* g\_aiT16[15][1] ),

( src[1] \* g\_aiT16[1][2] ) + ( src[3] \* g\_aiT16[3] [2] ) + ( src[5] \* g\_aiT16[5] [2] ) + ( src[7] \* g\_aiT16[7] [2] ) +

( src[9] \* g\_aiT16[9][2] ) + ( src[11] \* g\_aiT16[11][2] ) + ( src[13] \* g\_aiT16[13][2] ) + ( src[15] \* g\_aiT16[15][2] ),

( src[1] \* g\_aiT16[1][3] ) + ( src[3] \* g\_aiT16[3] [3] ) + ( src[5] \* g\_aiT16[5] [3] ) + ( src[7] \* g\_aiT16[7] [3] ) +

( src[9] \* g\_aiT16[9][3] ) + ( src[11] \* g\_aiT16[11][3] ) + ( src[13] \* g\_aiT16[13][3] ) + ( src[15] \* g\_aiT16[15][3] ),

( src[1] \* g\_aiT16[1][4] ) + ( src[3] \* g\_aiT16[3] [4] ) + ( src[5] \* g\_aiT16[5] [4] ) + ( src[7] \* g\_aiT16[7] [4] ) +

( src[9] \* g\_aiT16[9][4] ) + ( src[11] \* g\_aiT16[11][4] ) + ( src[13] \* g\_aiT16[13][4] ) + ( src[15] \* g\_aiT16[15][4] ),

( src[1] \* g\_aiT16[1][5] ) + ( src[3] \* g\_aiT16[3] [5] ) + ( src[5] \* g\_aiT16[5] [5] ) + ( src[7] \* g\_aiT16[7] [5] ) +

( src[9] \* g\_aiT16[9][5] ) + ( src[11] \* g\_aiT16[11][5] ) + ( src[13] \* g\_aiT16[13][5] ) + ( src[15] \* g\_aiT16[15][5] ),

( src[1] \* g\_aiT16[1][6] ) + ( src[3] \* g\_aiT16[3] [6] ) + ( src[5] \* g\_aiT16[5] [6] ) + ( src[7] \* g\_aiT16[7] [6] ) +

( src[9] \* g\_aiT16[9][6] ) + ( src[11] \* g\_aiT16[11][6] ) + ( src[13] \* g\_aiT16[13][6] ) + ( src[15] \* g\_aiT16[15][6] ),

( src[1] \* g\_aiT16[1][7] ) + ( src[3] \* g\_aiT16[3] [7] ) + ( src[5] \* g\_aiT16[5] [7] ) + ( src[7] \* g\_aiT16[7] [7] ) +

( src[9] \* g\_aiT16[9][7] ) + ( src[11] \* g\_aiT16[11][7] ) + ( src[13] \* g\_aiT16[13][7] ) + ( src[15] \* g\_aiT16[15][7] )

];

dst := [ even[0] + odd[0],

even[1] + odd[1],

even[2] + odd[2],

even[3] + odd[3],

even[4] + odd[4],

even[5] + odd[5],

even[6] + odd[6],

even[7] + odd[7],

even[7] - odd[7],

even[6] - odd[6],

even[5] - odd[5],

even[4] - odd[4],

even[3] - odd[3],

even[2] - odd[2],

even[1] - odd[1],

even[0] - odd[0]

];

**end**

# Transformée inverse 16x16

Transpose32x32\_0 et Transpose32x32\_1 sont des transpositions de matrices 32x32. IT32x32\_1D est une transformée 1D appliquée sur chaque ligne de la matrice 32x32.

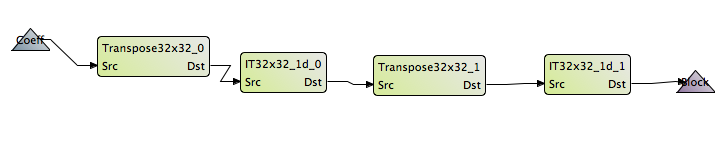


Figure 4: Diagramme de la transformée 32x32

Le pseudo associé à la transformée 32x32 est le suivant :

**action** Src:[ src ] **repeat** 32 // Coefficient matrices are processed column order, least significant first.

==>

Dst:[ [ (dst[i] + rouding\_factor) >> shift : **for** **int** i **in** 0 .. 31] ] **repeat** 32

**var**

**List**(**type**:**int**(**size**=25), **size**=2 ) evenEvenEvenEven, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=25), **size**=2 ) evenEvenEvenOdd, // one addition of two (16-bit + 8-bit) summands = 25 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) evenEvenEven, // one addition of two 25-bit summands = 26 bits

**List**(**type**:**int**(**size**=26), **size**=4 ) evenEvenOdd, // three additions of four (16-bit + 8-bit) summands = 26 bits

**List**(**type**:**int**(**size**=27), **size**=8 ) evenEven, // one addition of two 26-bit summands = 27 bits

**List**(**type**:**int**(**size**=27), **size**=8 ) evenOdd, // seven additions of eight (16-bit + 8-bit) summands = 27 bits

**List**(**type**:**int**(**size**=28), **size**=16 ) even, // one addition of two 27-bit summands = 28 bits

**List**(**type**:**int**(**size**=28), **size**=16 ) odd, // fifteen additions of sixteen (16-bit + 8-bit) summands = 28 bits

**List**(**type**:**int**(**size**=29), **size**=32 ) dst, // one addition of two 28-bit summands = 29 bits

**int** rouding\_factor := 1 << (shift - 1)

**do**

evenEvenEvenEven := [( src[0] \* g\_aiT32[0][0] ) + ( src[16] \* g\_aiT32[16][0] ),

( src[0] \* g\_aiT32[0][1] ) + ( src[16] \* g\_aiT32[16][1] )

];

evenEvenEvenOdd := [( src[8] \* g\_aiT32[8][0] ) + ( src[24] \* g\_aiT32[24][0] ),

( src[8] \* g\_aiT32[8][1] ) + ( src[24] \* g\_aiT32[24][1] )

];

evenEvenEven := [ evenEvenEvenEven[0] + evenEvenEvenOdd[0],

evenEvenEvenEven[1] + evenEvenEvenOdd[1],

evenEvenEvenEven[1] - evenEvenEvenOdd[1],

evenEvenEvenEven[0] - evenEvenEvenOdd[0]

];

evenEvenOdd := [ ( src[4] \* g\_aiT32[4][i] ) + ( src[12] \* g\_aiT32[12][i] ) + ( src[20] \* g\_aiT32[20][i] ) + ( src[28] \* g\_aiT32[28][i] )

: **for** **int** i **in** 0 .. 3

];

evenEven := [ evenEvenEven[0] + evenEvenOdd[0],

evenEvenEven[1] + evenEvenOdd[1],

evenEvenEven[2] + evenEvenOdd[2],

evenEvenEven[3] + evenEvenOdd[3],

evenEvenEven[3] - evenEvenOdd[3],

evenEvenEven[2] - evenEvenOdd[2],

evenEvenEven[1] - evenEvenOdd[1],

evenEvenEven[0] - evenEvenOdd[0]

];

evenOdd := [( src[2] \* g\_aiT32[2] [i] ) + ( src[6] \* g\_aiT32[6] [i] ) + ( src[10] \* g\_aiT32[10][i] ) + ( src[14] \* g\_aiT32[14][i] ) +

( src[18] \* g\_aiT32[18][i] ) + ( src[22] \* g\_aiT32[22][i] ) + ( src[26] \* g\_aiT32[26][i] ) + ( src[30] \* g\_aiT32[30][i] )

: **for** **int** i **in** 0 .. 7

];

even := [ evenEven[0] + evenOdd[0],

evenEven[1] + evenOdd[1],

evenEven[2] + evenOdd[2],

evenEven[3] + evenOdd[3],

evenEven[4] + evenOdd[4],

evenEven[5] + evenOdd[5],

evenEven[6] + evenOdd[6],

evenEven[7] + evenOdd[7],

evenEven[7] - evenOdd[7],

evenEven[6] - evenOdd[6],

evenEven[5] - evenOdd[5],

evenEven[4] - evenOdd[4],

evenEven[3] - evenOdd[3],

evenEven[2] - evenOdd[2],

evenEven[1] - evenOdd[1],

evenEven[0] - evenOdd[0]

];

odd := [( src[1] \* g\_aiT32[1] [i] ) + ( src[3] \* g\_aiT32[3] [i] ) + ( src[5] \* g\_aiT32[5] [i] ) + ( src[7] \* g\_aiT32[7] [i] ) +

( src[9] \* g\_aiT32[9] [i] ) + ( src[11] \* g\_aiT32[11][i] ) + ( src[13] \* g\_aiT32[13][i] ) + ( src[15] \* g\_aiT32[15][i] ) +

( src[17] \* g\_aiT32[17][i] ) + ( src[19] \* g\_aiT32[19][i] ) + ( src[21] \* g\_aiT32[21][i] ) + ( src[23] \* g\_aiT32[23][i] ) +

( src[25] \* g\_aiT32[25][i] ) + ( src[27] \* g\_aiT32[27][i] ) + ( src[29] \* g\_aiT32[29][i] ) + ( src[31] \* g\_aiT32[31][i] )

: **for** **int** i **in** 0 .. 15

];

dst := [ even[0] + odd[0] ,

even[1] + odd[1] ,

even[2] + odd[2] ,

even[3] + odd[3] ,

even[4] + odd[4] ,

even[5] + odd[5] ,

even[6] + odd[6] ,

even[7] + odd[7] ,

even[8] + odd[8] ,

even[9] + odd[9] ,

even[10] + odd[10],

even[11] + odd[11],

even[12] + odd[12],

even[13] + odd[13],

even[14] + odd[14],

even[15] + odd[15],

even[15] - odd[15],

even[14] - odd[14],

even[13] - odd[13],

even[12] - odd[12],

even[11] - odd[11],

even[10] - odd[10],

even[9] - odd[9] ,

even[8] - odd[8] ,

even[7] - odd[7] ,

even[6] - odd[6] ,

even[5] - odd[5] ,

even[4] - odd[4] ,

even[3] - odd[3] ,

even[2] - odd[2] ,

even[1] - odd[1] ,

even[0] - odd[0]

];

**end**

# Optimisation DSP

Travaux à réaliser :

* Optimisation des 4 transformées 4x4, 8x8, 16 et 32x32 et de la DST
  + Utilisation des const, restrict, nassert, pragmas
  + Utilisation d’intrinsèques
  + Utilisation de la cache, DMA
* Contraintes :
  + Tableaux coeff et dst à mettre en mémoire interne
* Fournir une recette des différentes optimisations

# Paréllélisation sur multi DSP

* Parallélisation sur l’architecture VITEC