|  |  |
| --- | --- |
| **99Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  9th Meeting: Geneva, CH, 27 April-7 May 2012 | Document: JCTVC-I0036 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Parallel AMVP candidate list construction** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Qin Yu,  Siwei Ma  Hongbin Liu Jie Jia | Tel: Email: [qyu@jdl.ac.cn](mailto:qyu@jdl.ac.cn)  [swma@jdl.ac.cn](mailto:swma@jdl.ac.cn)  [hongbin.liu@lge.com](mailto:hongbin.liu@lge.com)  [jie.jia@lge.com](mailto:jie.jia@lge.com) |  |
| *Source:* | LG Electronics (China) R&D Center, Peking University | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This contribution proposes three solutions for parallel construction of AMVP candidate list (AMVPCL) at different parallel levels. Solution one is a CU-based approach and it constructs AMVPCL of all PUs in the same CU in parallel. Solution two is also a CU-based approach but it generates a single AMVPCL for all PUs inside a CU. Solution three is a region-based approach, in which AMVPCL of all PUs in the same region are constructed in parallel. Solution two and solution three are designed to be compatible with the parallel merge/skip mode in HM6.0. It is reported that average BD-rate loss of Y, U and V is roughly 0.2%, 0.2% and 0.2% for solution 1, 0.1%, 0.1%, and 0.0% for solution 2, and 0.0 %, 0.1% and -0.1% for the solution 3.

# Introduction

In the current HEVC framework, there is dependency among neighboring PUs in all inter modes excluding Merge/Skip mode. This makes parallel processing of multiple PUs difficult for both the encoder and decoder in inter modes.

In HM6.0, the AMVPCL consists of both spatial candidates and temporal candidates. Spatial candidates are classified into 2 categories [1], i.e. top category (T-1, Tn and Tn+1) and left category (Lm and Lm+1), as shown in Figure 1(a). In each category, the first available candidate in the search order is added to the AMVPCL. After the spatial candidates are derived, a temporal candidate is added to the AMVPCL, as shown in Figure 1(b). Therefore, the candidate list contains 2 spatial candidates and 1 temporal candidate at most.

With the current HM6.0 design, when encoding a CU, only the first PU in it can immediately derive its AMVPCL while other PU has to wait until its preceding PUs are encoded, as shown in Figure 2. In Figure 2, different types of candidates are highlighted in different colors. The top ones are in red, the left ones are in blue, the gray ones indicate candidates that are available until they are coded, and the green ones are candidates that are not encoded yet. Figure 3 provides an example to describe this problem. As can be seen, at encoder side, the derivation process of AMVPCL as well as the motion estimation process for different PUs must be carried out sequentially; at the decoder, the derivation process of AMVPCL should also be carried out sequentially for different PUs

To meet parallel AMVPCL derivation requirement as well as parallel motion estimation/motion compensation requirement, this contribution proposes three solutions at different parallel levels.

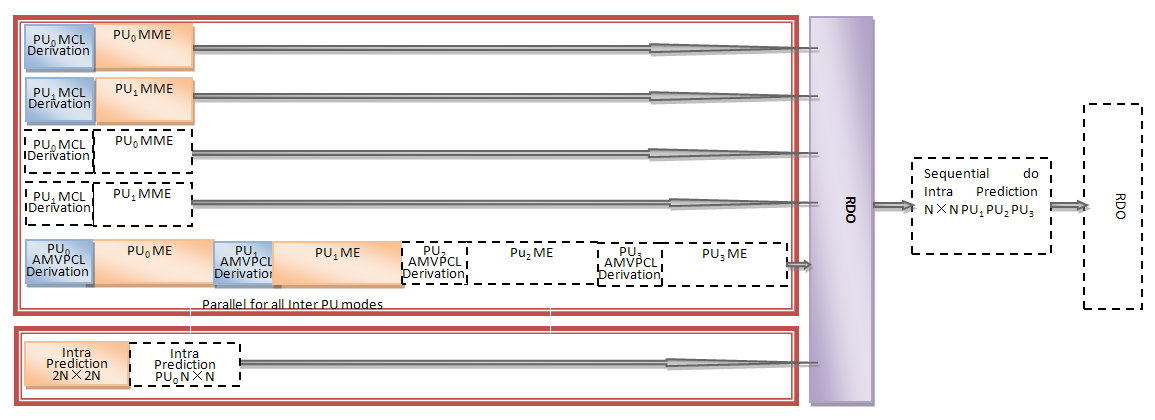


1. Spatial Candidates (b) Temporal Candidates

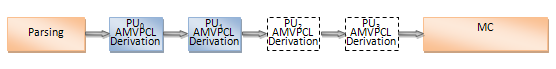
Figure 1. Illustration of the AMVP candidate list construction of HM6.0



**Figure 2.** AMVP candidate list of PUs in a CU



1. Encoder

****

(b) Decoder

**Figure 3.** An example of motion estimation in HM6.0

# Algorithm Description

## Solution 1

In solution 1, AMVPCL of all PUs in a CU are constructed in parallel. In the parallel AMVPCL construction process, candidates within the CU are unavailable and replaced by corresponding candidates outside the CU. Figure 4 illustrates proposed method for both symmetric motion partition (SMP) cases and asymmetric motion partition (AMP) cases. As can be seen from Figure 4, inner candidates (in gray and green) of the CU are replaced by corresponding outer candidates pointed by the arrows. Specifically, if the above or above right candidate is within the current CU, it is replaced by the above or above right candidate of the current PU’s above neighboring PU respectively; if the above left, left or below left candidate is within the current CU, it is replaced by the above left, left or below left candidate of the current PU’s left neighboring PU respectively.

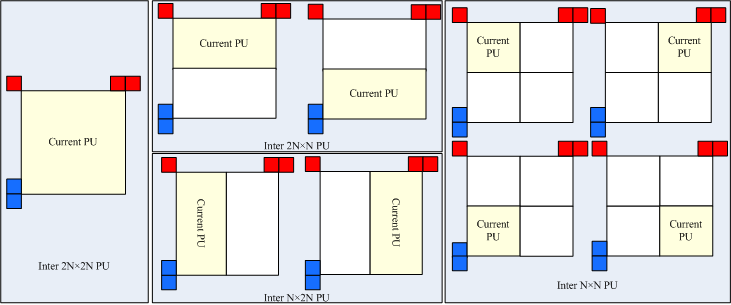




**Figure 4.** Proposed solution: a CU-based approach for AMVP candidate list construction.

## Solution 2

In solution 2, all PUs in the CU uses motion information (including MV, reference list and reference index) of same set of neighboring PUs to construct AMVPCL. To be specific, motion information used by inter 2Nx2N PU for constructing AMVPCL is used by all PUs. Figure 5 illustrates proposed method for SMP cases, and the same concept also applies to AMP cases. To balance the parallelism and rate-distortion performance, this idea is only applied to 8x8 CU.



**Figure 5.** Proposed solution: all PUs in the CU use motion information of same set of neighboring PUs to construct AMVPCL.

## Solution 3

Enlightened by parallel merge/skip mode [2] in HM6.0, this contribution proposes a parallel AMVPCL construction method. In the proposed method, AMVPCL of all PUs in one motion estimation region (MER) are constructed in parallel. If some involved candidate in AMVPCL construction process is within the current processing MER, it is disabled. To be compatible with parallel merge/skip mode in HM6.0, the MER size is also derived from **log2\_parallel\_merge\_level\_minus2**, and proposed method is employed on when **log2\_parallel\_merge\_level\_minus2** is 1, i.e., the MER size is 8x8.

To determine whether a PU and its neighboring PU are within the same MER, we need to check the following condition as parallel merge/skip mode does:

(xP >>( **log2\_parallel\_merge\_level\_minus2+2)) == (**xN **>> ( log2\_parallel\_merge\_level\_minus2+2))**

and

(yP >>( **log2\_parallel\_merge\_level\_minus2+2)) == (**yN **>> ( log2\_parallel\_merge\_level\_minus2+2))**

Where (xP, yP) and (xN, yN) are the coordinates of the top-left corner pixel of the current PU and its neighboring PU. If this condition is true, the neighboring PU is unavailable.

# Experimental Results

The proposed method is implemented on HM6.0 software and compared with it under common test conditions [3].

Encoding of all sequences is run on computer cluster whose compute note is equipped with Intel (R) Xeon (R) CPU X5660 @ 2.80GHZ Memory 24G, Windows xp, 64 bit compiler. Decoding of all bit streams is run on computer of the specification: Intel (R) Xeon (TM) 2 Duo CPU E8400 @3GHz Memory 3.25GB, Windows XP, 32 bit compiler.

Experimental results of three solutions are shown in Table 1 to Table 3 respectively. It is reported that average BD-rate loss of Y, U and V is roughly 0.2%, 0.2% and 0.2% for solution 1, 0.1%, 0.1%, and 0.0% for solution 2, and 0.0 %, 0.1% and -0.1% for the solution 3.

Table 1. Summary results of Solution 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Random Access Main** | | | **Random Access HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A | 0.2% | 0.5% | 0.3% | 0.2% | 0.3% | 0.3% |
| Class B | 0.2% | 0.3% | 0.4% | 0.2% | 0.2% | 0.3% |
| Class C | 0.3% | 0.4% | 0.4% | 0.4% | 0.4% | 0.4% |
| Class D | 0.3% | 0.4% | 0.3% | 0.3% | 0.3% | 0.1% |
| Class E |  |  |  |  |  |  |
| **Overall** | 0.3% | 0.4% | 0.4% | 0.3% | 0.3% | 0.3% |
|  | 0.3% | 0.4% | 0.4% | 0.3% | 0.3% | 0.3% |
| Class F | 0.5% | 0.6% | 0.5% | 0.6% | 0.5% | 0.6% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay B Main** | | | **Low delay B HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.1% | -0.1% | 0.5% | 0.1% | 0.2% | 0.1% |
| Class C | 0.2% | 0.4% | 0.3% | 0.2% | 0.3% | 0.2% |
| Class D | 0.3% | 0.3% | 0.7% | 0.2% | 0.1% | -0.3% |
| Class E | 0.1% | -0.3% | 0.0% | 0.1% | 0.3% | 0.2% |
| **Overall** | 0.2% | 0.1% | 0.4% | 0.2% | 0.2% | 0.0% |
|  | 0.2% | 0.1% | 0.4% | 0.2% | 0.2% | 0.0% |
| Class F | 0.6% | 0.8% | 1.1% | 0.6% | 1.2% | 1.1% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay P Main** | | | **Low delay P HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.1% | 0.1% | 0.0% | 0.0% | 0.2% | 0.2% |
| Class C | 0.2% | -0.2% | -0.1% | 0.1% | 0.0% | 0.3% |
| Class D | 0.1% | 0.5% | -0.8% | 0.1% | 0.5% | 0.0% |
| Class E | 0.0% | -0.1% | 0.5% | 0.1% | 0.3% | 0.1% |
| **Overall** | 0.1% | 0.1% | -0.1% | 0.1% | 0.2% | 0.1% |
|  | 0.1% | 0.1% | -0.1% | 0.1% | 0.2% | 0.2% |
| Class F | 0.6% | 0.7% | 0.5% | 0.5% | 2.4% | 0.2% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |

Table 2. Summary results of Solution 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Random Access Main** | | | **Random Access HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A | 0.1% | 0.2% | 0.2% | 0.1% | 0.3% | 0.2% |
| Class B | 0.1% | 0.1% | 0.1% | 0.0% | 0.1% | 0.0% |
| Class C | 0.1% | 0.2% | 0.2% | 0.1% | 0.1% | 0.0% |
| Class D | 0.1% | 0.1% | 0.3% | 0.2% | 0.1% | 0.0% |
| Class E |  |  |  |  |  |  |
| **Overall** | 0.1% | 0.2% | 0.2% | 0.1% | 0.1% | 0.0% |
|  | 0.1% | 0.2% | 0.2% | 0.1% | 0.1% | 0.0% |
| Class F | 0.2% | 0.2% | 0.3% | 0.2% | 0.1% | 0.3% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay B Main** | | | **Low delay B HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.0% | -0.2% | -0.3% | 0.0% | -0.1% | 0.3% |
| Class C | 0.1% | 0.2% | 0.1% | 0.1% | 0.1% | 0.2% |
| Class D | 0.1% | 0.1% | 0.3% | 0.1% | 0.1% | -0.5% |
| Class E | 0.0% | 0.1% | 0.3% | 0.0% | -0.1% | 0.2% |
| **Overall** | 0.1% | 0.0% | 0.1% | 0.0% | 0.0% | 0.0% |
|  | 0.1% | 0.1% | 0.1% | 0.0% | 0.0% | 0.0% |
| Class F | 0.3% | 0.3% | 0.3% | 0.3% | 0.7% | 0.5% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay P Main** | | | **Low delay P HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.0% | 0.0% | -0.1% | 0.0% | 0.2% | 0.2% |
| Class C | 0.1% | -0.4% | 0.0% | 0.1% | 0.0% | -0.2% |
| Class D | 0.1% | -0.2% | -0.9% | 0.0% | 0.6% | -0.1% |
| Class E | 0.1% | 0.2% | -0.1% | 0.1% | 0.7% | 0.3% |
| **Overall** | 0.1% | -0.1% | -0.3% | 0.0% | 0.3% | 0.0% |
|  | 0.1% | -0.1% | -0.2% | 0.0% | 0.3% | 0.1% |
| Class F | 0.4% | 0.5% | 0.4% | 0.3% | 1.5% | 1.2% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |

Table 3. Summary results of Solution 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Random Access Main** | | | **Random Access HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A | 0.0% | 0.2% | 0.1% | 0.0% | 0.1% | 0.1% |
| Class B | 0.0% | 0.1% | 0.2% | 0.0% | 0.0% | 0.0% |
| Class C | 0.1% | 0.2% | 0.1% | 0.1% | 0.1% | 0.2% |
| Class D | 0.1% | 0.2% | 0.1% | 0.2% | 0.1% | 0.0% |
| Class E |  |  |  |  |  |  |
| **Overall** | 0.1% | 0.2% | 0.1% | 0.1% | 0.1% | 0.1% |
|  | 0.1% | 0.2% | 0.1% | 0.1% | 0.1% | 0.1% |
| Class F | 0.2% | 0.3% | 0.3% | 0.2% | 0.3% | 0.3% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay B Main** | | | **Low delay B HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.0% | -0.2% | -0.3% | 0.0% | -0.1% | 0.0% |
| Class C | 0.0% | 0.3% | 0.2% | 0.1% | 0.0% | 0.0% |
| Class D | 0.1% | 0.2% | 0.4% | 0.0% | 0.0% | -0.3% |
| Class E | 0.1% | 0.0% | 0.3% | -0.1% | -0.3% | -0.1% |
| **Overall** | 0.0% | 0.1% | 0.1% | 0.0% | -0.1% | -0.1% |
|  | 0.0% | 0.1% | 0.1% | 0.0% | -0.1% | -0.1% |
| Class F | 0.4% | 0.2% | 1.0% | 0.5% | 0.9% | 0.8% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay P Main** | | | **Low delay P HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B | 0.0% | 0.0% | 0.1% | 0.0% | 0.2% | -0.3% |
| Class C | 0.0% | 0.0% | -0.1% | 0.0% | 0.0% | -0.1% |
| Class D | 0.0% | -0.4% | -0.3% | 0.0% | 0.3% | -0.8% |
| Class E | -0.1% | -0.2% | -0.2% | 0.0% | 0.7% | -0.4% |
| **Overall** | 0.0% | -0.1% | -0.1% | 0.0% | 0.3% | -0.4% |
|  | 0.0% | -0.1% | -0.1% | 0.0% | 0.3% | -0.2% |
| Class F | 0.4% | 0.7% | 0.2% | 0.3% | 1.1% | 0.7% |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |

# Complexity Analysis

## Solution 1

From Fig.4, it can be seen that the proposed approach needs one additional position derivation operation for Size\_2NxN, Size\_2NxnU, Size\_2NxnD, Size\_Nx2N, Size\_nLx2N and Size\_nRx2N partition for CU larger than 8x8 and needs five additional position derivation operations for NxN partition of 8x8 CU.

## Solution 2

**Encoder:** In the current HM encoder design, every PU should construct its own AMVPCL in each motion estimation process, which makes the number of AMVPCLs very large. For a 64x64 block, this number is 593, as shown in the Table 1. Proposed solution can reduce the number of AMVPCLs required for 8x8 CU in motion estimation by 80%, and reduce the total AMVPCL construction number by 43%.

Table 4. Number of AMVPCLs that should be constructed for 64x64 block

|  |  |  |
| --- | --- | --- |
| **CU Size** | **AMVPCL construction for a 64x64 block** | |
| **HM6.0** | **Proposed** |
| 64x64 | 13 | 13 |
| 32x32 | 13\*4 = 52 | 13\*4 = 52 |
| 16x16 | 13\*4\*4 = 208 | 13\*4\*4 = 208 |
| 8x8 | 5\*4\*4\*4 = 320 | 1\*4\*4\*4 = 64 |
| **Sum** | **593** | **337** |

**Decoder:** From Fig.5, it can be seen that AMVPCL needs to be constructed only once for PUs sharing the same reference picture.

In the software implementation, the decoder needs to check whether the size of an inter-predicted CU is 8x8. Meanwhile, for inter-predicted 8x8 CU, the decoder needs to change (to Size\_2Nx2N) and restore its partition size before and after constructing AMVPCL for a PU in it respectively.

## Solution 3

The proposed parallel AMVPCL construction approach needs to determine whether the current PU and its neighboring PU belong to the same MER during the spatial MV candidate derivation process. However, this complexity increase is negligible when compared to the complexity of availability checks required in the HM6.0 spatial MV candidate derivation process.

# Conclusion and Recommendations

Proposed approaches improve parallelism of all inter modes excluding merge/skip mode hence make the HM design more friendly to high-throughput implementation, at the cost of only small BD-rate increase. It is recommended to review this proposal and adopt proposed approaches into the HEVC.

# References

1. Liang Zhao,Xun Guo,Shawmin Lei, Siwei Ma, Debin Zhao, Wen Gao, “Non-CE9: Simplification of AMVP”, JCTVC-H0316, 8th JCT-VC Meeting, San Jose, CA, USA, 1-10 February, 2012.
2. Minhua Zhou, “AHG10: Configurable and CU-group level parallel merge/skip”, JCTVC-H0082, 8th JCT-VC Meeting, San Jose, CA, USA, 1-10 February, 2012.
3. Frank Bossen, “Common test conditions and software reference configurations”, JCTVC-H1100, 8th JCT-VC Meeting, San Jose, CA, USA, 1-10 February, 2012.

# Patent rights declaration(s)

**LG Electronics / LG Electronics (china) R&D Center and Peking University** **may have IPR relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# WD Text Changes (changes marked in yellow)

* **Solution 1:**

**8.5.2.1.6 Derivation process for motion vector predictor candidates**

Inputs to this process are

* a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
* PartMode of the current coding unit
* Variables specifying the current prediction unit index in the coding unit, PartIdx
* a variable nCS specifying the size of the current coding unit,
* variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
* the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are (with N being replaced by A, or B)

* the motion vectors mvLXN of the neighbouring prediction units,
* the availability flags availableFlagLXN of the neighbouring prediction units.



**Figure 8‑3 – Spatial motion vector neighbours**

The variable isScaledFlagLX with X being 0 or 1 is set equal to 0.

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. Let a set of two sample locations be (xAk, yAk), with k = 0, 1, specifies sample locations with xAk = xP − 1, yA0 = yP + nPSH and yA1 = yA0 - MinPuSize. The set of sample locations ( xAk, yAk ) represent the sample locations immediately to the left side of the left partition boundary and it’s extended line. If PartMode is equal to PART\_NxN and PartIdx is equal to 1, xA0 = xA0 - (nCS - nPSW). If PartMode is equal to PART\_Nx2N, PART\_nLx2N or PART\_nRx2N and PartIdx is equal to 1, xA1 = xA1 - (nCS - nPSW). If PartMode is equal to PART\_NxN and PartIdx is equal to 1 or 3, xA1 = xA1 - (nCS - nPSW).
2. Let the availability flag availableFlagLXA be initially set equal to 0 and the both components of mvLXA are set equal to 0.
3. If one of the following conditions is true, the variable isScaledFlagLX is set equal to 1.

* the prediction unit covering luma location ( xA0, yA0 ) is available and PredMode is not MODE\_INTRA.
* the prediction unit covering luma location ( xA1, yA1 ) is available and PredMode is not MODE\_INTRA.

1. For ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 − MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk,yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1 and the reference index refIdxLX[ xAk ][ yAk ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXA is set equal to 1 and the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ] and ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xAk ][ yAk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ] ,  ListA is set equal to LY and mvLXA is set equal to mvLXA.

1. When availableFlagLXA is equal to 0, for ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 - MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ], ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ], ListA is set equal to LY.
* When availableFlagLXA is equal to 1, mvLXA is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑136)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑137)

mvLXA = Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*  ( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑138)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListA( refIdxA ) ) ) (8‑139)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑140)

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. Let a set of three sample location (xBk, yBk), with k = 0,1,2, specifies sample locations with xB0 = xP + nPSW, xB1 = xB0− MinPuSize , xB2 = xP − MinPuSize and yBk = yP − 1. The set of sample locations ( xBk, yBk ) represent the sample locations immediately to the upper side of the above partition boundary and its extended line. If PartMode is equal to PART\_NxN and PartIdx is equal to 2, yB0 = yB0 –(nCS - nPSH); If PartMode is equal to PART\_2NxN, PART\_2NxnU or PART\_2NxnD and PartIdx is equal to 1, yB1 = yB1 – (nCS - nPSH); If PartMode is equal to PART\_NxN and PartIdx is equal to 2 or 3, yB1 = yB1 –(nCS - nPSH); If PartMode is equal to PART\_NxN and PartIdx is equal to 3, xB2 = xB2 –(nCS - nPSH).
2. If yP−1 is less than yCtb, the following applies.

xB0 = (xB0>>Log2MinCbSize)<<Log2MinCbSize) + ((xB0>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑141)  
xB1 = (xB1>>Log2MinCbSize)<<Log2MinCbSize) + ((xB1>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑142)  
xB2 = (xB2>>Log2MinCbSize)<<Log2MinCbSize) + ((xB2>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑143)

where MinPuSize = 1 << ( Log2MinCbSize − 1)

1. Let the availability flag availableFlagLXB be initially set equal to 0 and the both components of mvLXB are set equal to 0.
2. For ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP + nPSW, xB1 = xB0 − MinPuSize , and xB2 =  xP − MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xBk ][ yBk ] is equal to 1, and the reference index refIdxLX[ xBk ][ yBk ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXB is set equal to 1 and the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ] and ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xBk ][ yBk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ],  and ListB is set equal to LY.

1. When isScaledFlagLX is equal to 0 and availableFlagLXB is equal to 1,  mvLXA is set equal to mvLXB and refIdxA is set equal to refIdxB and availableFlagLXA is set equal to 1.
2. When isScaledFlagLX is equal to 0, availableFlagLXB is set equal to 0 and for ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP +nPSW, xB1 = xB0 - MinPuSize , and xB2 =  xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xBk ][ yBk ] is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ], ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ], ListB is set equal to LY.
* When availableFlagLXB is equal to 1 and PicOrderCnt( RefPicListListB( refIdxB ) ) is not equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), mvLXB is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑144)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑145)

mvLXB =Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*   
 ( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑146)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListB( refIdxB ) ) ) (8‑147)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑148)

* **Solution 2:**

**8.5.2.1.6 Derivation process for motion vector predictor candidates**

Inputs to this process are

* a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,
* a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
* a variable nCS specifying the size of the current coding unit,
* variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
* the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are (with N being replaced by A, or B)

* the motion vectors mvLXN of the neighbouring prediction units,
* the availability flags availableFlagLXN of the neighbouring prediction units.



**Figure 8‑3 – Spatial motion vector neighbours**

When nCS is not greater than 8, (xP, yP) is set equal to (xC, yC) and nPSW and nPSH are set equal to nCS.

The variable isScaledFlagLX with X being 0 or 1 is set equal to 0.

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. Let a set of two sample locations be (xAk, yAk), with k = 0, 1, specifies sample locations with xAk = xP − 1, yA0 = yP + nPSH and yA1 = yA0 - MinPuSize. The set of sample locations ( xAk, yAk ) represent the sample locations immediately to the left side of the left partition boundary and it’s extended line.
2. Let the availability flag availableFlagLXA be initially set equal to 0 and the both components of mvLXA are set equal to 0.
3. If one of the following conditions is true, the variable isScaledFlagLX is set equal to 1.

* the prediction unit covering luma location ( xA0, yA0 ) is available and PredMode is not MODE\_INTRA.
* the prediction unit covering luma location ( xA1, yA1 ) is available and PredMode is not MODE\_INTRA.

1. For ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 − MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk,yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1 and the reference index refIdxLX[ xAk ][ yAk ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXA is set equal to 1 and the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ] and ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xAk ][ yAk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ] ,  ListA is set equal to LY and mvLXA is set equal to mvLXA.

1. When availableFlagLXA is equal to 0, for ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 - MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ], ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ], ListA is set equal to LY.
* When availableFlagLXA is equal to 1, mvLXA is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑136)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑137)

mvLXA = Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*

( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑138)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListA( refIdxA ) ) ) (8‑139)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑140)

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. Let a set of three sample location (xBk, yBk), with k = 0,1,2, specifies sample locations with xB0 = xP + nPSW, xB1 = xB0− MinPuSize , xB2 = xP − MinPuSize and yBk = yP − 1. The set of sample locations ( xBk, yBk ) represent the sample locations immediately to the upper side of the above partition boundary and its extended line.
2. If yP−1 is less than yCtb, the following applies.

xB0 = (xB0>>Log2MinCbSize)<<Log2MinCbSize) + ((xB0>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑141)  
xB1 = (xB1>>Log2MinCbSize)<<Log2MinCbSize) + ((xB1>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑142)  
xB2 = (xB2>>Log2MinCbSize)<<Log2MinCbSize) + ((xB2>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑143)

where MinPuSize = 1 << ( Log2MinCbSize − 1)

1. Let the availability flag availableFlagLXB be initially set equal to 0 and the both components of mvLXB are set equal to 0.
2. For ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP + nPSW, xB1 = xB0 − MinPuSize , and xB2 =  xP − MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xBk ][ yBk ] is equal to 1, and the reference index refIdxLX[ xBk ][ yBk ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXB is set equal to 1 and the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ] and ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xBk ][ yBk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ],  and ListB is set equal to LY.

1. When isScaledFlagLX is equal to 0 and availableFlagLXB is equal to 1,  mvLXA is set equal to mvLXB and refIdxA is set equal to refIdxB and availableFlagLXA is set equal to 1.
2. When isScaledFlagLX is equal to 0, availableFlagLXB is set equal to 0 and for ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP +nPSW, xB1 = xB0 - MinPuSize , and xB2 =  xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xBk ][ yBk ] is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ], ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ], ListB is set equal to LY.
* When availableFlagLXB is equal to 1 and PicOrderCnt( RefPicListListB( refIdxB ) ) is not equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), mvLXB is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑144)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑145)

mvLXB =Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*   
 ( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑146)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListB( refIdxB ) ) ) (8‑147)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑148)

* **Solution 3:**

**8.5.2.1.6 Derivation process for motion vector predictor candidates**

Inputs to this process are

* a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
* variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
* the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are (with N being replaced by A, or B)

* the motion vectors mvLXN of the neighbouring prediction units,
* the availability flags availableFlagLXN of the neighbouring prediction units.



Figure ‑ – Spatial motion vector neighbours

The variable isScaledFlagLX with X being 0 or 1 is set equal to 0.

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. Let a set of two sample locations be (xAk, yAk), with k = 0, 1, specifies sample locations with xAk = xP − 1, yA0 = yP + nPSH and yA1 = yA0 - MinPuSize. The set of sample locations ( xAk, yAk ) represent the sample locations immediately to the left side of the left partition boundary and it’s extended line.
2. Let the availability flag availableFlagLXA be initially set equal to 0 and the both components of mvLXA are set equal to 0.
3. If one of the following conditions is true, the variable isScaledFlagLX is set equal to 1.

* the prediction unit covering luma location ( xA0, yA0 ) is available and PredMode is not MODE\_INTRA, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xA0 >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yA0 >> (log2\_parallel\_merge\_level\_minus2 +2)).
* the prediction unit covering luma location ( xA1, yA1 ) is available and PredMode is not MODE\_INTRA, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xA1 >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yA1 >> (log2\_parallel\_merge\_level\_minus2 +2)).

1. For ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 − MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk,yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1 and the reference index refIdxLX[ xAk ][ yAk ] is equal to the reference index of the current prediction unit refIdxLX, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xAk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yAk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXA is set equal to 1 and the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ] and ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xAk ][ yAk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xAk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yAk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ] ,  ListA is set equal to LY and mvLXA is set equal to mvLXA.

1. When availableFlagLXA is equal to 0, for ( xAk, yAk ) from ( xA0, yA0 ) to ( xA1, yA1 ) where yA1 = yA0 - MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:

* If the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xAk ][ yAk ] is equal to 1, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xAk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yAk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLX[ xAk ][ yAk ], refIdxA is set equal to refIdxLX[ xAk ][ yAk ], ListA is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xAk ][ yAk ] (with Y = !X) is equal to 1, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xAk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yAk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ xAk ][ yAk ], refIdxA is set equal to refIdxLY[ xAk ][ yAk ], ListA is set equal to LY.
* When availableFlagLXA is equal to 1, mvLXA is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑136)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑137)

mvLXA = Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*

( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑138)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListA( refIdxA ) ) ) (8‑139)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑140)

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. Let a set of three sample location (xBk, yBk), with k = 0,1,2, specifies sample locations with xB0 = xP + nPSW, xB1 = xB0− MinPuSize , xB2 = xP − MinPuSize and yBk = yP − 1. The set of sample locations ( xBk, yBk ) represent the sample locations immediately to the upper side of the above partition boundary and its extended line.
2. If yP−1 is less than yCtb, the following applies.

xB0 = (xB0>>Log2MinCbSize)<<Log2MinCbSize) + ((xB0>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑141)  
xB1 = (xB1>>Log2MinCbSize)<<Log2MinCbSize) + ((xB1>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑142)  
xB2 = (xB2>>Log2MinCbSize)<<Log2MinCbSize) + ((xB2>>Log2MinCbSize)&1)\*(MinPuSize\*2−1) (8‑143)

where MinPuSize = 1 << ( Log2MinCbSize − 1)

1. Let the availability flag availableFlagLXB be initially set equal to 0 and the both components of mvLXB are set equal to 0.
2. For ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP + nPSW, xB1 = xB0 − MinPuSize , and xB2 =  xP − MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xBk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yBk >> (log2\_parallel\_merge\_level\_minus2 +2)), predFlagLX[ xBk ][ yBk ] is equal to 1, and the reference index refIdxLX[ xBk ][ yBk ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXB is set equal to 1 and the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ] and ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1 and PicOrderCnt( RefPicListLY( refIdxLY[ xBk ][ yBk ] ) ) is equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xBk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yBk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ],  and ListB is set equal to LY.

1. When isScaledFlagLX is equal to 0 and availableFlagLXB is equal to 1,  mvLXA is set equal to mvLXB and refIdxA is set equal to refIdxB and availableFlagLXA is set equal to 1.
2. When isScaledFlagLX is equal to 0, availableFlagLXB is set equal to 0 and for ( xBk, yBk ) from ( xB0, yB0 ) to ( xB2, yB2 ) where xB0 = xP +nPSW, xB1 = xB0 - MinPuSize , and xB2 =  xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:

* If the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLX[ xBk ][ yBk ] is equal to 1, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xBk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yBk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLX[ xBk ][ yBk ], refIdxB is set equal to refIdxLX[ xBk ][ yBk ], ListB is set equal to LX.
* Otherwise if the prediction unit covering luma location ( xBk, yBk ) is available, PredMode is not MODE\_INTRA, predFlagLY[ xBk ][ yBk ] (with Y = !X) is equal to 1, and (xP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (xBk >> (log2\_parallel\_merge\_level\_minus2 +2)) or (yP >> (log2\_parallel\_merge\_level\_minus2 +2)) is not equal to (yBk >> (log2\_parallel\_merge\_level\_minus2 +2)), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ xBk ][ yBk ], refIdxB is set equal to refIdxLY[ xBk ][ yBk ], ListB is set equal to LY.
* When availableFlagLXB is equal to 1 and PicOrderCnt( RefPicListListB( refIdxB ) ) is not equal to PicOrderCnt( RefPicListLX( refIdxLX ) ), mvLXB is derived as specified below.

tx = ( 16384 + ( Abs( td ) >> 1 ) ) / td (8‑144)

DistScaleFactor = Clip3( −4096, 4095, ( tb \* tx + 32 ) >> 6 ) (8‑145)

mvLXB =Clip3( −8192, 8191.75, Sign( DistScaleFactor \* mvLXA ) \*   
 ( (Abs( DistScaleFactor \* mvLXA ) + 127 ) >> 8 ) ) (8‑146)

where td and tb are derived as

td = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListListB( refIdxB ) ) ) (8‑147)

tb = Clip3( −128, 127, PicOrderCntVal – PicOrderCnt( RefPicListLX( refIdxLX ) ) ) (8‑148)