Slice buffer design

Table of Contents

1. Overview for new design 2

1.1. Origin design 2

1.1.1. Single thread 2

1.1.2. Multi thread 3

i. 3

1.2 New design in review 4

1.2.1 Single thread 4

1.2.2. Multi-thread 5

2. Slice Buffer and thread 6

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

# Overview for new design

## Origin design

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

### 1.1.1. Single thread

pSliceInLayer

…

…

…

Slc3

Slc2

Slc1

Slc0

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

…

pSliceInLayer

Slc2

Slc1

Slc0

Slc0

Slc1

Slc2

…

…

…

…

### 1.1.2. Multi thread

pSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

…

Thrd0

Thrd1

**reallocate** 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

pSliceInLayer

Slc0

Slc1

Slc2

...

**Step 0/1**

Thrd1

Thrd0

Thrd0

pSliceInLayer

Slc0

Slc1

Slc2

…

**Step 2**

Slc0

Slc1

Slc2

...

…

…

…

**Step 3**

Thrd2

Thrd0

## 

## 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

Slc0

Slc1

Slc2

…

ppSliceInLayer

pSliceInThread[0]

Slc0

Slc1

Slc2

…

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

Slc0

Slc1

Slc2

…

ppSliceInLayer

pSliceInThread[0]

Slc0

Slc1

Slc2

…

Slc0

Slc1

Slc2

Slc3

…

…

**reallocate**

…

Slc0

Slc1

Slc2

Slc3

…

### 1.2.2. Multi-thread

Slc0

Slc1

Slc2

Slc3

Slc4

…

ppSliceInLayer

pSliceInThread[0]

Slc4

Slc2

Slc0

…

Slc1

Slc3

Slc5

…

pSliceInThread[1]

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

ppSliceInLayer

…

Slc4

Slc3

Slc2

Slc1

Slc0

**Main thread**

Slc0

Slc1

Slc2

Slc3

Slc4

…

…

Slc4

Slc2

Slc0

…

…

Slc0

Slc2

Slc4

…

…

Slc1

Slc3

Slc5

…

…

**Thread 1**

pSliceInThread[0]

**Reallocate**

**/update**

**Reallocate**

**Thread 0**

pSliceInThread[1]

# 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

bThreadBufferUsage[iThrdIdx] = false

**Slice buffer**

pSliceInThread[1]

pSliceInThread[2]

pSliceInThread[0]

Thrd\_0

**Idle**

**thread**

Thrd\_2

Thrd\_1

**Get\_UnUsed\_Buffer()**

slice in layer

Get\_Idle\_Thrd()

Slc8

Slc1

Slc2

Slc0

….

**Slice tasks**

Get\_waiting\_task()

**Task manager**

Encode\_One\_Slice()

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

**Task manager**

Thrd\_2

Slice\_2

pSliceBuffer[2]

Thrd\_1

Slice\_1

pSliceBuffer[1]

Thrd\_0

Slice\_0

pSliceBuffer[0]

Thrd\_2

Slice\_8

pSliceBuffer[2]

…

…

…

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

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

pSliceBuffer[0]

Slc6

Slc3

Slc0

Slc1

Slc4

Slc7

pSliceBuffer[1]

Thrd\_0

Thrd\_1

Thrd\_2

pSliceBuffer[2]

Slc8

Slc5

Slc2

## 2.3. Slice num not the same among threads

case 1:

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

Slc0

Slc3

pSliceBuffer[0]

Thrd\_0

Thrd\_1

Slc7

Slc1

Slc4

pSliceBuffer[1]

Slc6

Thrd\_2

pSliceBuffer[2]

Slc8

Slc5

Slc2

case 2:

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

Slc0

pSliceBuffer[0]

Thrd\_0

Thrd\_1

Slc8

Slc1

Slc3

pSliceBuffer[1]

…

Thrd\_2

pSliceBuffer[2]

Slc2

Corner case 1, no encoded slice for one thread:

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

Slc0

Slc2

pSliceBuffer[0]

Thrd\_0

Thrd\_1

Slc7

Slc1

Slc3

pSliceBuffer[1]

…

Thrd\_2

pSliceBuffer[2]

Corner case 2, all slices encoded by one thread :

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

Thrd\_0

…

Slc1

Slc0

Slc8

pSliceBuffer[0]

Thrd\_1

pSliceBuffer[1]

Thrd\_2

pSliceBuffer[2]

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

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

pSliceBuffer[0]

Slc6

Slc3

Slc0

Thrd\_1

Thrd\_0

pSliceBuffer[1]

Slc7

Slc4

Slc1

Thrd\_2

pSliceBuffer[2]

Slc8

Slc5

Slc2

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

ppSliceInLayer

Slc0

Slc1

Slc2

Slc3

…

Slc8

pSliceBuffer[0]

Slc2

Slc3

Slc0

Out-of-order

Thrd\_1

Thrd\_0

pSliceBuffer[1]

Slc7

Slc4

Slc1

Thrd\_2

pSliceBuffer[2]

Slc8

Slc6

Slc5

## 2.6. Dynamic slice mode case1

**Partition\_0**: **Slc0, Slc3**

Slc0

Slc3

**Partition\_1:** **Slc1, Slc4,Slc7,Slc10**

Slc1

Slc4

Slc7

Slc10

Slc5

Slc8

Slc5

Slc2

**Partition\_2:** **Slc2, Slc5, Slc8**

Slice\_Index = PartitonID + Partition\_Num \* Slice\_Index\_InPartition

Example: Slc7 = 1 + 3 \* 2

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;

Slc7 = PartitonOffset[1] + 7 / 3 = 2 + 2 = 4;

Which ppSliceInLayer[4] = Slc7

Slc8

…

Slc3

Slc2

Slc1

Slc0

ppSliceInLayer

Slc2

Slc5

Slc8

pSliceBuffer[2]

Thrd\_2

Slc7

pSliceBuffer[1]

Slc4

Slc1

Thrd\_1

Slc10

Slc0

Slc3

pSliceBuffer[0]

Thrd\_0

## 2.7. Dynamic slice mode case2

**Thread\_0 encoded two partitions**

**while no partition for Thread\_2**

**Partition\_0**: **Slc0, Slc3**

Slc0

Slc3

**Partition\_1:** **Slc1, Slc4,Slc7,Slc10**

Slc1

Slc4

Slc7

Slc10

Slc5

Slc8

Slc5

Slc2

**Partition\_2:** **Slc2, Slc5, Slc8**

Slice\_Index = PartitonID + Partition\_Num \* Slice\_Index\_InPartition

Example: Slc7 = 1 + 3 \* 2

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;

Slc7 = PartitonOffset[1] + 7 / 3 = 2 + 2 = 4;

Which ppSliceInLayer[4] = Slc7

Slc8

…

Slc3

Slc2

Slc1

Slc0

ppSliceInLayer

pSliceBuffer[2]

Slc7

pSliceBuffer[1]

Slc4

Slc1

Slc10

Thrd\_0

Slc8

Slc5

Slc2

Slc0

Slc3

pSliceBuffer[0]

Thrd\_2

Thrd\_1

# 3. Slice Buffer Update/Reorder

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

Dynamic slice mode

**Partition\_0**: **Slc0, Slc3**

Slc0

Slc3

**Partition\_1:** **Slc1, Slc4,Slc7,Slc10**

Slc1

Slc4

Slc7

Slc10

Slc5

Slc8

Slc5

Slc2

**Partition\_2:** **Slc2, Slc5, Slc8**

Slc0

Slc1

Slc2

Slc3

…

Slc8

ppSliceInLayer

Non-dynamic slice mode

Slc0

Slc1

…

Slc8

Slc2

Slc3

# 4. Bs buffer design

## 4.1. Slice Bs buffer

**Slice Buffer**

**Recycle used**

1. For thread slices, refer to InitBits()

pCtx->pSliceThreading->pThreadBsBuffer

2. Single thread,

replace by pOut->pBsBuffer

SWelsSliceBs sSliceBs

SBitStringAux\* pSliceBsa

……

……

iStartPos

iPayloadSize

pRawData

pEndBuf

pCurBuf

pStartBuf

iIndex

iLeftBits

uiCurBits

iBits

sNalExt

iNalIndex

sNalList[2]

pBs

uiSize

sBsWrite

iNalLen[2]

pBsBuffer

uiBsPos

pRawData

iPayloadSize

iStartPos

sNalExt

Slice Bs which will be write to layer bs buffer.

Multithread set to null

sNalUnitHeader

bIdrFlag

uiPriorityId

iNoInterLayerPredFlag

uiDependencyId

uiQualityId

uiTemporalId

bUseRefBasePicFlag

bDiscardableFlag

bOutputFlag

uiReservedThree2Bits

uiLayerDqId

bNalExtFlag

uiForbiddenZeroBit

uiNalRefIdc

eNalUnitType

uiReservedOneByte

### 4.1.1. init

InitSliceBsBuffer()

pSlice->sSliceBs.uiSize = iMaxSliceBufferSize;

(pCtx)->iSliceBufferSize[kiDlayerIndex];

(\*ppCtx)->pOut->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->pOut->sBsWrite;

pSlice->sSliceBs.pBs = NULL;

}

### 4.1.2 encode one slice

//set recycle bs buffer to m\_pSliceBs->pBsBuffer

SetOneSliceBsBufferUnderMultithread (m\_pCtx,

m\_iThreadIdx, 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)->pOut->uiSize = iCountBsLen;

(\*ppCtx)->pOut->pBsBuffer =

(uint8\_t\*)pMa->WelsMallocz (iCountBsLen, "pOut->pBsBuffer");

(\*ppCtx)->pFrameBs = (uint8\_t\*)pMa->WelsMalloc (iTotalLength, "pFrameBs");

(\*ppCtx)->iFrameBsSize = iTotalLength;

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

if (bIndependenceBsBuffer) {

pSlice->pSliceBsa = &pSlice->sSliceBs.sBsWrite;

pSlice->sSliceBs.pBs =

(uint8\_t\*)pMa->WelsMalloc (iMaxSliceBufferSize, "SliceBs");

} else {

pSlice->pSliceBsa = pCtx->pOut->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],

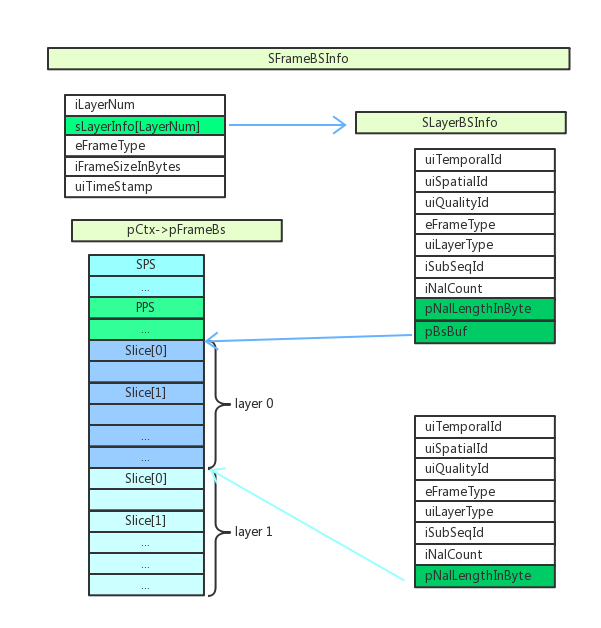
&pCtx->pCurDqLayer->sLayerInfo.sNalHeaderExt,

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 kiThreadIndex);

### 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 kiThreadIndex);

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[iThreadIdx]

***//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);

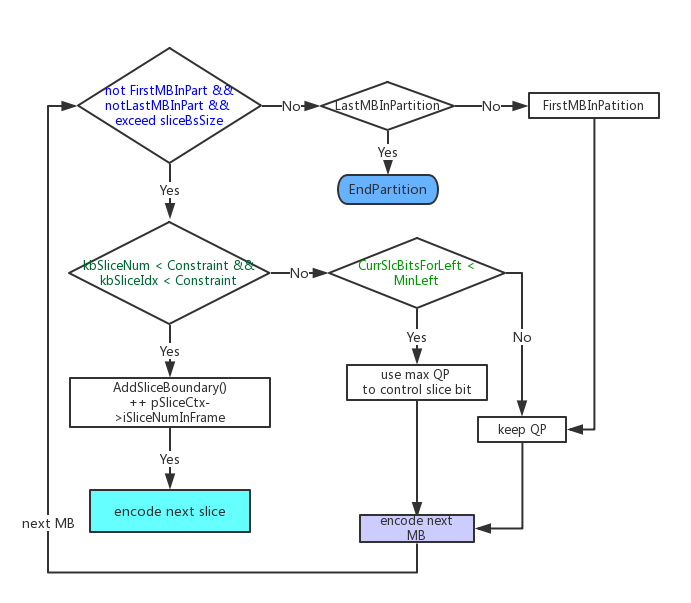
#

}

***//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 befor 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);

---->DeblockingFilterSliceAvcbase() // function pointer only, no processing

---->pFuncList->pfUpdateFMESwitch = UpdateFMESwitch; //function pointer

---->CountFMECostDown() //ppSlice

---->WelsInitCurrentDlayerMltslc (pCtx, iPicIPartitionNum) //Dynamic slice mode only

---->UpdateSlicepEncCtxWithPartition() // 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()**

**----> DeblockingFilterFrameAvcbase (pCurLayer, pEnc->pFuncList); //ppSlice**

**---->DeblockingFilterSliceAvcbase() //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**