

# Slice buffer design

## Table of Contents

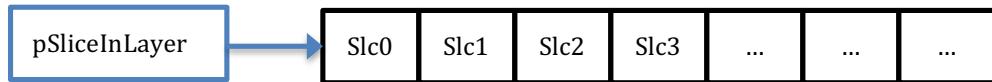
<b>1. OVERVIEW FOR NEW DESIGN .....</b>	<b>2</b>
1.1. ORIGIN DESIGN .....	2
1.1.1. <i>Single thread</i> .....	2
1.1.2. <i>Multi thread</i> .....	3
I.....	3
1.2 NEW DESIGN IN REVIEW.....	4
1.2.1 <i>Single thread</i> .....	4
1.2.2. <i>Multi-thread</i> .....	5
<b>2. SLICE BUFFER AND THREAD.....</b>	<b>6</b>
2.1 BEFORE ENCODING ONE LAYER.....	6
2.2. NORMAL CASE FOR THREAD INDEX AND SLICE BUFFER INDEX.....	7
2.3. SLICE NUM NOT THE SAME AMONG THREADS .....	8
2.4. DIFFERENT THREAD INDEX IN THE SAME SLICE BUFFER .....	10
2.5. SLICE INDEX OUT-OF-ORDER CASE.....	11
2.6. DYNAMIC SLICE MODE CASE1 .....	12
2.7. DYNAMIC SLICE MODE CASE2 .....	13

# 1. Overview for new design

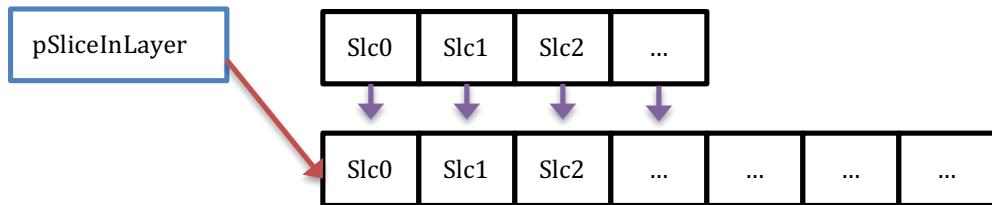
## 1.1. Origin design

```
SSlice* pSliceInLayer // slice buffer for all slices in layer
```

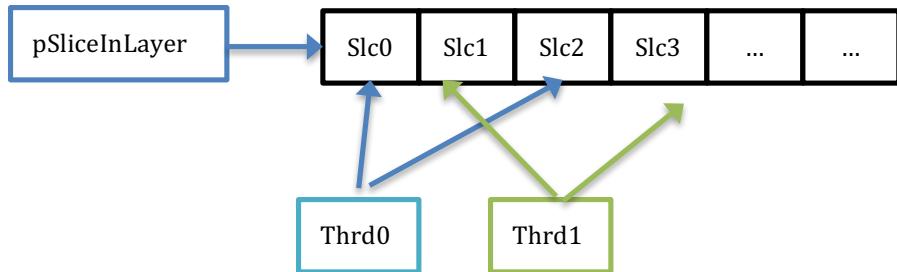
### 1.1.1. Single thread



**realloc** when current slice index larger than max slice num

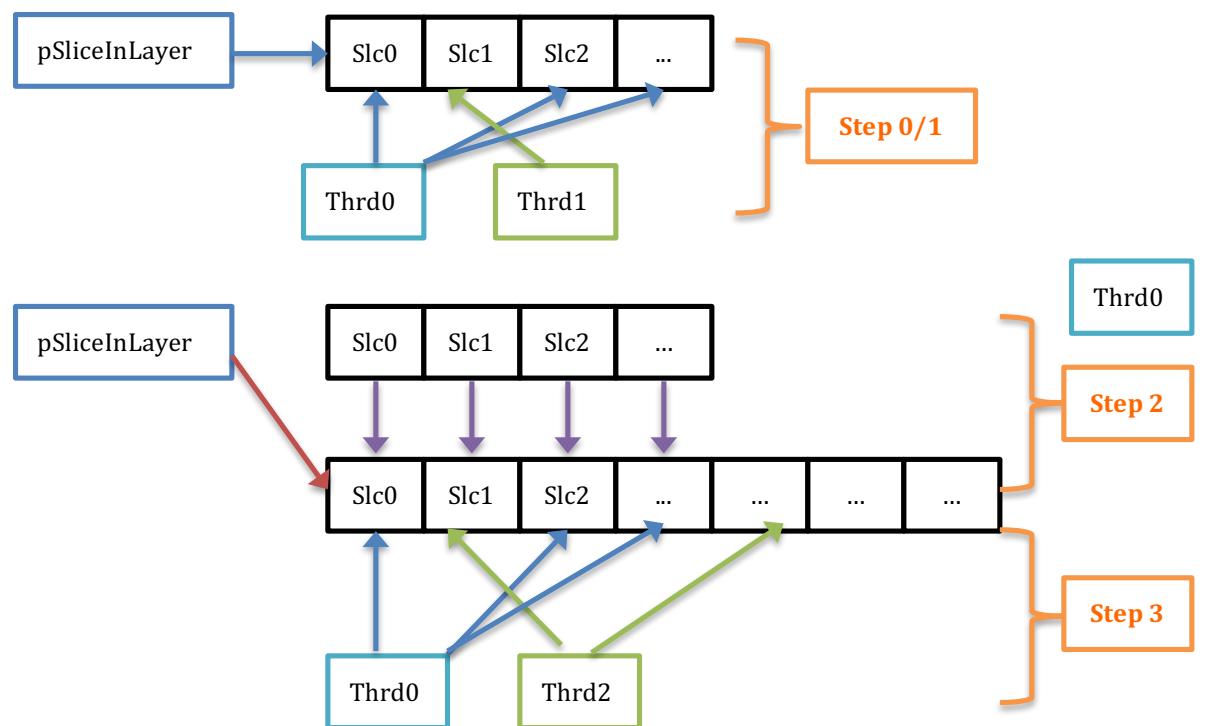


### 1.1.2. Multi thread



**realloc** when current slice index larger than max slice num ,

- step 0: thread[0] detect that current slice index larger than max slice num;
- step 1: thread[0] need to wait thread[1] completed current slice encoding task
- step 2: thread[1] stop slice encoding and thread[0] reallocate slice buffer,
- step 3: thread[0]/thread[1] start to encode new slice

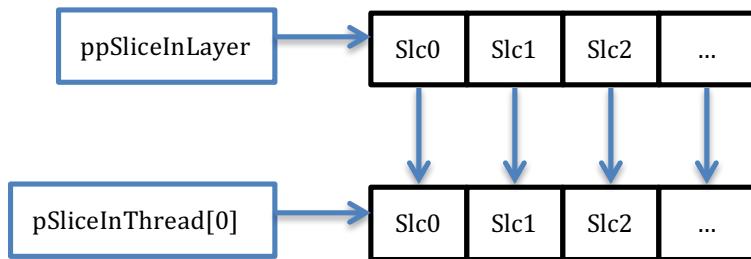


i.

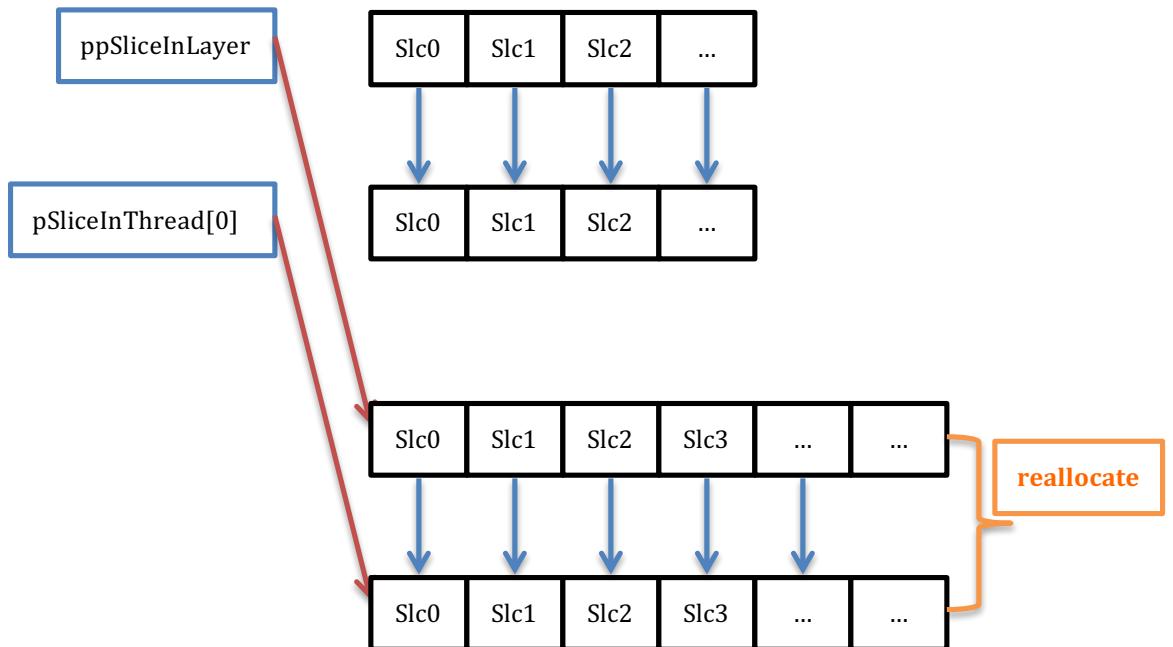
## 1.2 New design in review

```
SSlice* pSliceInLayer; //will be removed and replaced by  
pSliceInThread[]  
SSlice** ppSliceInLayer; // point to actual slice buffer  
//based on slice index  
SSlice* pSliceInThread[MaxThreadNum]; // actual slice buffer
```

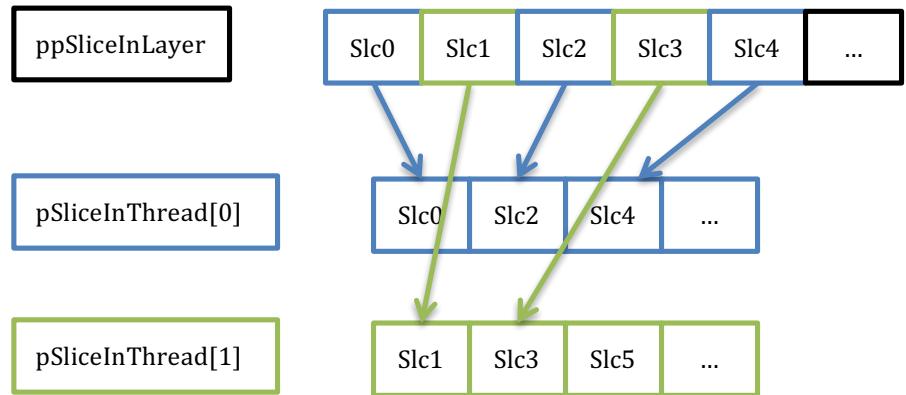
### 1.2.1 Single thread



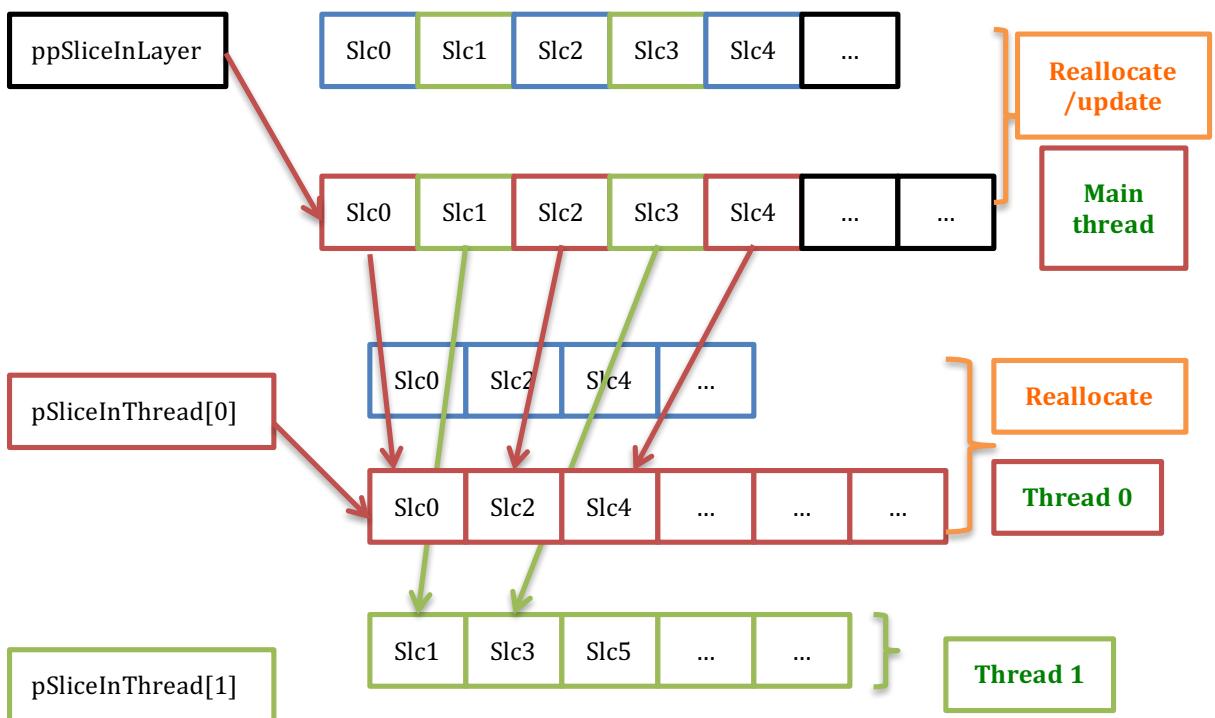
**reallocate** when current slice index larger than max slice num



### 1.2.2. Multi-thread



**for reallocate**, each thread will do it independently, and will update `ppSliceInLayer` by **main thread** when all slices in layer are encoded.



## 2. SLICE BUFFER AND THREAD

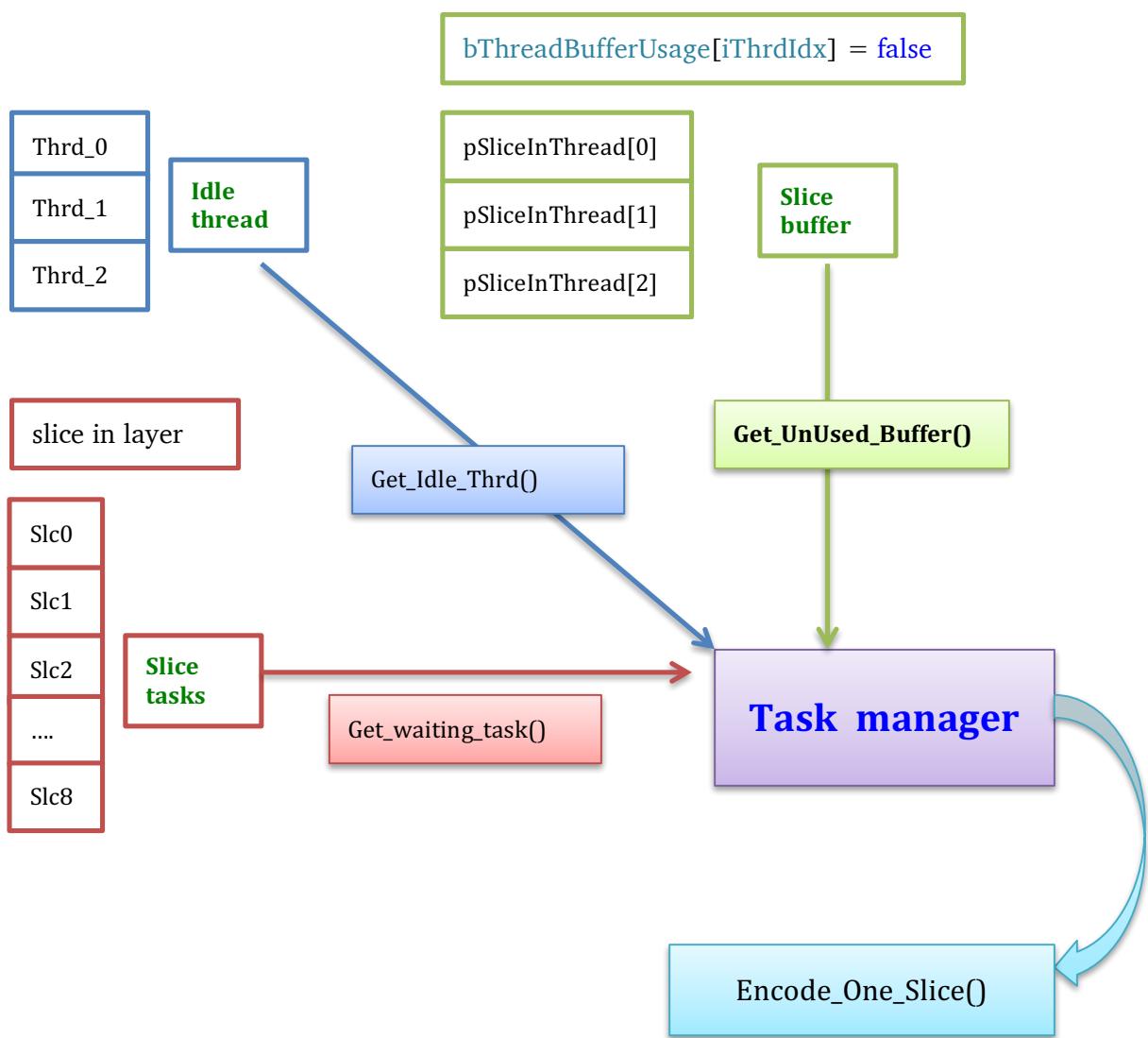
### 2.1 Before encoding one layer

the status of slice buffer and thread :

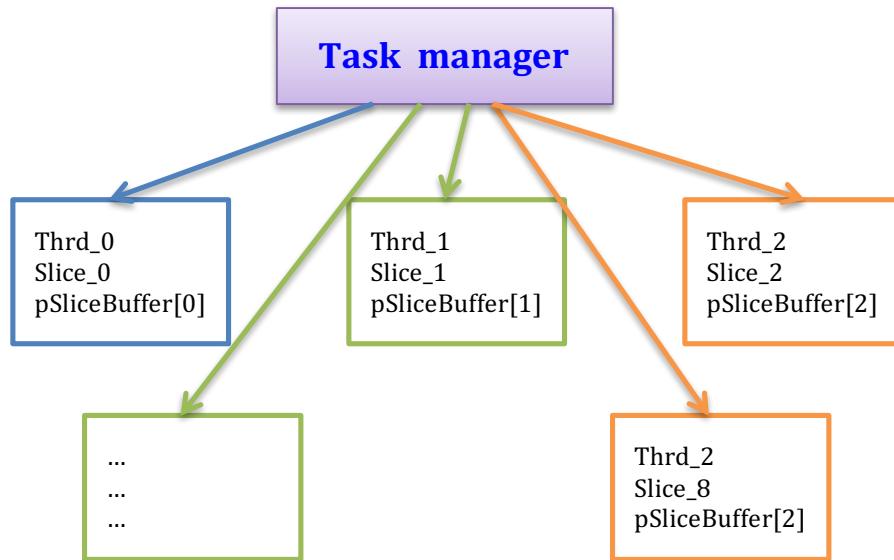
example:

thread: 3 threads

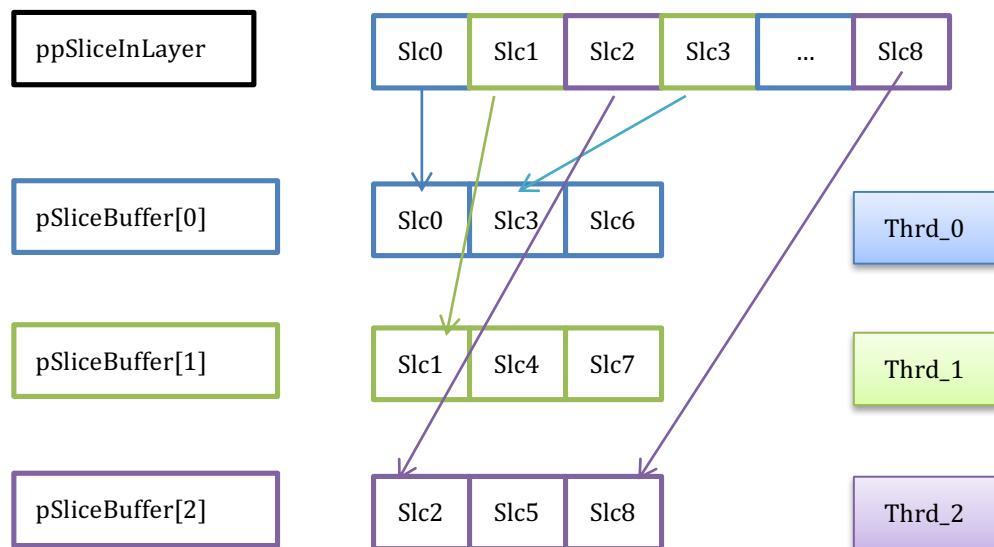
slices: 9 slices in layer



## 2.2. Normal case for thread index and slice buffer index

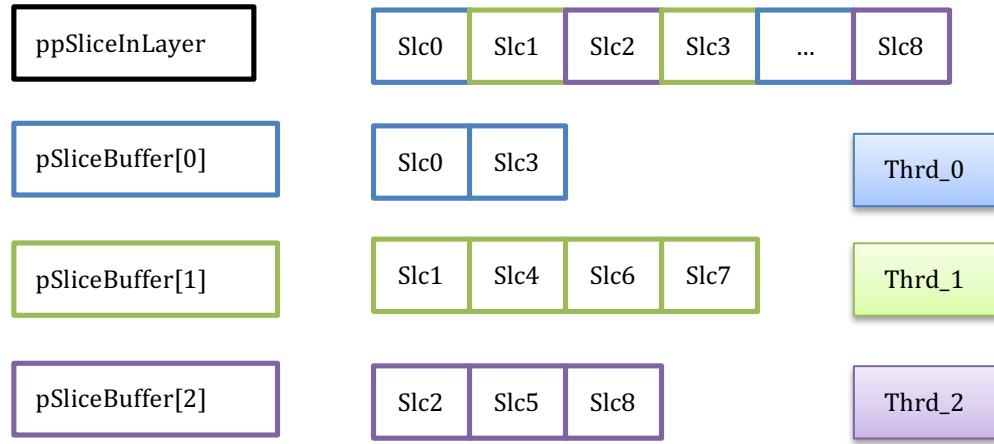


Final map for thread index and slice buffer index, slice index

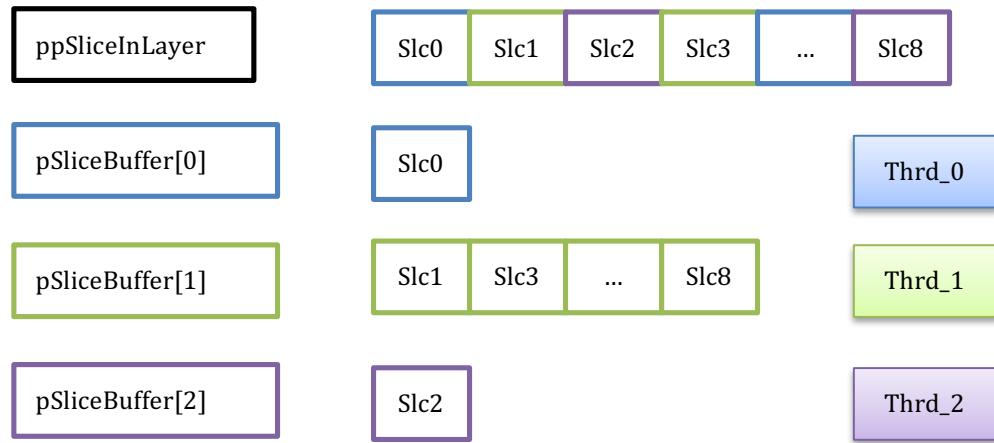


## 2.3. Slice num not the same among threads

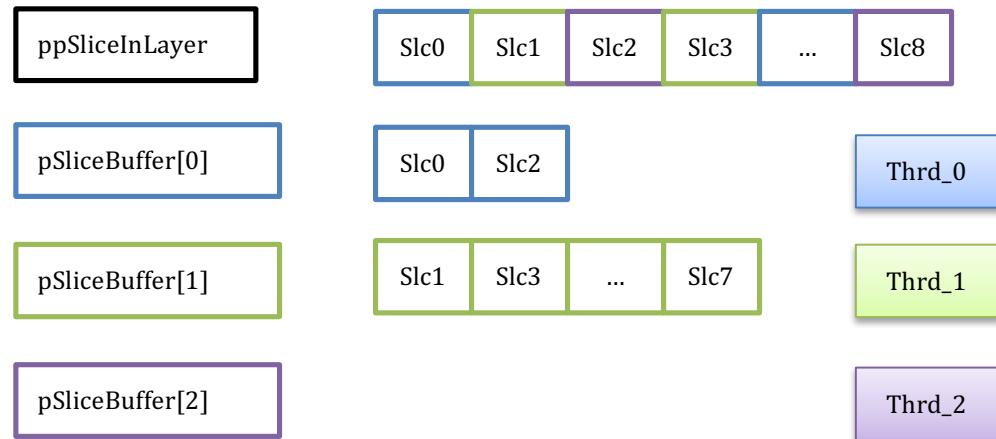
case 1:



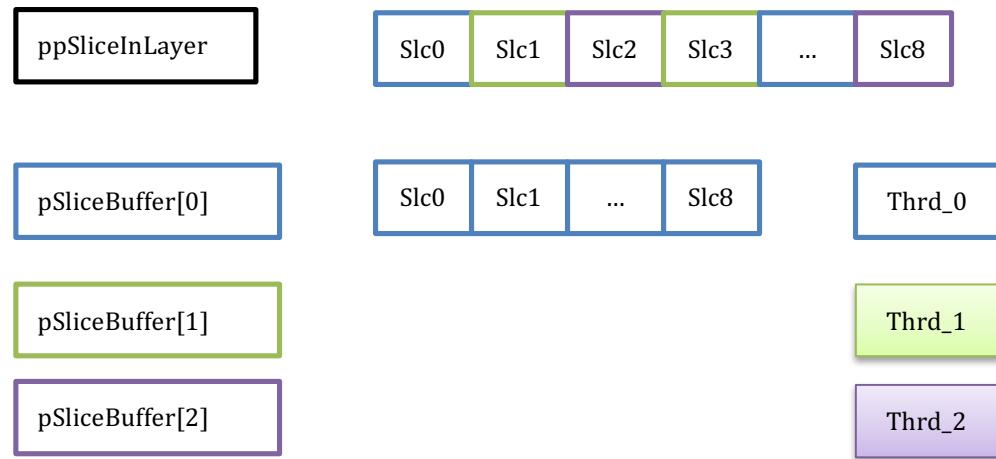
case 2:



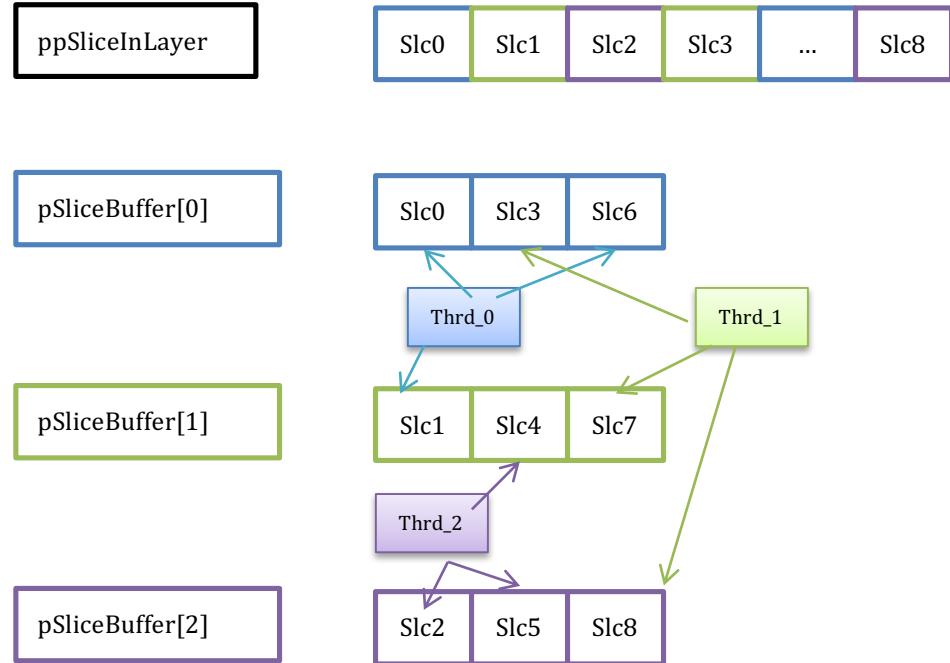
Corner case 1, no encoded slice for one thread:



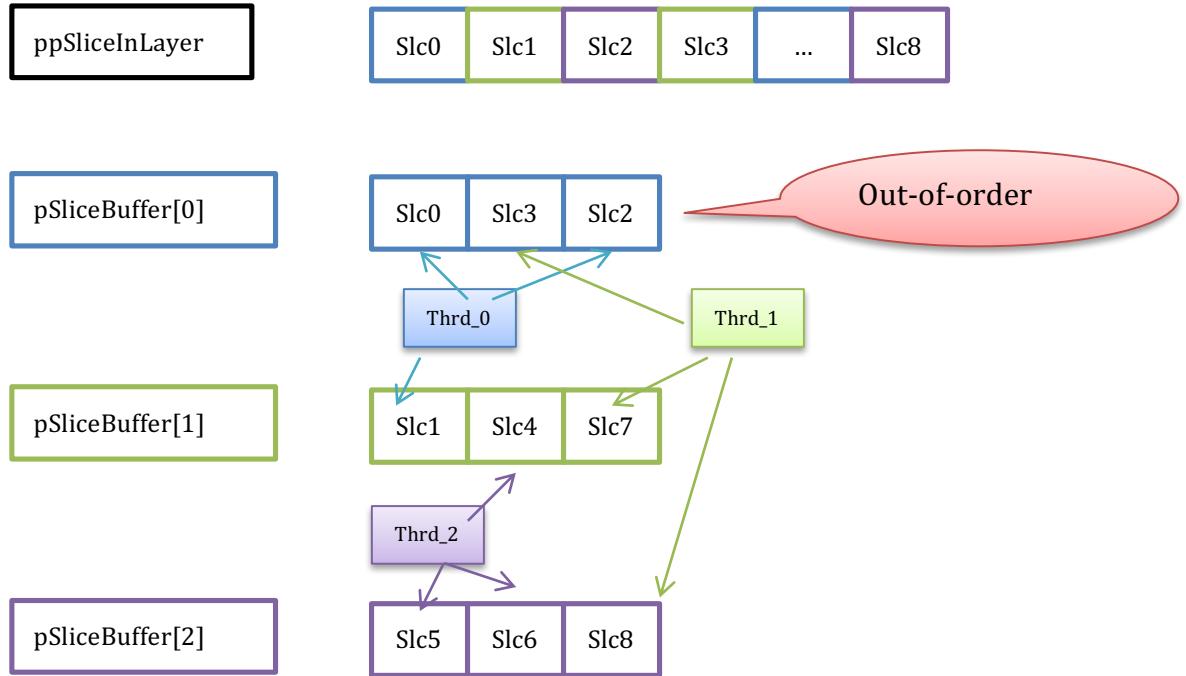
Corner case 2, all slices encoded by one thread :



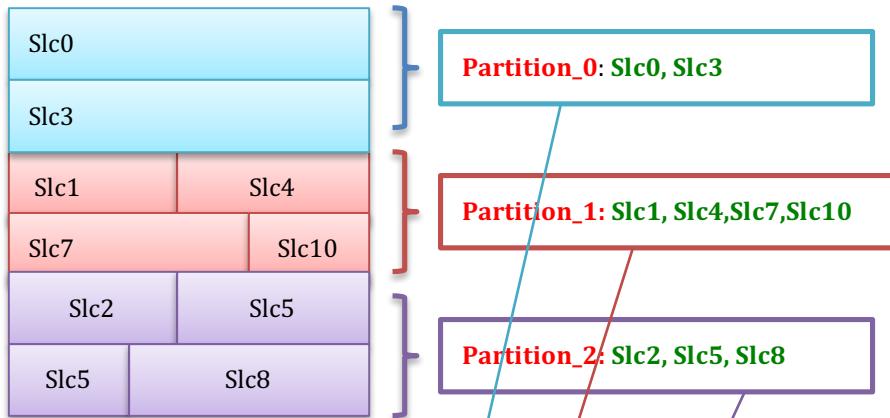
## 2.4. Different thread index in the same slice buffer



## 2.5. Slice index out-of-order case



## 2.6. Dynamic slice mode case1



`Slice_Index = PartitonID + Partition_Num * Slice_Index_InPartition`

Example:  $\text{Slc7} = 1 + 3 * 2$

$\text{Slc5} = 2 + 3 * 1$

`Slice_Index_InLayer = PartitonOffset[Partiton_ID] + Slice_Index / Partition_Num`

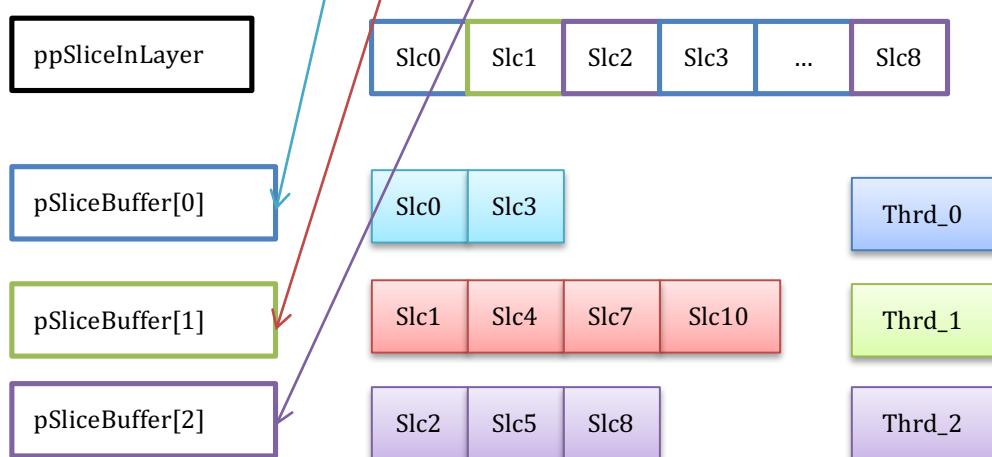
Here in example, `PartitonOffset[0] = 0;`

`PartitonOffset[1] = 0 + 2 = 2;`

`PartitonOffset[2] = 0 + 2 + 4 = 6;`

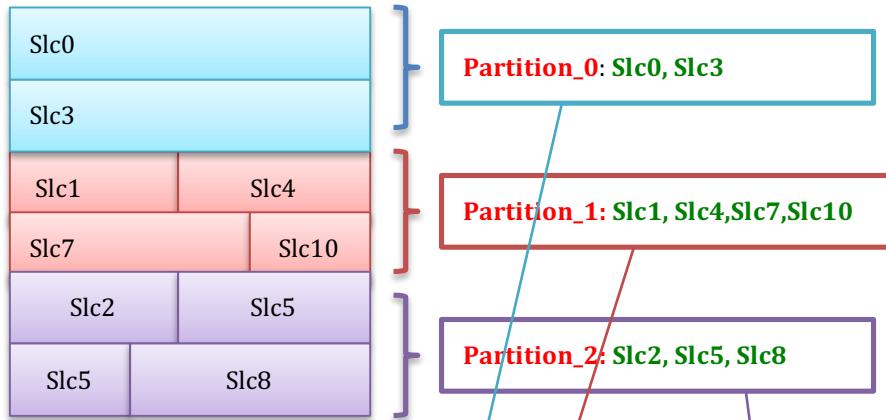
$\text{Slc7} = \text{PartitonOffset}[1] + 7 / 3 = 2 + 2 = 4;$

Which `ppSliceInLayer[4] = Slc7`



## 2.7. Dynamic slice mode case2

**Thread\_0 encoded two partitions  
while no partition for Thread\_2**



$$\text{Slice\_Index} = \text{PartitonID} + \text{Partition\_Num} * \text{Slice\_Index\_InPartition}$$

$$\text{Example: } \text{Slc7} = 1 + 3 * 2$$

$$\text{Slc5} = 2 + 3 * 1$$

$$\text{Slice\_Index\_InLayer} = \text{PartitonOffset}[\text{Partiton\_ID}] + \text{Slice\_Index} / \text{Partition\_Num}$$

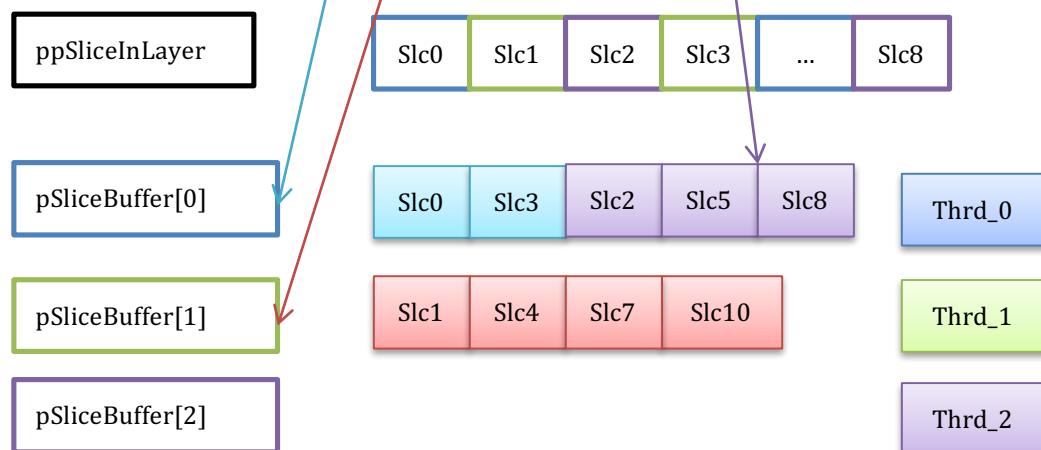
$$\text{Here in example, PartitonOffset[0] = 0;}$$

$$\text{PartitonOffset[1] = } 0 + 2 = 2;$$

$$\text{PartitonOffset[2] = } 0 + 2 + 4 = 6;$$

$$\text{Slc7} = \text{PartitonOffset[1]} + 7 / 3 = 2 + 2 = 4;$$

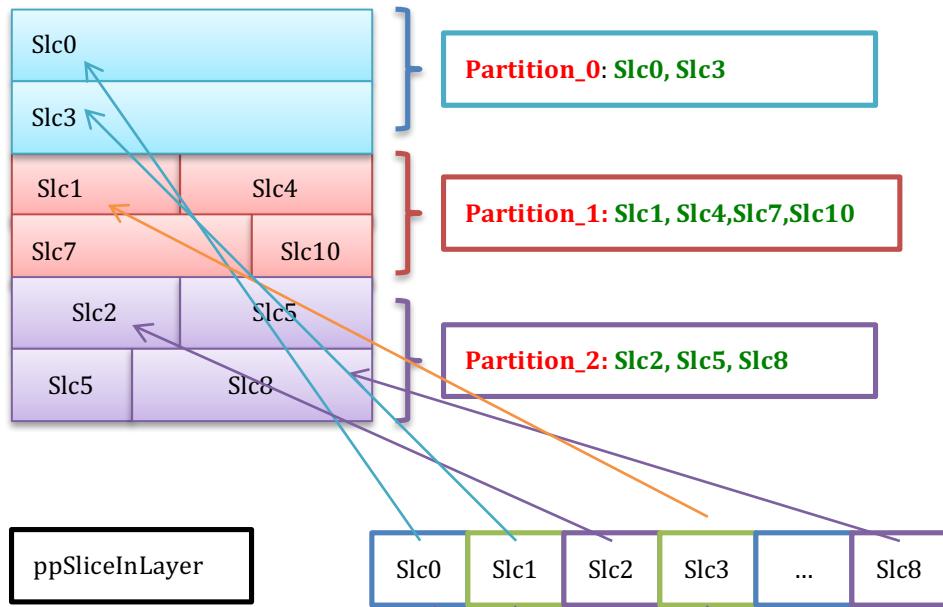
Which ppSliceInLayer[4] = Slc7



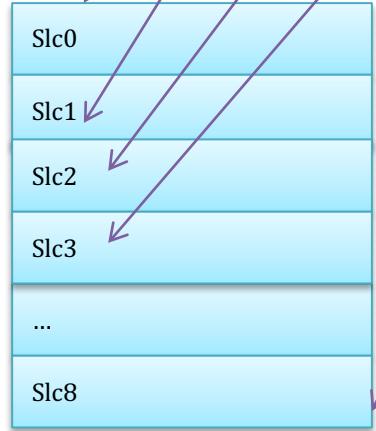
### 3. Slice Buffer Update/Reorder

#### 3.1. Slice index in different slice mode.

Dynamic slice mode

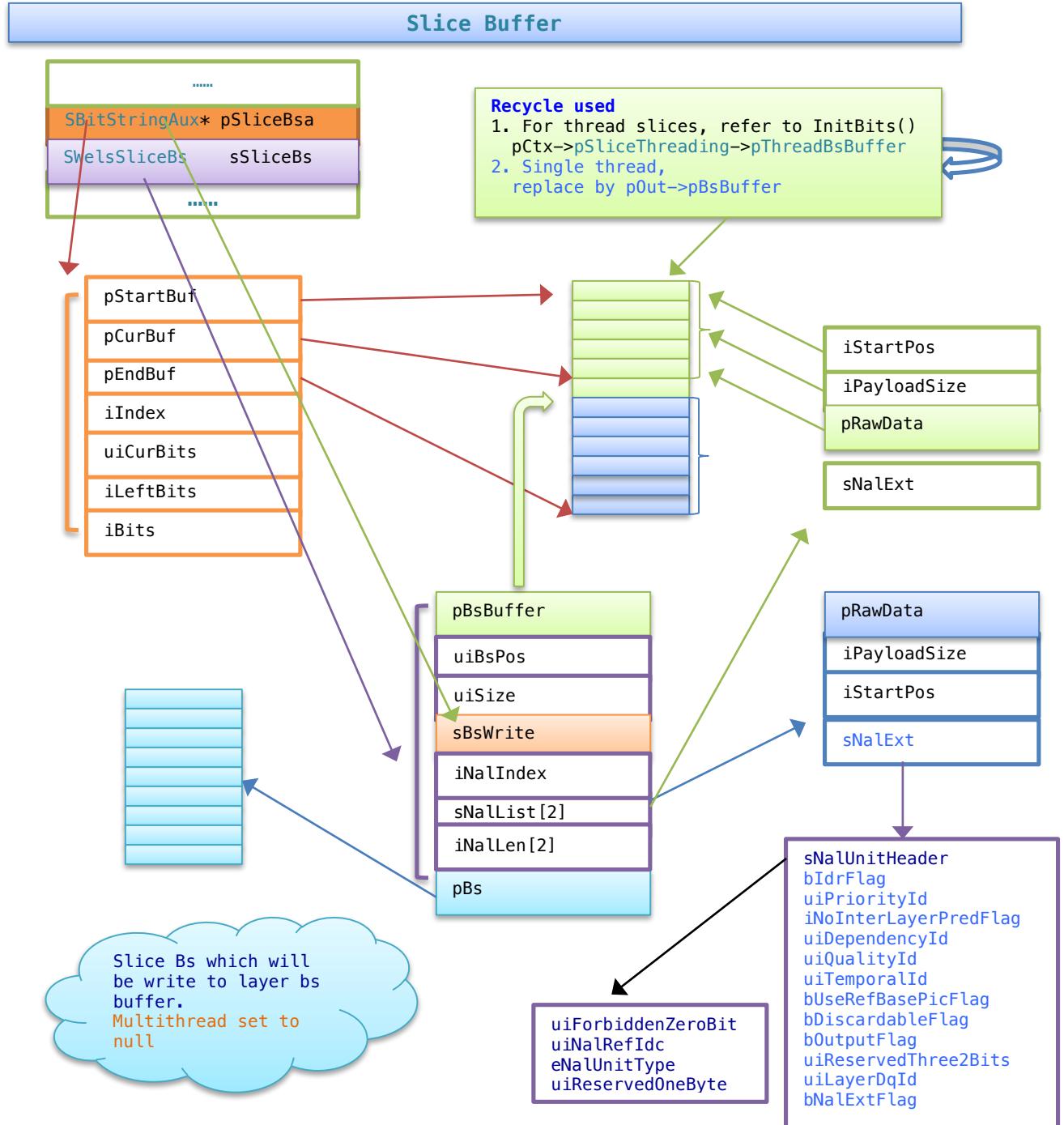


Non-dynamic slice mode



## 4. Bs buffer design

### 4.1. Slice Bs buffer



#### 4.1.1. init

```
InitSliceBsBuffer()
    pSlice->sSliceBs.uiSize = iMaxSliceBufferSize;
    (pCtx)->iSliceBufferSize[kiDlayerIndex];
    (*ppCtx)->p0ut->uiSize = iCountBsLen;

InitBits (&m_pSliceBs->sBsWrite,
          m_pSliceBs->pBsBuffer,
          m_pSliceBs->uiSize);

if (bIndependenceBsBuffer) {
    pSlice->pSliceBsa      = &pSlice->sSliceBs.sBsWrite;
    pSlice->sSliceBs.pBs    =
        (uint8_t*)pMa->WelsMalloc (iMaxSliceBufferSize, "SliceBs");

} else {
    pSlice->pSliceBsa      = pCtx->p0ut->sBsWrite;
    pSlice->sSliceBs.pBs    = NULL;
}
```

#### 4.1.2 encode one slice

```
//set recycle bs buffer to m_pSliceBs->pBsBuffer
SetOneSliceBsBufferUnderMultithread (m_pCtx,
                                      m_iThreadId, m_iSliceIdx);
//init m_pSliceBs->sBsWrite, and set its start buffer to recycle buffer
InitBits (&m_pSliceBs->sBsWrite,
          m_pSliceBs->pBsBuffer,
          m_pSliceBs->uiSize);

WelsLoadNalForSlice (m_pSliceBs,
                     m_eNalType,
                     m_eNalRefIdc);
WelsCodeOneSlice (m_pCtx, m_iSliceIdx,
                  m_eNalType);

WelsUnloadNalForSlice (m_pSliceBs);

WriteSliceBs (m_pCtx,
              m_pSliceBs,
              m_iSliceIdx,
              m_iSliceSize)

iLayerSize = AppendSliceToFrameBs (pCtx, pLayerBsInfo, iSliceCount);
```

## 4.2. Frame bs buffer

### 4.2.1 init

```
iVclLayersBsSizeCount += iLayerBsSize;
iCountBsLen    = iNonVclLayersBsSizeCount + iVclLayersBsSizeCount;
(*ppCtx)->p0ut->uiSize = iCountBsLen;

(*ppCtx)->p0ut->pBsBuffer =
    (uint8_t*)pMa->WelsMallocz (iCountBsLen, "p0ut->pBsBuffer");
(*ppCtx)->pFrameBs = (uint8_t*)pMa->WelsMalloc (iTotalLength,
"pFrameBs");
(*ppCtx)->iFrameBsSize = iTotalLength;

    InitBits (&pEncCtx->p0ut->sBsWrite, pEncCtx->p0ut->pBsBuffer,
pEncCtx->p0ut->uiSize)

if (bIndependenceBsBuffer) {
    pSlice->pSliceBsa      = &pSlice->sSliceBs.sBsWrite;
    pSlice->sSliceBs.pBs    =
        (uint8_t*)pMa->WelsMalloc (iMaxSliceBufferSize, "SliceBs");

} else {
    pSlice->pSliceBsa      = pCtx->p0ut->sBsWrite;
    pSlice->sSliceBs.pBs    = NULL;
}
```

#### 4.2.2. WelsWriteOneSPS()

```
WelsLoadNal (pCtx->pOut,
              NAL_UNIT_SPS,
              NRI_PRI_HIGHEST);

WelsWriteSpsNal (&pCtx->pSpsArray[kiSpsIdx],
                  &pCtx->pOut->sBsWrite,
                  ...);
WelsUnloadNal (pCtx->pOut);

WelsEncodeNal (&pCtx->pOut->sNalList[iNal],
               NULL,
               pCtx->iFrameBsSize - pCtx->iPosBsBuffer,
               pCtx->pFrameBs + pCtx->iPosBsBuffer,
               &iNalSize);
```

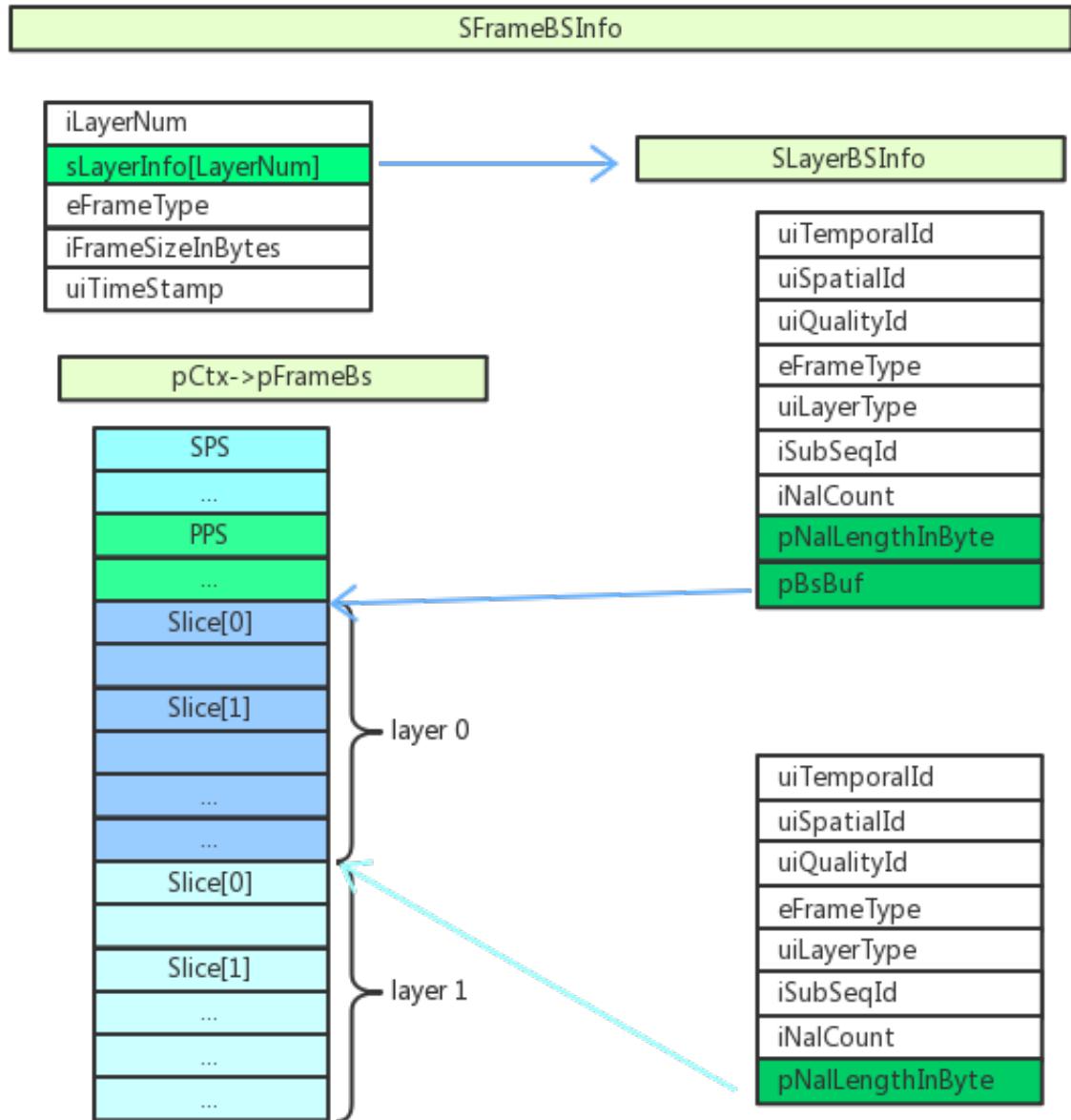
#### 4.2.3. Single thread, encode one slice

```
WelsLoadNal (pCtx->pOut,
              eNalType,
              eNalRefIdc);
WelsCodeOneSlice (pCtx,
                  iSliceIdx,
                  eNalType);

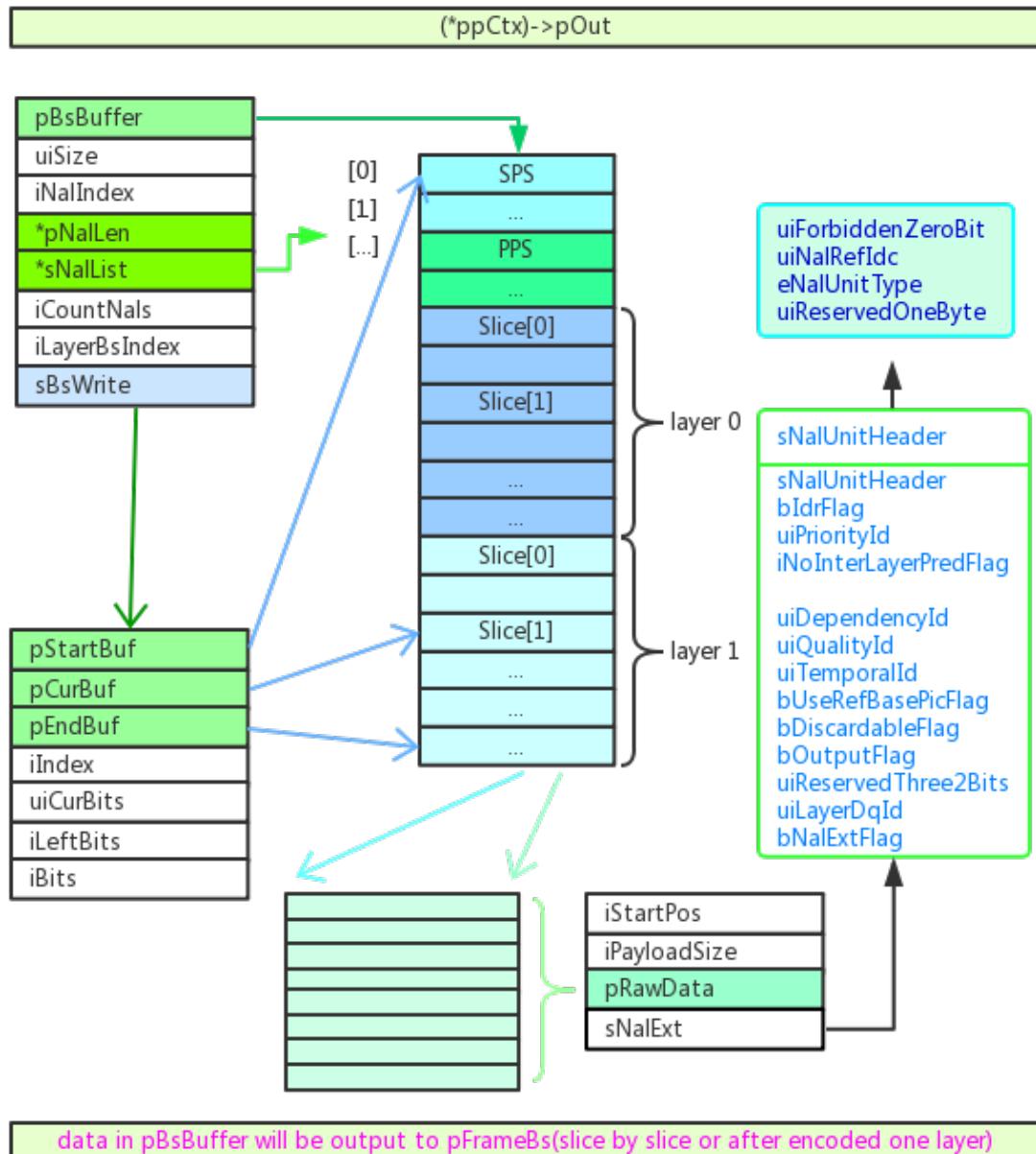
WelsUnloadNal (pCtx->pOut);

WelsEncodeNal (&pCtx->pOut->sNalList[pCtx->pOut->iNalIndex - 1],
               &pCtx->pCurDqLayer->sLayerInfo.sNalHeaderExt,
               pCtx->iFrameBsSize - pCtx->iPosBsBuffer,
               pCtx->pFrameBs + pCtx->iPosBsBuffer,
               &pLayerBsInfo->pNalLengthInByte[iNalIdxInLayer]);
```

#### 4.2.4. frame bs and layer bs



#### 4.2.5. pCtx-pOut



## 4.3. Slice layer buffer reallocate

### 4.3.1. origin design

```
?? max slice num limitation in JM?? Standard??
int32_t FrameBsRealloc (sWelsEncCtx* pCtx,
                        SFrameBSInfo* pFrameBsInfo,
                        SLayerBSInfo* pLayerBsInfo)

int32_t ReallocSliceBuffer (sWelsEncCtx* pCtx){

    ReallocateSliceList()
    for (iSliceIdx = 0; iSliceIdx < iMaxSliceNumNew; iSliceIdx++) {
        pCurLayer->ppSliceInLayer[iSliceIdx] =
            pCurLayer->sSliceThreadInfo.pSliceInThread[0] + iSliceIdx;
    }
}

int32_t DynSliceRealloc (sWelsEncCtx* pCtx,
                        SFrameBSInfo* pFrameBsInfo,
                        SLayerBSInfo* pLayerBsInfo) {

    iRet = FrameBsRealloc (pCtx, pFrameBsInfo, pLayerBsInfo);
    iRet = ReallocSliceBuffer (pCtx);
}

int32_t ReallocateSliceList (sWelsEncCtx* pCtx,
                            SSliceArgument* pSliceArgument,
                            SSlice*& pSliceList,
                            const int32_t kiMaxSliceNumOld,
                            const int32_t kiMaxSliceNumNew){
    for(){
        iRet = InitSliceBsBuffer (pSlice,
                                  &pCtx->pOut->sBsWrite,
                                  bIndependenceBsBuffer,
                                  iMaxSliceBufferSize, pMA);
        iRet = AllocateSliceMBBuffer (pSlice, pMA);
        iRet = InitSliceMBInfo (pSliceArgument,
                               pSlice,
                               pCurLayer->iMbWidth,
                               pCurLayer->iMbHeight);

        InitSliceHeadWithBase (pSlice, pBaseSlice);
        InitSliceRefInfoWithBase (pSlice, pBaseSlice,
                                pCtx->iNumRef0);

        iRet = InitSliceRC (pSlice,
                           pCtx->iGlobalQp, iBitsPerMb);
    }
}

int32_t ReallocateSliceInThread (sWelsEncCtx* pCtx,
                                SDqLayer* pDqLayer,
                                const int32_t kiDlayerIdx,
                                const int32_t kiThreadId);
```

#### 4.3.2. new design

```
in layer level, reallocate thread by thread;  
ReallocSliceBuffer() will be removed later  
int32_t ReallocateSliceInThread (sWelsEncCtx* pCtx,  
                                SDqLayer* pDqLayer,  
                                const int32_t kiDlayerIdx,  
                                const int32_t kiThreadId);  
  
after encoded one layer, reallocate layer level buffer if need.  
if (iSliceIdx >= (pSliceCtx->iMaxSliceNumConstraint - kiSliceIdxStep)){}  
  
int32_t FrameBsRealloc (sWelsEncCtx* pCtx,  
                        SFrameBSInfo* pFrameBsInfo,  
                        SLayerBSInfo* pLayerBsInfo);  
  
AppendSliceToFrameBs (pCtx, pLayerBsInfo, iSliceCount);
```

#### **TODO:**

```
//will update slice by slice after encoded one slice  
sSliceThreadInfo.iEncodedSliceNumInThread[iThreadId]
```

```
//will update layer maxsliceNum  
pCurLayer->sSliceEncCtx.iMaxSliceNumConstraint  
// ++ pSliceCtx->iSliceNumInFrame
```

```
//will remove below check, no need to check in MB level but slice level  
const bool kbSliceNumNotExceedConstraint =  
    pSliceCtx->iSliceNumInFrame <  
    pSliceCtx->iMaxSliceNumConstraint;  
const bool kbSliceIdxNotExceedConstraint =  
    ((int) pCurSlice->uiSliceIdx + kiActiveThreadsNum) <  
    pSliceCtx->iMaxSliceNumConstraint;  
const bool kbSliceNumReachConstraint =  
    (pSliceCtx->iSliceNumInFrame ==  
    pSliceCtx->iMaxSliceNumConstraint);
```

#### 4.4. Slice buffer update

```
int32_t ReOrderSliceInLayer (SDqLayer* pCurLayer,
                           const int32_t kiThreadNum,
                           const int32_t kiPartitionNum);

int32_t CheckAllSliceBuffer(SDqLayer* pCurLayer,
                           const int32_t kiCodedSliceNum);

int32_t SliceLayerInfoUpdate (sWelsEncCtx* pCtx,
                           const int32_t kiDlayerIndex);
```

## 4.5 slice encode and MB info update

```
int32_t WelsISliceMdEncDynamic (sWelsEncCtx* pEncCtx,
                                SSlice* pSlice) {
    pCurLayer->pLastCodedMbIdxOfPartition[kiPartitionId] =
        iCurMbIdx -1;
    ++ pCurLayer->pNumSliceCodedOfPartition[kiPartitionId];
}

pCurLayer->sSliceEncCtx.iMaxSliceNumConstraint
// ++ pSliceCtx->iSliceNumInFrame

//MB—Slice—partition map
pSliceCtx->pOverallMbMap[iCurMbIdx]
pEncCtx->iActiveThreadsNum;
pEncCtx->pCurDqLayer->pLastMbIdxOfPartition[kiPartitaionId];

const int32_t kiPartitaionId = pCurSlice->uiSliceIdx %
    kiActiveThreadsNum;
const int32_t kiLastMbIdxInPartition =
    pEncCtx->pCurDqLayer->pLastMbIdxOfPartition[kiPartitaionId];
const bool kbCurMbNotFirstMbOfCurSlic = ((iCurMbIdx > 0) &&
    (pSliceCtx->pOverallMbMap[iCurMbIdx] ==
     pSliceCtx->pOverallMbMap[iCurMbIdx - 1]));
const bool kbCurMbNotLastMbOfCurPartition = iCurMbIdx <
    kiLastMbIdxInPartition;
```

## 4.6. dynamic slice, slice boundary strategy

### 4.6.1. relate function

```
int32_t WelsISliceMdEncDynamic (sWelsEncCtx* pEncCtx, SSlice* pSlice)

int32_t WelsMdInterMbLoopOverDynamicSlice (sWelsEncCtx* pEncCtx,
                                            SSlice* pSlice,
                                            void* pWelsMd,
                                            const int32_t kiSliceFirstMbXY)

void AddSliceBoundary (sWelsEncCtx* pEncCtx,
                      SSlice* pCurSlice, SSliceCtx* pSliceCtx,
                      SMB* pCurMb,
                      int32_t iFirstMbIdxOfNextSlice,
                      const int32_t kiLastMbIdxInPartition) {

    const int32_t kiSliceIdxStep = pEncCtx->iActiveThreadsNum;
    uint16_t     iNextSliceIdc = iCurSliceIdc + kiSliceIdxStep;

    //update cur pSlice info
    pCurSlice->sSliceHeaderExt.uiNumMbsInSlice = 1 +
        iCurMbIdx - pCurSlice->sSliceHeaderExt.sSliceHeader.iFirstMbInSlice;

    //pNextSlice pointer/initialization
    pNextSlice = pCurLayer->ppSliceInLayer[ iNextSliceIdc ];

    ppSliceInLayer[ iNextSliceIdc ]->sSliceHeaderExt.sSliceHeader.iFirstMbInSlice =
        iFirstMbIdxOfNextSlice;

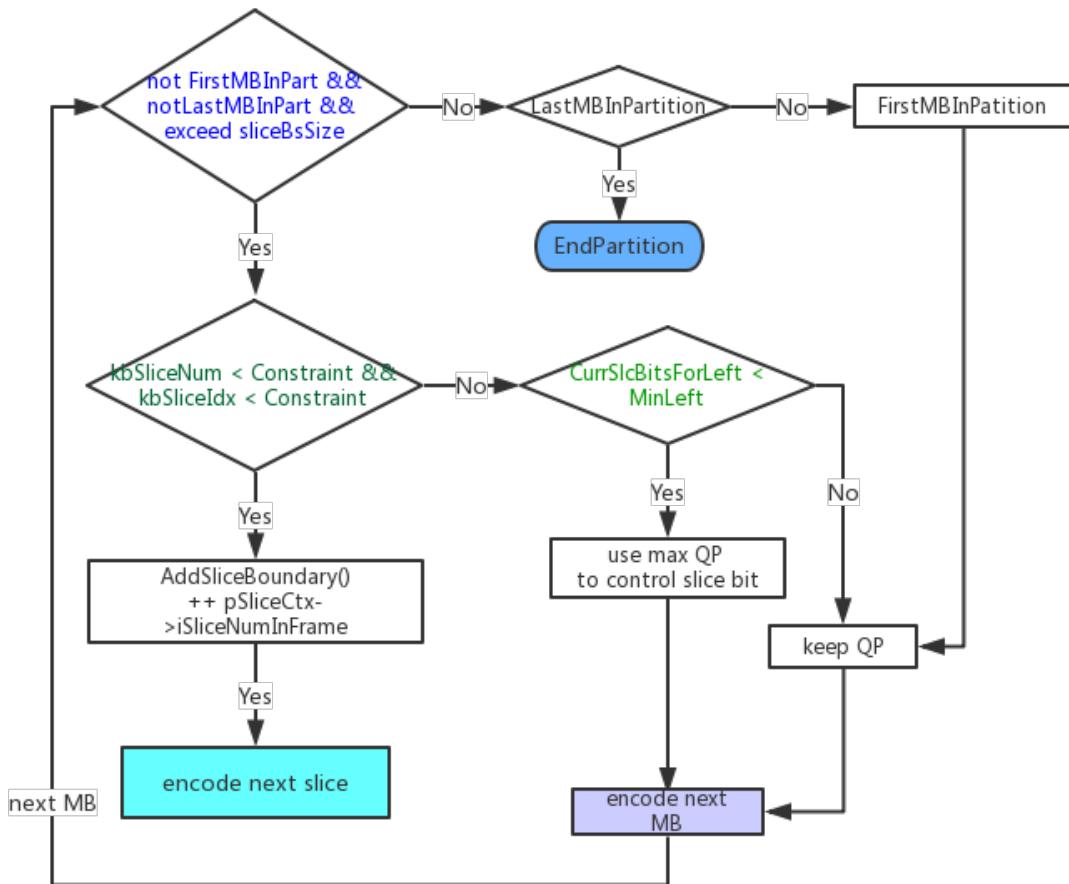
    WelsSetMemMultiplebytes_c (pSliceCtx->pOverallMbMap + iFirstMbIdxOfNextSlice,
                              iNextSliceIdc,
                              (kiLastMbIdxInPartition - iFirstMbIdxOfNextSlice + 1),
                              sizeof (uint16_t));

    // update left/right/up ect available info, one row only
    UpdateMbNeighbor(pCurDq, &pMbList[iIdx], kiMbWidth, uiSliceIdc);

}

//ToDo:
pNextSlice = pCurLayer->ppSliceInLayer[ iNextSliceIdc ];
will be replace by pSliceInThread[]
```

#### 4.6.2. flow chart



## 4.7. ppSliceInLayer init/update functions

### 4.7.1 ppSliceInLayer info init/update before encode all slice in one layer

#### before encode one sequence

```
---->pPtrEnc->SetOption()
      ---->WelsRcInitFuncPointers()
          ---->pRcf->pfWelsRcPictureInit = WelsRcPictureInitGom()
              ---->RcInitGomParameters() //ppSlice, init function pointer only
          ---->pRcf->pfWelsRcPictureInit = WelRcPictureInitScc;           // no ppSlice
          ---->pRcf->pfWelsRcPictureInit = WelRcPictureInitBufferBasedQp // no ppSlice
          ---->pRcf->pfWelsRcPictureInit = WelsRcPictureInitDisable;       // no ppSlice

          ---->pRcf->pfWelsRcPictureInfoUpdate = WelsRcPictureInfoUpdateGom;
              ----> RcUpdatePictureQpBits (pEncCtx, iCodedBits); //ppSlice, init function only
                  ---->RcInitSliceInformation (pEncCtx); //ppSlice

          ---->pRcf->pfWelsRcPictureInfoUpdate = WelsRcPictureInfoUpdateGomTimeStamp;
              ----> RcUpdatePictureQpBits (pEncCtx, iCodedBits); //ppSlice, init function only
          ---->pRcf->pfWelsRcPictureInfoUpdate = WelsRcPictureInfoUpdateScc; // no ppSlice

          ---->pRcf->pfWelsRcMbInit = WelsRcMbInitGom;
              ---->WelsRcMbInitGom() //ppSlice
          ---->pRcf->pfWelsRcMbInit = WelsRcMbInitScc; // no ppSlice

          ---->pRcf->pfWelsRcMbInfoUpdate = WelsRcMbInfoUpdateGom;
              ---->WelsRcMbInfoUpdateGom() //ppSlice

---->WelsInitEncoderExt()
    ----> RequestMemorySvc (&pCtx, pExistingParasetList) //multi thread
        ---->InitDqLayers (ppCtx, pExistingParasetList);
            ---->InitSliceInLayer() //ppSlice
```

#### Before encode one frame

```
EncodeFrameInternal()
-->WelsEncoderEncodeExt()
  before encode all slice in layer
  ---->WelsUpdateRefSyntax (pCtx, pParamInternal->iPOC, eFrameType) // ppSlice
  ---->PreprocessSliceCoding (pCtx);
      ---->DeblockingFilterSliceAvbase() // function pointer only, no processing
      ---->pFuncList->pfUpdateFMESwitch = UpdateFMESwitch; //function pointer
          ---->CountFMECostDown() //ppSlice

  ---->WelsInitCurrentDlayerMltslc (pCtx, iPclPartitionNum) //Dynamic slice mode only
      ---->UpdateSliceEncCtxWithPartition() // ppSlice Dynamic slice mode only

  ---->WelsInitCurrentLayer() // ppSlice
  ---->PrefetchReferencePicture (pCtx, eFrameType); // ppSlice update reference picture
  ---->pCtx->pFuncList->pfRc.pfWelsRcPictureInit (pCtx, pFbi->uiTimeStamp)
      ---->RcInitGomParameters() //ppSlice
      ---->RcInitSliceInformation (pEncCtx); //ppSlice

  ---->pCtx->pReferenceStrategy->MarkPic();
      ---->WelsMarkPic (m_pEncoderCtx) //ppSlice
      ---->WelsMarkPicScreen (m_pEncoderCtx); //ppSlice

  ---->AdjustEnhanceLayer (pCtx, iCurDid); //ppSlice FixedSlice Mode only
      ---->NeedDynamicAdjust() //ppSlice
      ---->DynamicAdjustSlicing() //ppSlice
```

```
---->int32_t FiredSliceThreads (sWelsEncCtx* pCtx, ...) // ppSlice for Partition first MB info  
    ---->SetOneSliceBsBufferUnderMultithread() //ppSlice  
---->WelsErrorType CWelsSliceEncodingTask::InitTask() //ppSlice
```

#### 4.7.2. ppSliceInLayer buffer update functions during encode all slice in layer

##### during encoding one slice

```
EncodeFrameInternal()
-->WelsEncoderEncodeExt()
    ---->WelsCodeOnePicPartition
    ---->CWelsConstrainedSizeSlicingEncodingTask::ExecuteTask() //ppSlice
        ---->int32_t ReallocSliceBuffer (sWelsEncCtx* pCtx)

    ---->int32_t WelsCodeOneSlice () // ppSlice
        ---->WelsSliceHeaderExtInit (pEncCtx, pCurLayer, pCurSlice); //ppSlice
            ---->WelsGetFirstMbOfSlice() //ppSlice

        ---->WelsCodePSlice()
        ----> WelsCodePOverDynamicSlice()
            ---->AddSliceBoundary() //ppSlice
        ---->WelsISliceMdEnc()
        ----> WelsISliceMdEncDynamic()
            ---->pEncCtx->pFuncList->pfRc(pfWelsRcMbInit (pEncCtx, pCurMb, pSlice)
                ---->WelsRcMbInitGom() //ppSlice
            ----> pEncCtx->pFuncList->pfRc(pfWelsRcMbInfoUpdate( );
                ---->WelsRcMbInfoUpdateGom() //ppSlice

    ---->void UpdateMbListNeighborParallel (SDqLayer* pCurDq..) //ppSlice, seem no call this funtion

    ---->void CWelsLoadBalancingSlicingEncodingTask::FinishTask() //ppSlice
```

#### 4.7.3. ppSliceInLayer buffer update functions after encode all slice in layer

```
EncodeFrameInternal()
-->WelsEncoderEncodeExt()
    after encode all slice in layer
---->int32_t SliceLayerInfoUpdate()
        ---->int32_t ReOrderSliceInLayer (SDqLayer* pCurLayer..)
        ---->static inline int32_t CheckAllSliceBuffer...}()

        ---->AppendSliceToFrameBs (pCtx, pLayerBsInfo, iSliceCount); //ppSlice, MT only
        ---->PerformDeblockingFilter()
            ----> DeblockingFilterFrameAvibase (pCurLayer, pEnc->pFuncList); //ppSlice
            ---->DeblockingFilterSliceAvibase() //ppSlice
---->pCtx->pFuncList->pfRc.pfWelsRcPictureInfoUpdate (pCtx, iLayerSize);
    ---->WelsRcPictureInfoUpdateGomTimeStamp
        ---->RcUpdatePictureQpBits() //ppSlice
    ---->WelsRcPictureInfoUpdateGom()
        ---->RcUpdatePictureQpBits() //ppSlice
---->pCtx->pFuncList->pfUpdateFMESwitch (pCtx->pCurDqLayer);
    ---->CountFMECostDown() //ppSlice

    ---->TrackSliceComplexities()
    ----> CalcSliceComplexRatio (pCtx->pCurDqLayer);

    after encode all slice in all layer
    ---->TrackSliceConsumeTime (pCtx, iDidList, iSpatialNum); //ppSlice
    ----> AdjustBaseLayer (pCtx) //ppSlice, FixedSlice Mode && multiLayer && MultiThread only
        ---->NeedDynamicAdjust() //ppSlice
        ---->DynamicAdjustSlicing() //ppSlice
```