

Slice buffer design

Table of Contents

1. Overview for new design	3
1.1 Origin design	3
1.1.1. Single thread	3
1.1.2. Multi thread.....	4
1.2 New design in review.....	5
1.2.1 Single thread.....	5
1.2.2. Multi-thread.....	6
2. SLICE BUFFER AND THREAD	7
2.1 Before encoding one layer	7
2.2. Normal case for thread index and slice buffer index.....	9
2.3. Encoded slice num is not the same among threads.....	10
2.4. Different thread index in the same slice buffer.....	12
2.5. Slice index out-of-order case	13
2.6. Dynamic slice mode case1	14
2.7. Dynamic slice mode case2	15
2.8 Slice index in different slice mode	16
3. Slice Buffer Init/Update/Reorder	17
3.1. Allocate and initial before encoding first IDR.....	17
3.2 Init slice buffer info before encoder one layer	18
3.3.1 basic flow chart for encoding one slice task.....	19
3.3.2 Task manager	20
3.3.3. InitTask before encode one slice.....	21
3.3.4 Slice buffer update/reallocate during execute task.....	22
3.4. Slice buffer update after encoder one layer	23
3.4.1. ppSliceInLayer update.....	23
4. Final design	25
4.1 overall flow chart.....	25
4.2. functions for new design	27
4.2.1 Reallocate module	27
4.2.2 reallocate step	27
4.2.3 Extend layer buffer module.....	28
4.2.4. Reorder slice buffer module.....	28
4.2.5 other functions for new design	29
5. TestData	30
5.1 Test Script	30
5.1.1 Test script brief introduction	30
5.1.2 Memory analyse for encode one case	32
5.1.3 memory analyse for one YUV	33
5.1.4 Memory analyse for all YUVs	34
5.2 Final Test Data	35

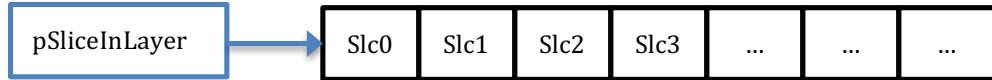
6. Code review and PRs.....	36
7. Buffer Structure.....	37
7.1. Slice Bs buffer.....	37
7.2. Frame bs buffer	38
7.3 pCtx-pOut.....	39
8. Other design.....	40
8.1. dynamic slice, slice boundary strategy	40

1. Overview for new design

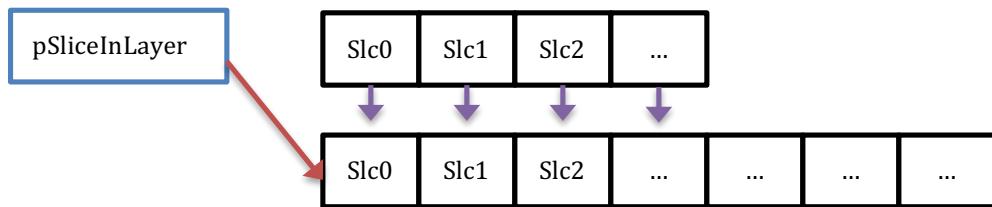
1.1 Origin design

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

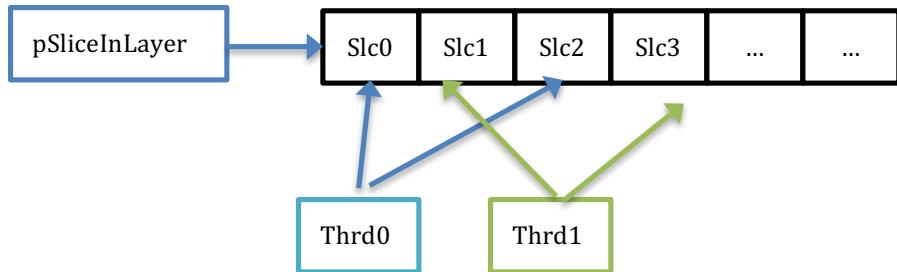
1.1.1. Single thread



realloc when current slice index larger than max slice num

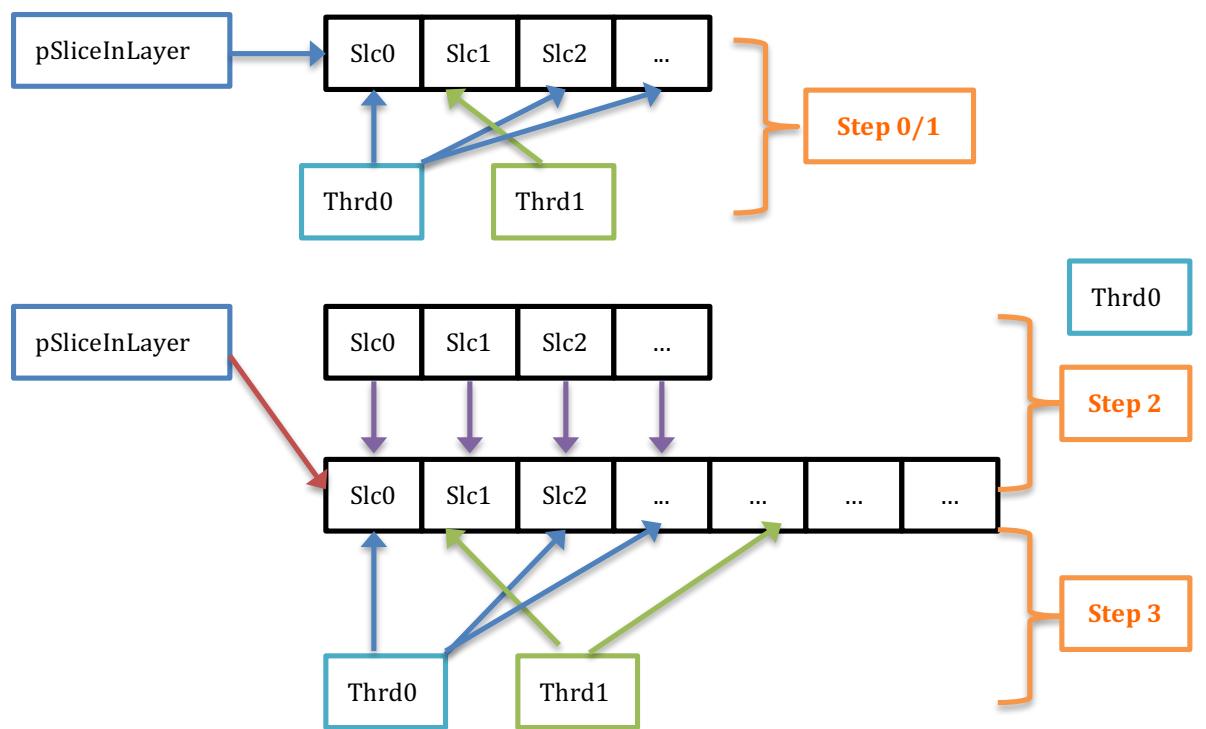


1.1.2. Multi thread



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

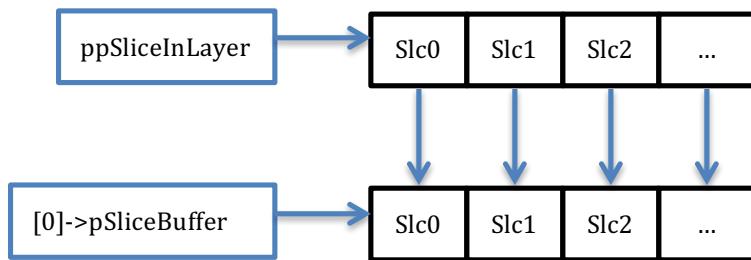


1.2 New design in review

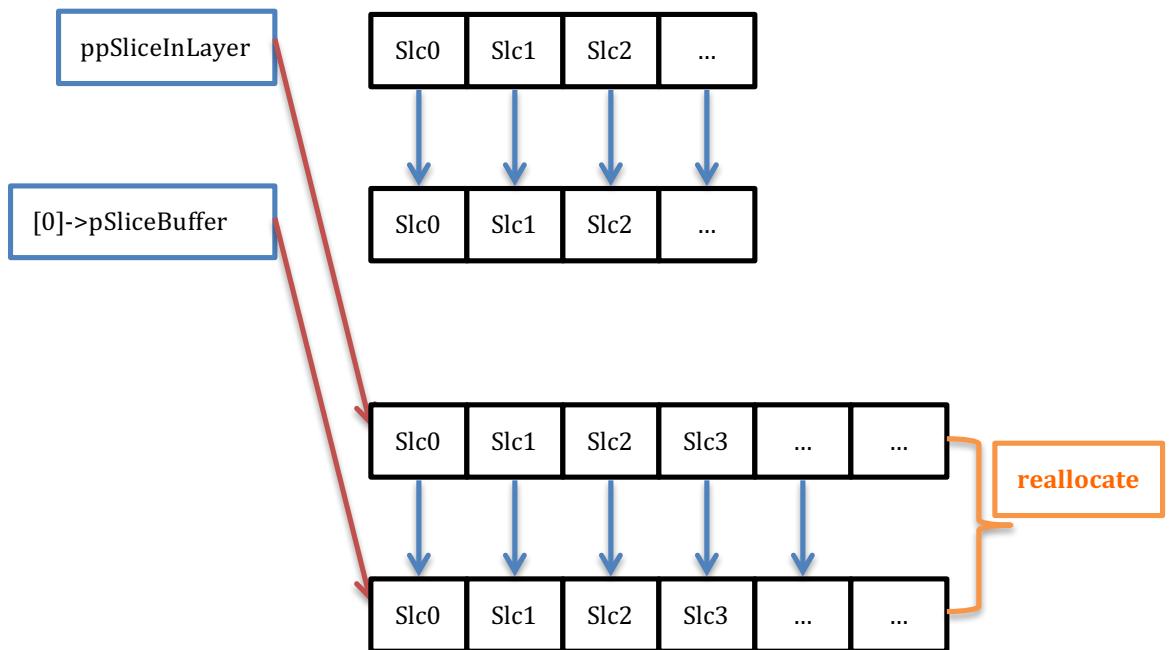
```
SSlice** ppSliceInLayer;           // point to actual slice buffer

typedef struct TagSliceBufferInfo {
    SSlice*             pSliceBuffer; // slice buffer for multi thread,
    int32_t              iMaxSliceNum;
    int32_t              iCodedSliceNum;
}SSliceBufferInfo;
```

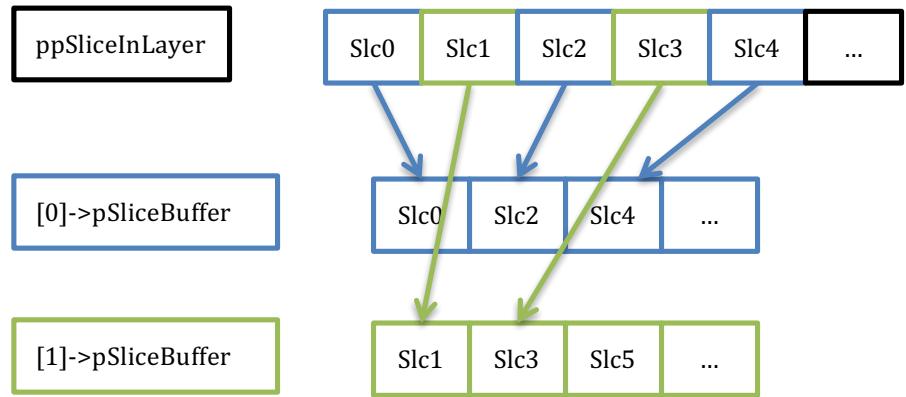
1.2.1 Single thread



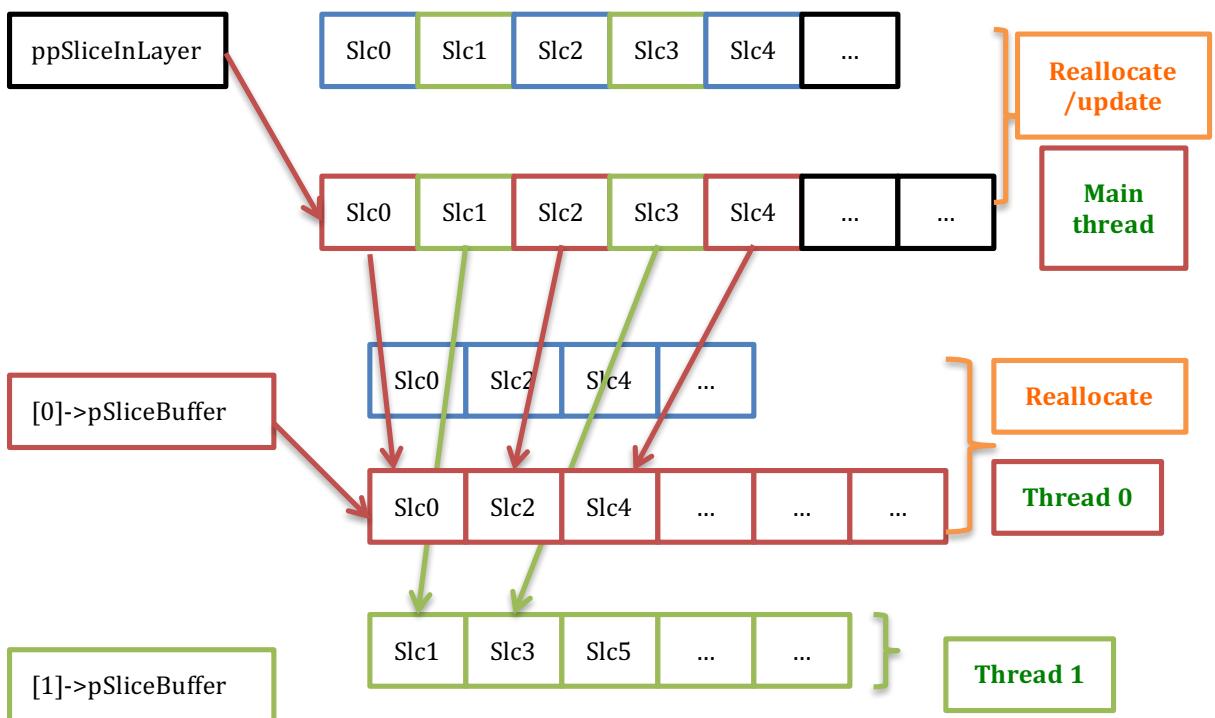
realloc when current slice index larger than max slice num



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

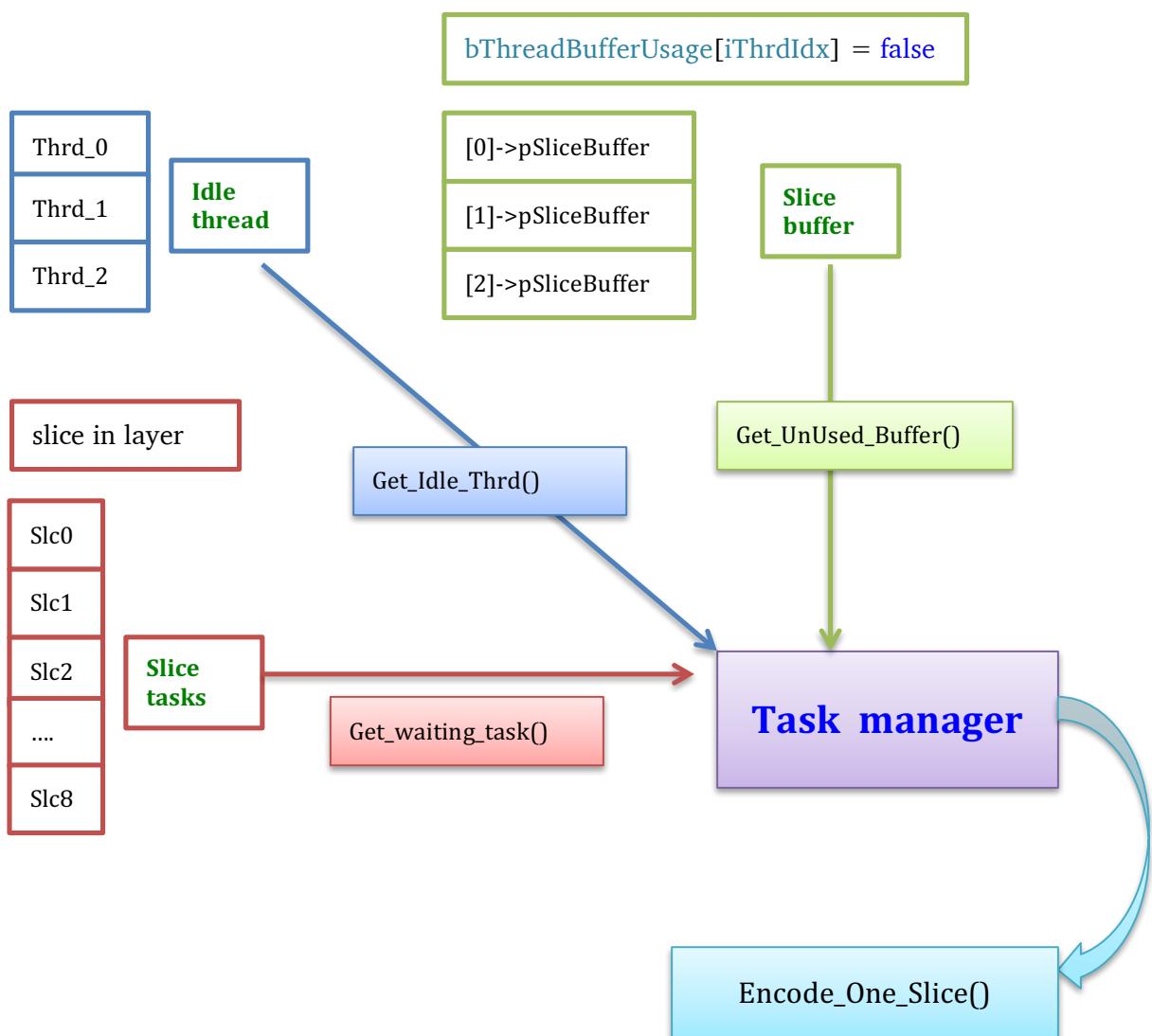
2.1 Before encoding one layer

the status of slice buffer and thread :

example:

thread: 3 threads

slices: 9 slices in layer



Query thread buffer function:

```
int32_t CWelsSliceEncodingTask::QueryEmptyThread (bool* pThreadBsBufferUsage) {  
    for (int32_t k = 0; k < MAX_THREADS_NUM; k++) {  
        if (pThreadBsBufferUsage[k] == false) {  
            pThreadBsBufferUsage[k] = true;  
            return k;  
        }  
    }  
    return -1;  
}
```

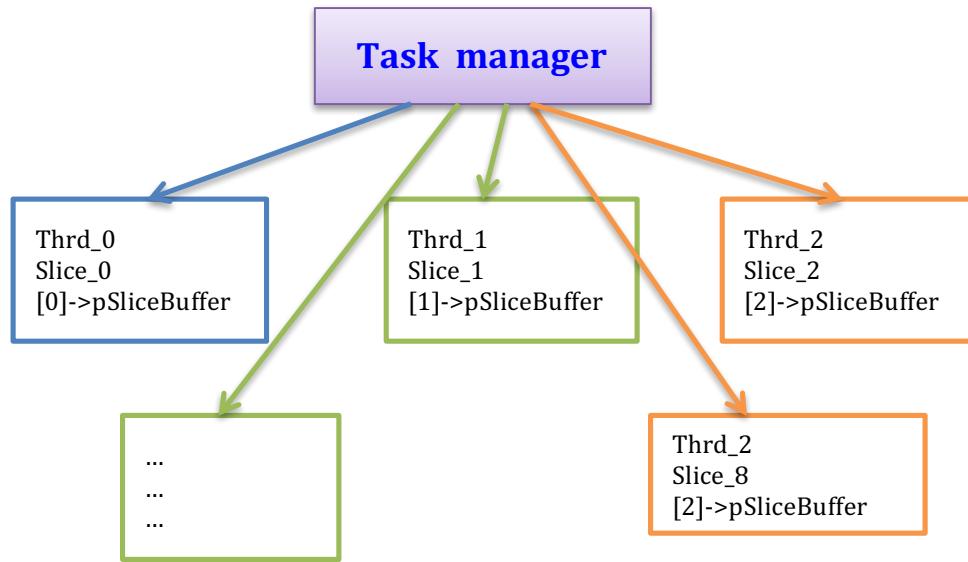
Example:

k=1, means that encoding slice task using [0]->pSliceBuffer as slice buffer

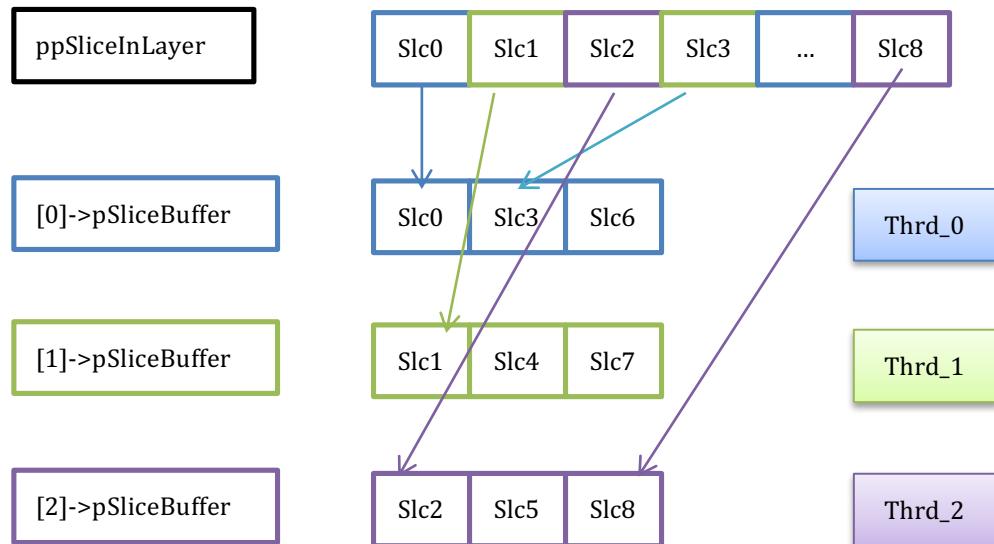
Query IDLE thread for slice task function:

```
CWelsTaskThread* CWelsThreadPool::GetIdleThread() {  
    CWelsAutoLock cLock (m_cLockIdleTasks);  
  
    if (m_cIdleThreads->size() == 0) {  
        return NULL;  
    }  
  
    CWelsTaskThread* pThread = m_cIdleThreads->begin();  
    m_cIdleThreads->pop_front();  
    return pThread;  
}
```

2.2. Normal case for thread index and slice buffer index

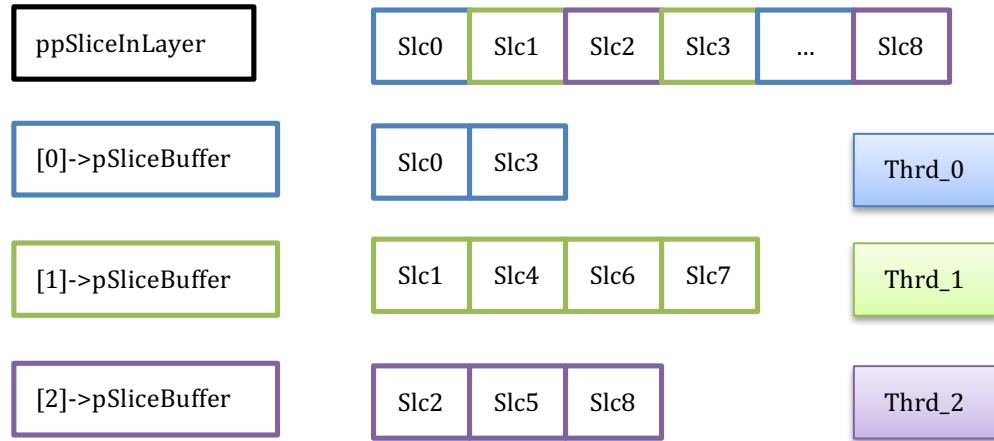


Final map for thread index and slice buffer index, slice index



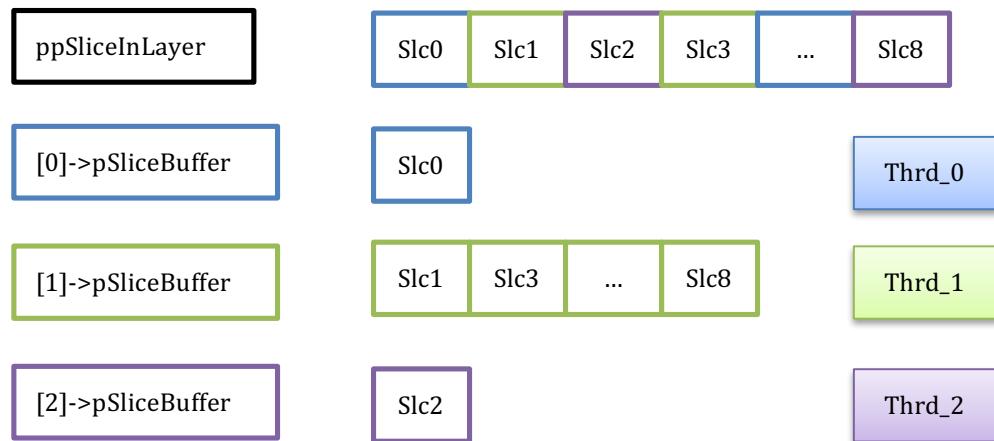
2.3. Encoded slice num is not the same among threads

case 1:



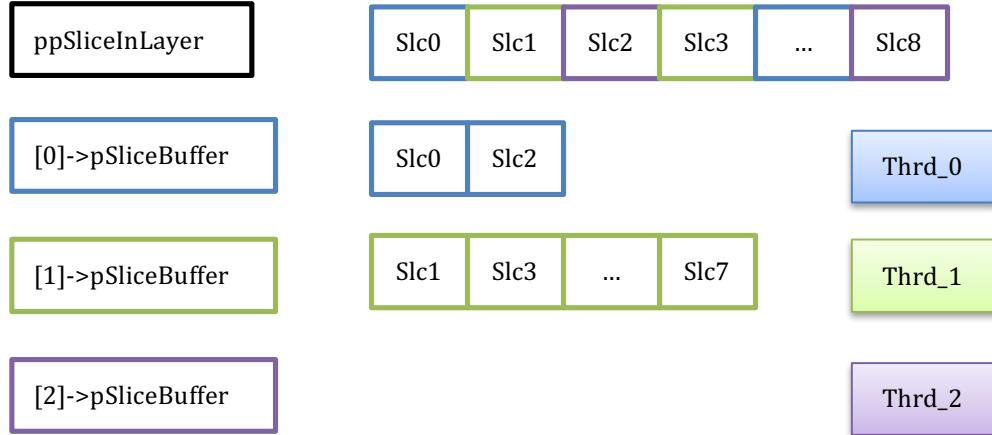
case 2:

encode time for Slc0 and Slc_2 are longer than Slc1, Slc3~Slc8



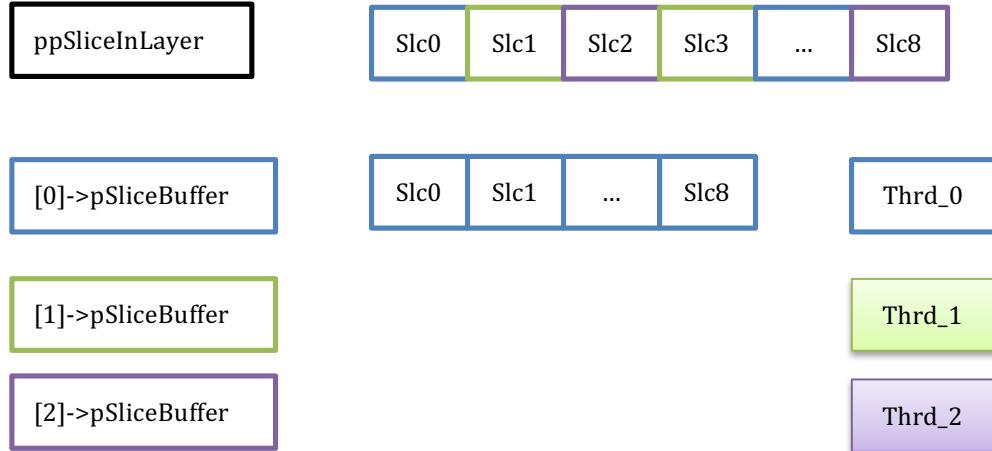
Corner case 1, no encoded slice for one thread:

Thread_0 and Thread_1 are fast enough and always get CPU resource,
Thread_2 is under waiting status/IDLE status.



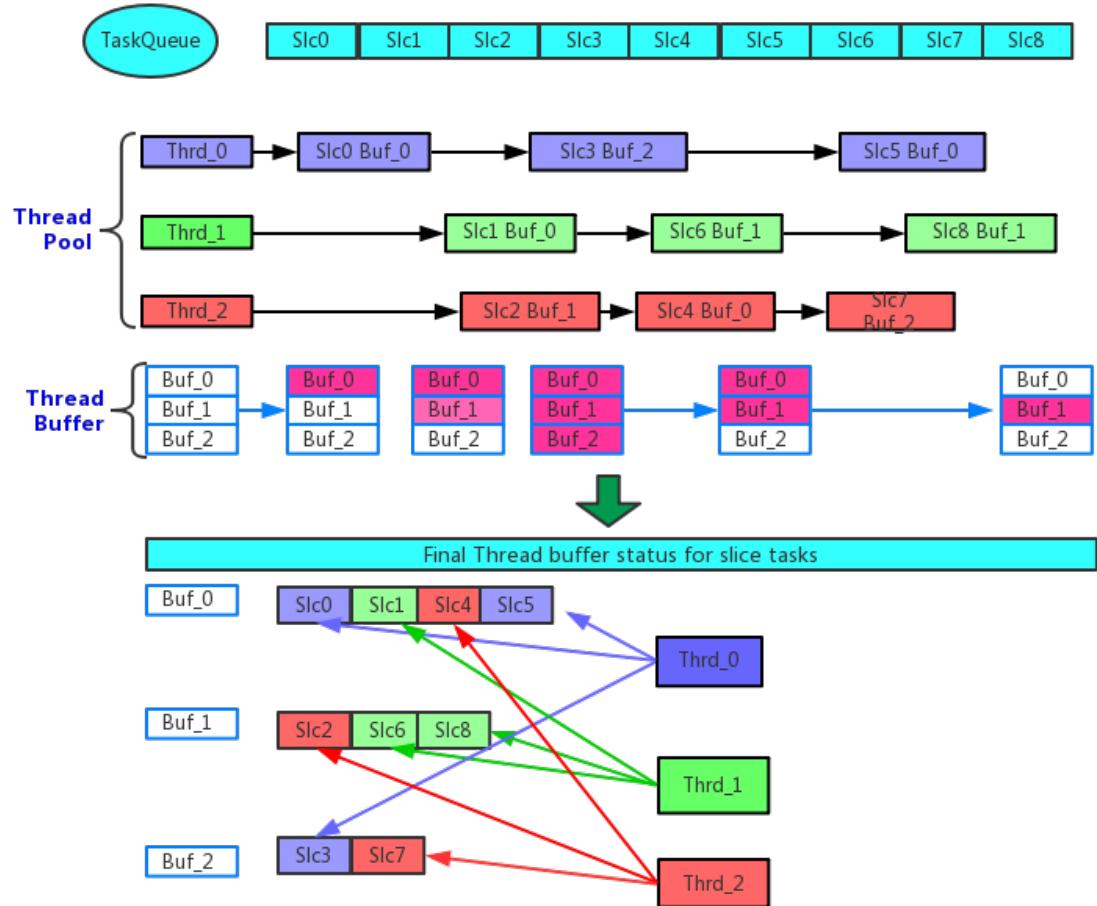
Corner case 2, all slices encoded by one thread :

Thread_0 is fast enough and always get CPU resource,
Thread_1 and Thread_2 are under waiting status/IDLE status.

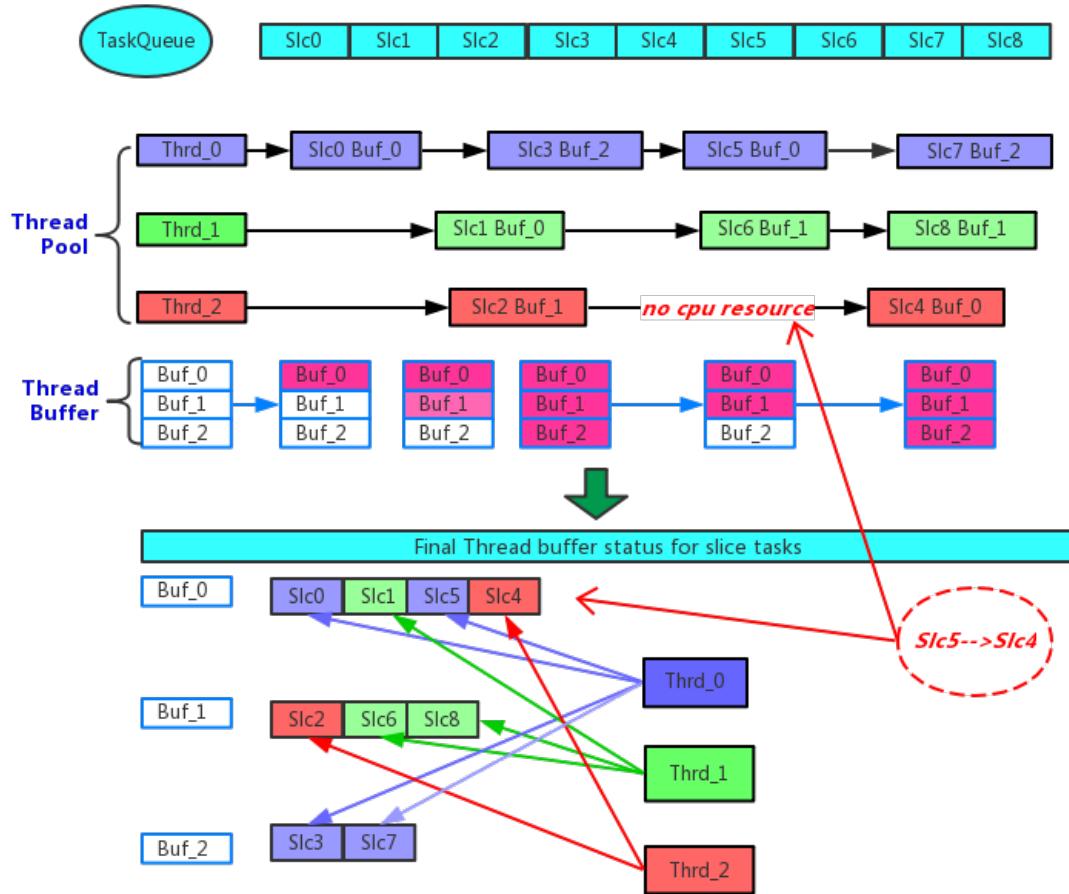


2.4. Different thread index in the same slice buffer

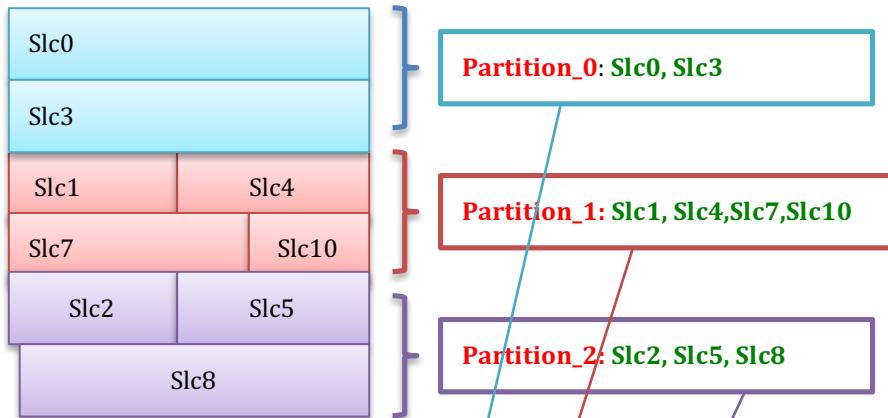
Timing diagram for Thread—SlcTask—ThreadBuffer



2.5. Slice index out-of-order case



2.6. Dynamic slice mode case1



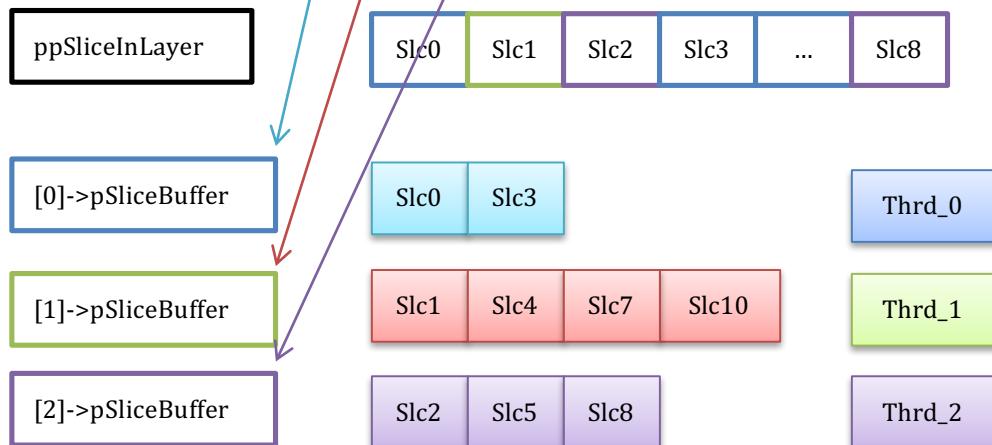
$$\text{Slice_Index} = \text{PartitonID} + \text{Partition_Num} * \text{Slice_Index_InPartition}$$

Example: the 2nd slice in partition 1, $\text{Slice_Index} = 1 + 3 * 2 = \text{Slc7}$
 the 3rd slice in partition 2, $\text{Slice_Index} = 2 + 3 * 1 = \text{Slc5}$

$$\text{Slice_Index_InLayer} = \text{PartitonOffset}[\text{Partiton_ID}] + \text{Slice_Index} / \text{Partition_Num}$$

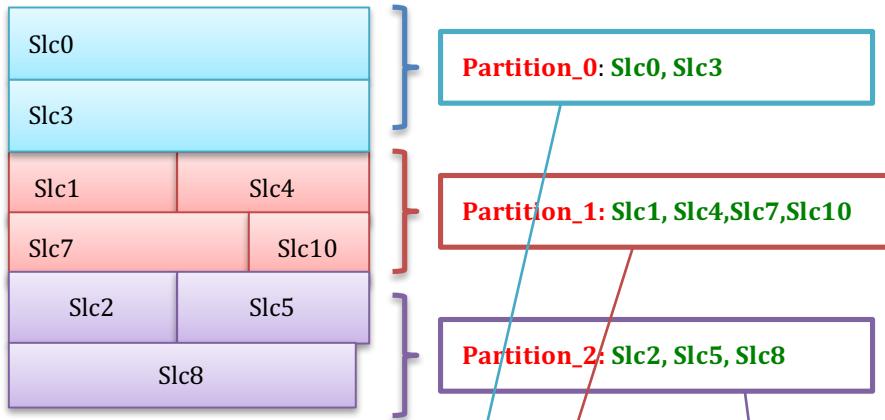
Here in example, PartitonOffset[0] = 0,
 PartitonOffset[1] = 0 + 2 = 2;
 PartitonOffset[2] = 0 + 2 + 4 = 6;

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



2.7. Dynamic slice mode case2

**Thread_0 encoded two partitions
while no partition for Thread_2**



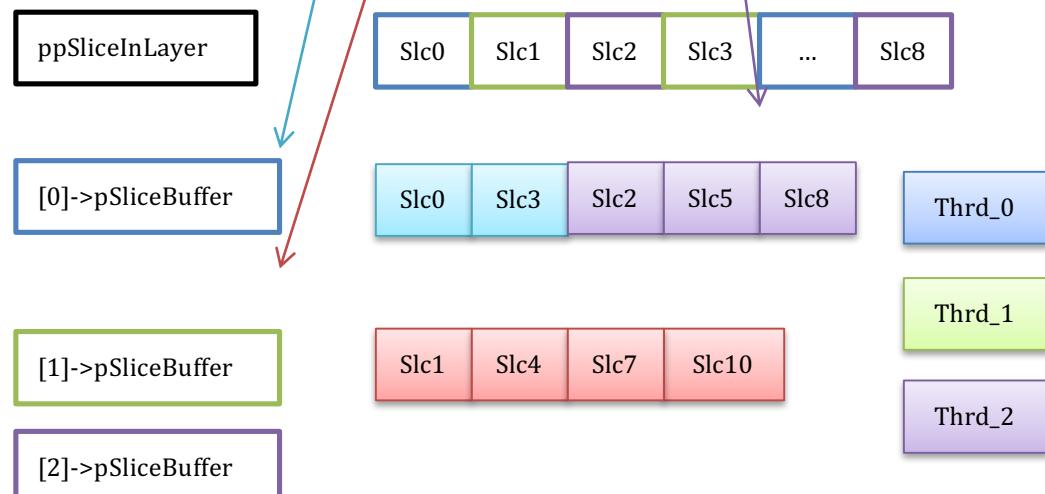
$$\text{Slice_Index} = \text{PartitonID} + \text{Partition_Num} * \text{Slice_Index_InPartition}$$

Example: the 2nd slice in partition 1, $\text{Slice_Index} = 1 + 3 * 2 = \text{Slc7}$
the 3rd slice in partition 2, $\text{Slice_Index} = 2 + 3 * 1 = \text{Slc5}$

$$\text{Slice_Index_InLayer} = \text{PartitonOffset}[\text{Partiton_ID}] + \text{Slice_Index} / \text{Partition_Num}$$

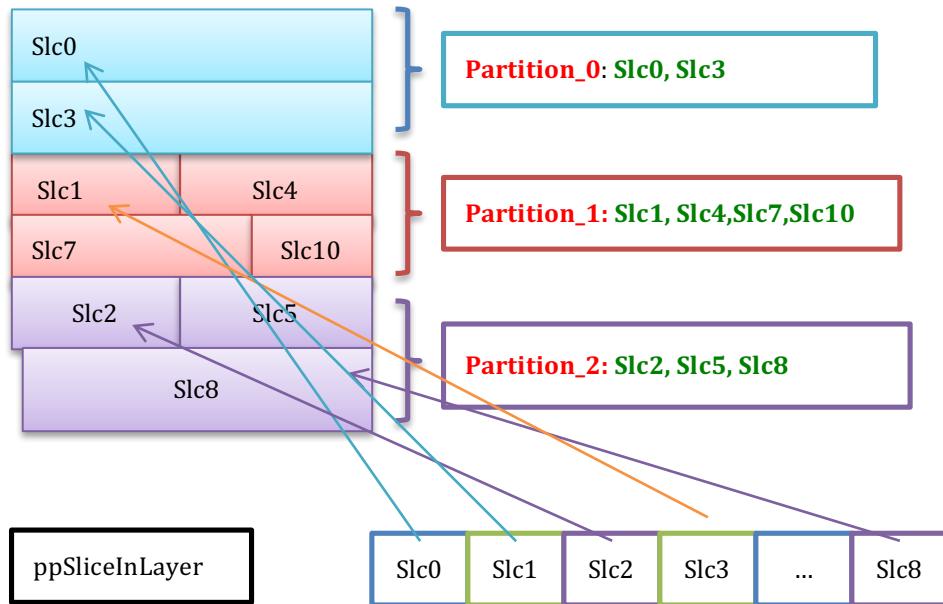
Here in example, PartitonOffset[0] = 0;
PartitonOffset[1] = 0 + 2 = 2;
PartitonOffset[2] = 0 + 2 + 4 = 6;

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

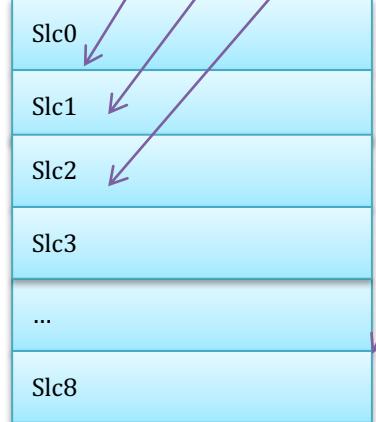


2.8 Slice index in different slice mode

Dynamic slice mode



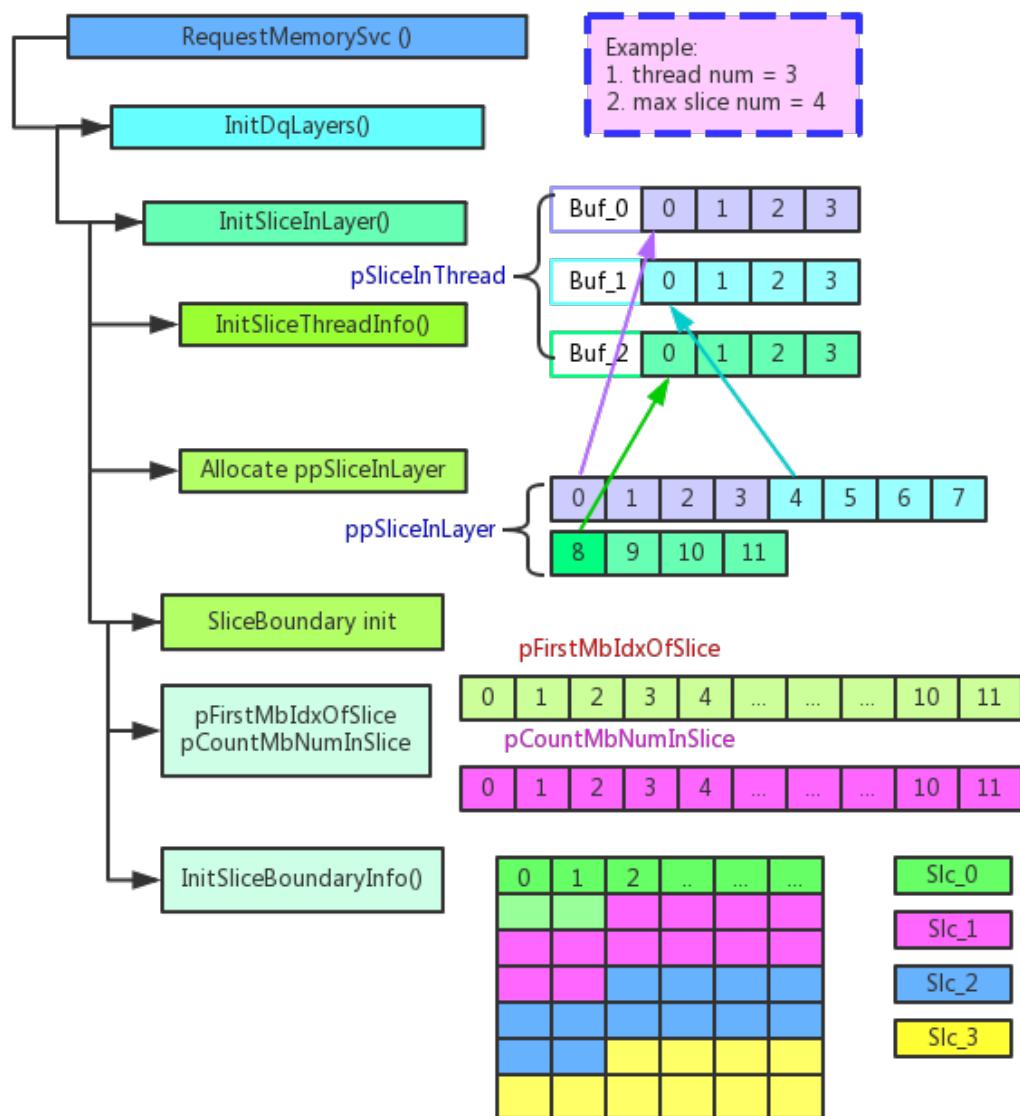
Non-dynamic slice mode



3. Slice Buffer Init/Update/Reorder

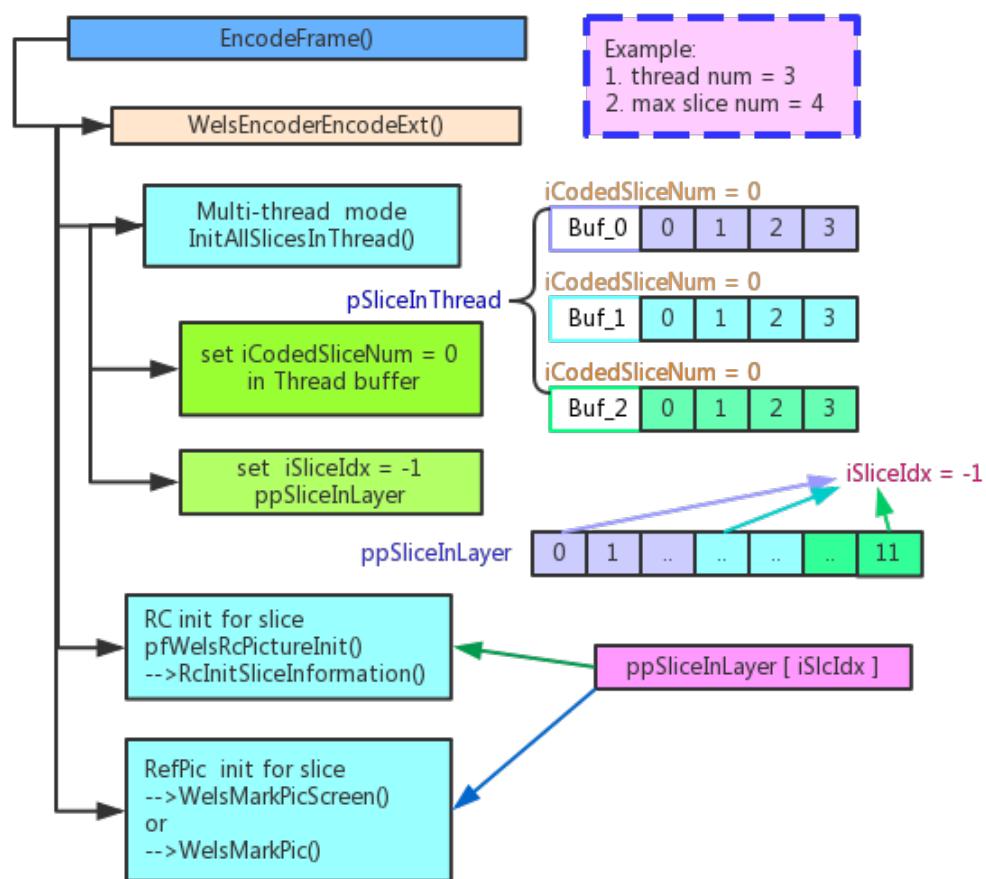
3.1. Allocate and initial before encoding first IDR

2. Allocate [0]->pSliceBuffer, [1] ->pSliceBuffer, etc.
here pSliceInThread has the same meaning with [0/1/2...]->pSliceBuffer
3. Allocate ppSliceInLayer buffer
4. Allocate pFirstMbIdxOfSlice and pCountMbNumInSlice buffer
Init slice boundary info



3.2 Init slice buffer info before encoder one layer

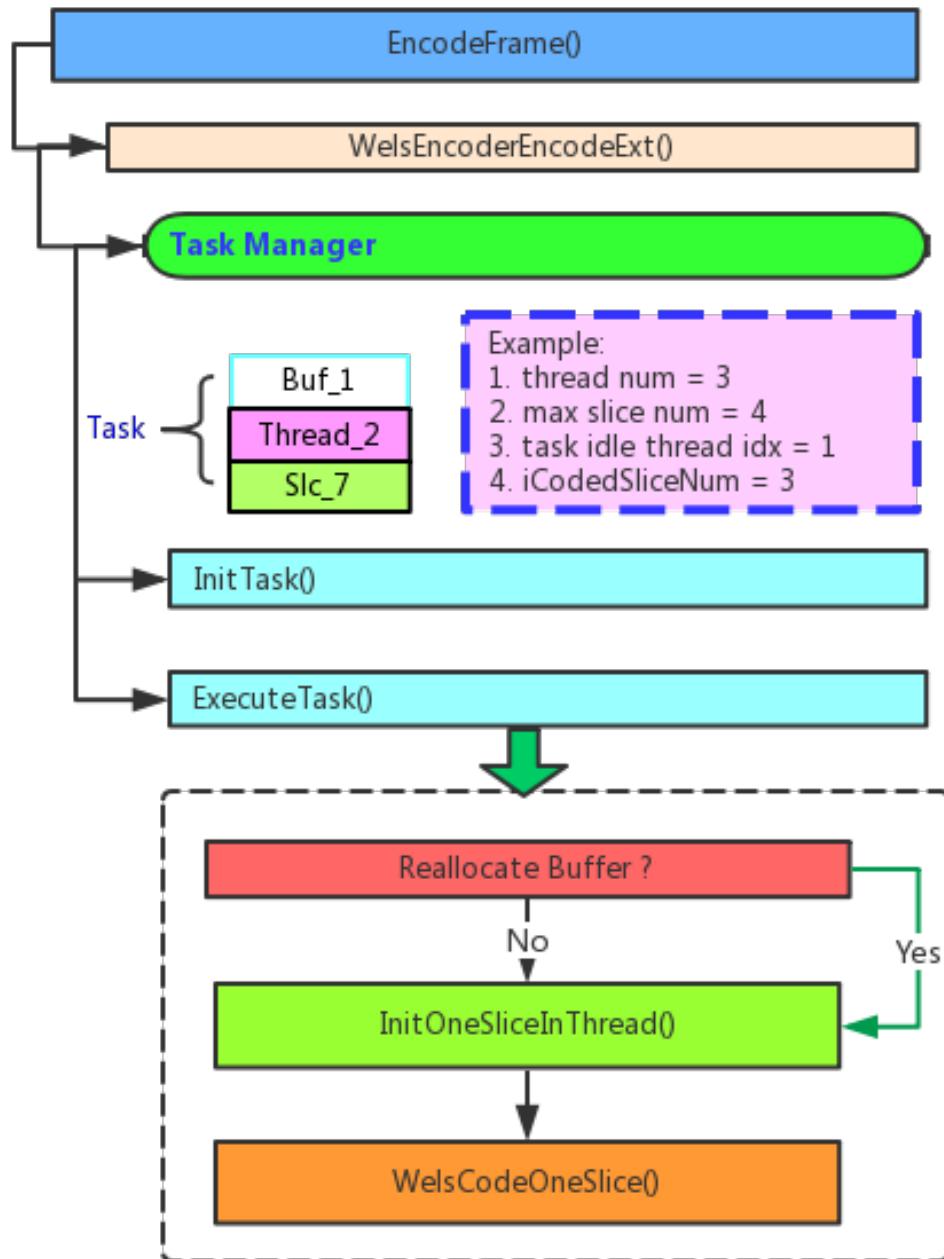
- Set iCodedSliceNum = 0 for all thread buffer
- Set iSliceIdx = -1 for all slice buffer (will set to actual slice idx during encoding one slice)
- Init RC info for all slice level parameters using ppSliceInLayer[iSlcIdx]
- Init reference pic info for all slices using ppSliceInLayer[iSlcIdx]



3.3 Slice buffer init/update/reallocate during encoding one slice

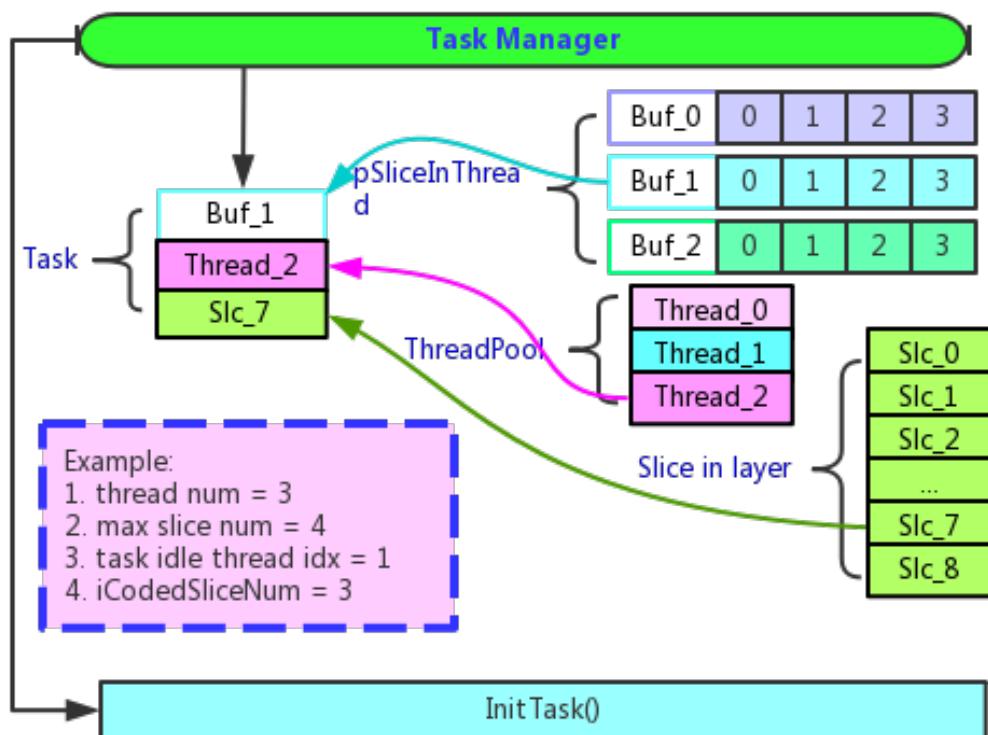
3.3.1 basic flow chart for encoding one slice task

- Task manager: Get idle thread, unused thread buffer, and encoding slice
- Init task: Init slice boundary info etc.
- Execute task: Encode one slice etc



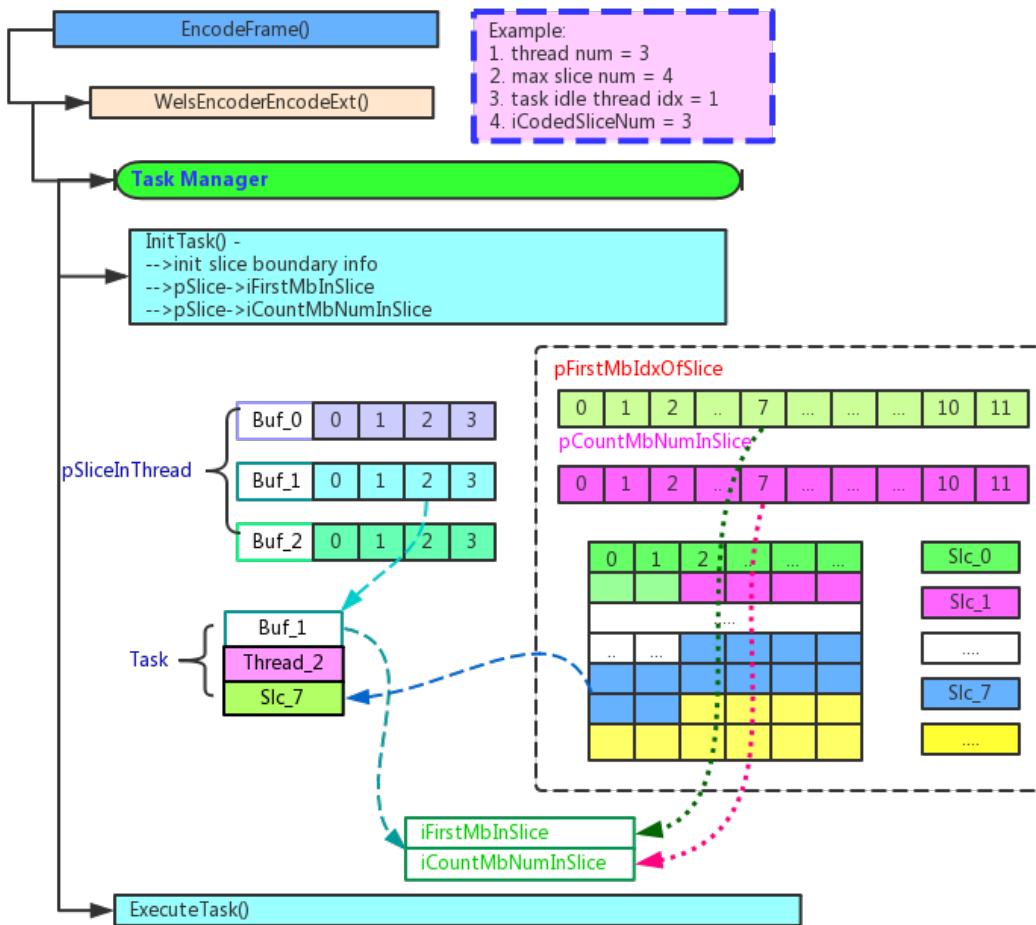
3.3.2 Task manager

- Get unused thread buffer, in this example, choose Buf_1 as slice buffer
- Get Idle thread, thread_2
- Get encode slice index in layer which will be encoded in this task, Slc_7



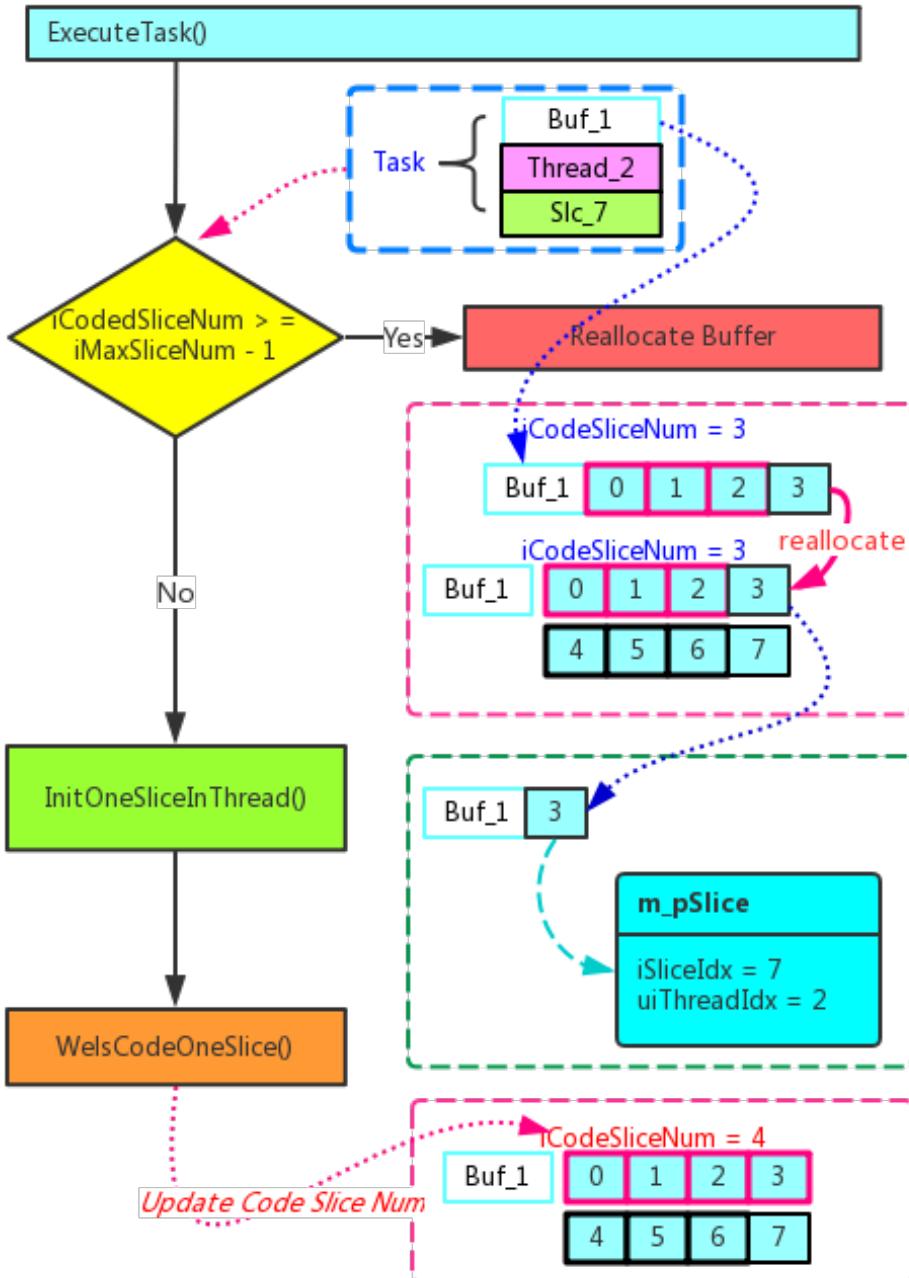
3.3.3. InitTask before encode one slice

- init rc with boundary info
GomRCInitForOneSlice()
- update next slice boundary(start MB index etc.)



3.3.4 Slice buffer update/reallocate during execute task

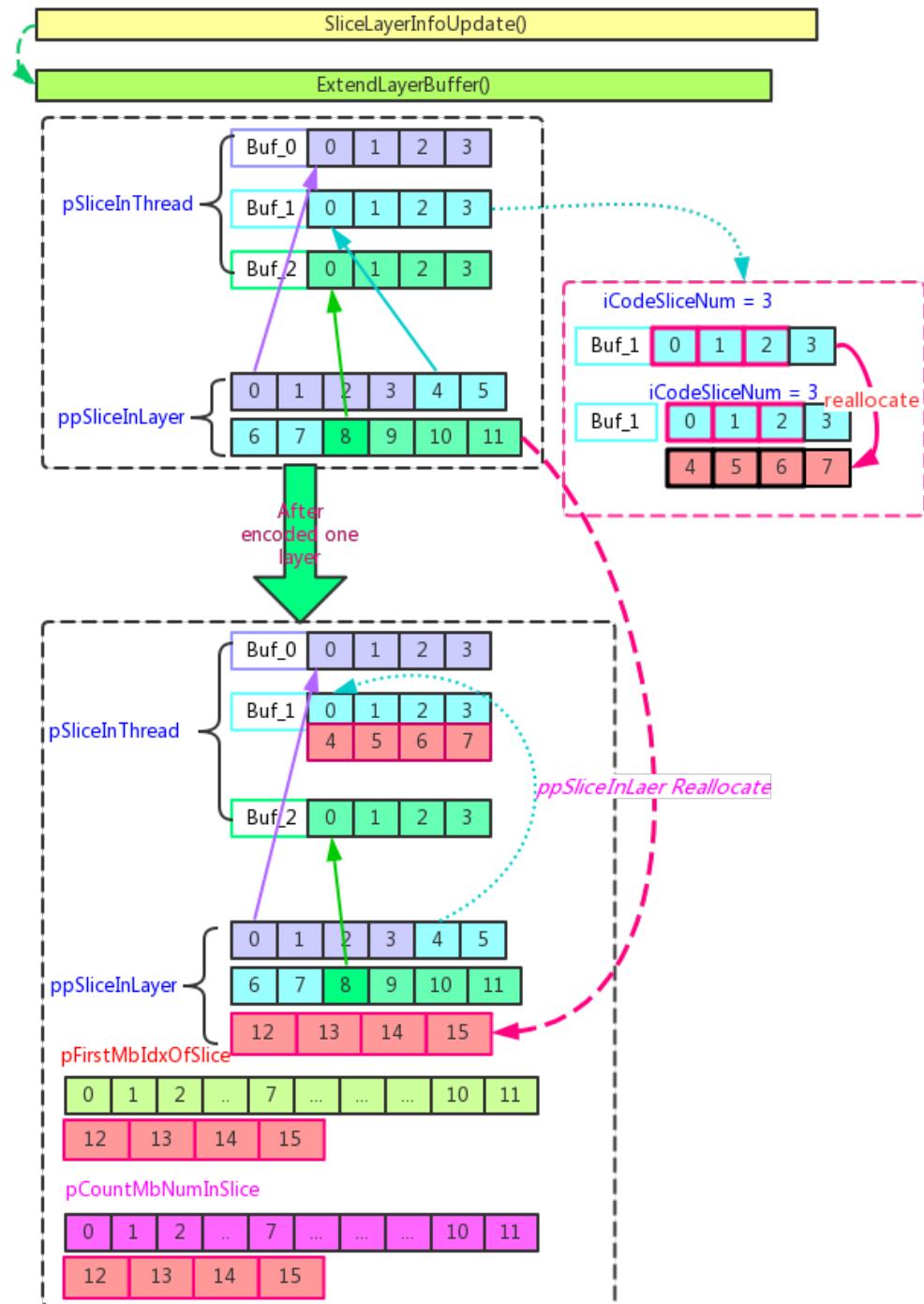
- ⊕ check whether need to reallocate
- ⊕ init m_pSlice buffer
- ⊕ update thread buffer info, iCodeSliceNum ++



3.4. Slice buffer update after encoder one layer

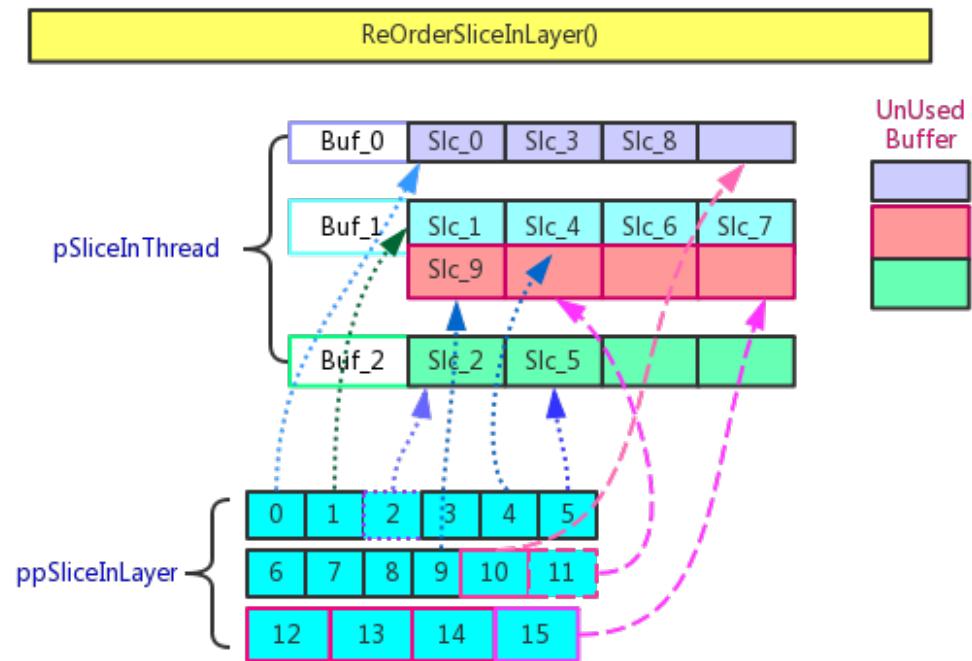
3.4.1. ppSliceInLayer update

extend and reallocate ppSliceInLayer buffer,
`ppSliceInLayer` buffer num == All thread slice buffer num



3.4.2. Slice buffer reorder

example 1

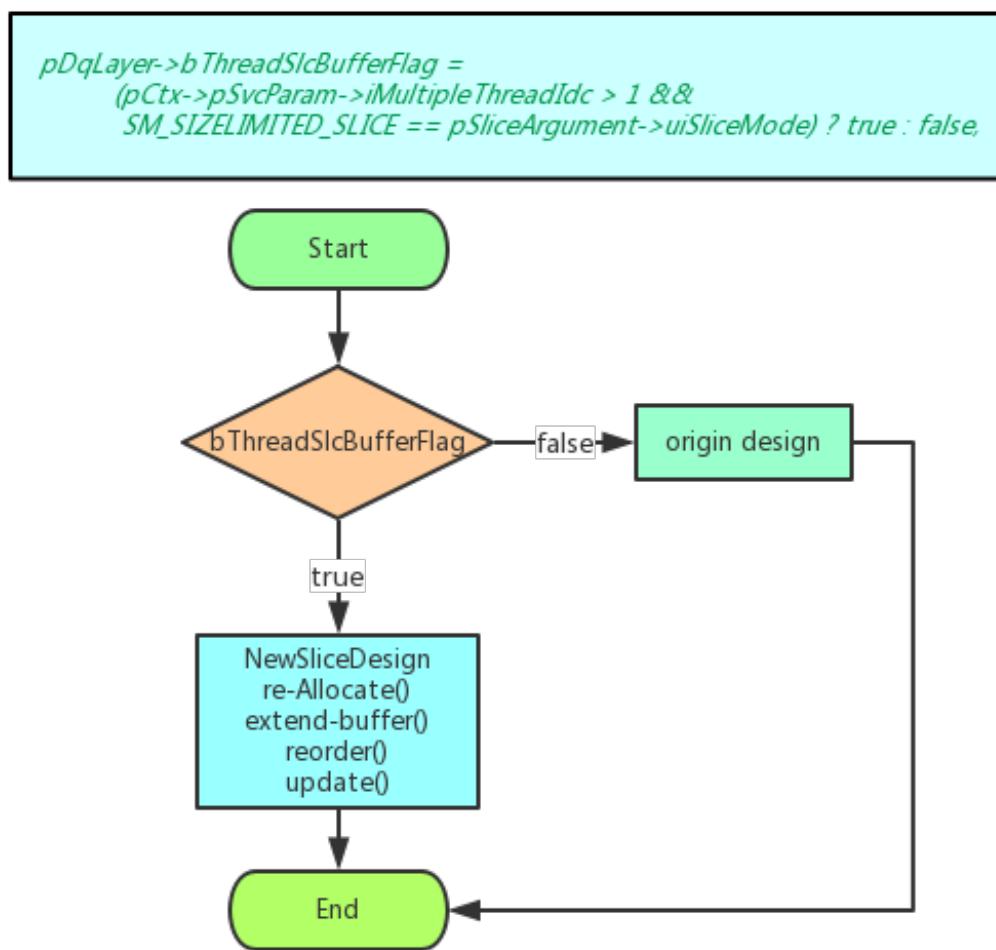


4. Final design

4.1 overall flow chart

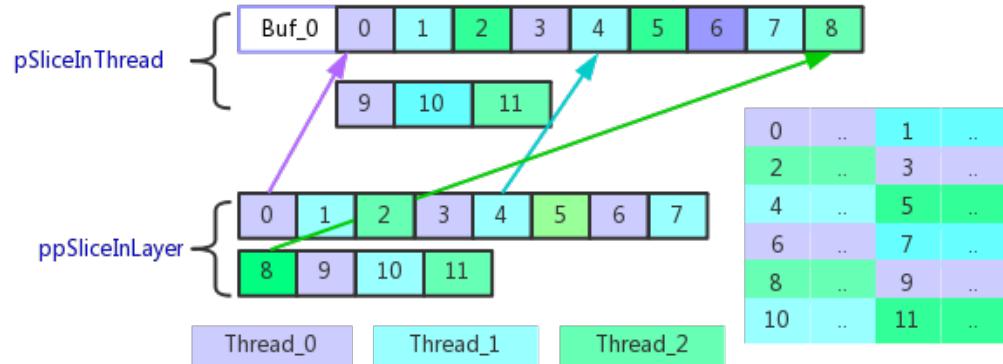
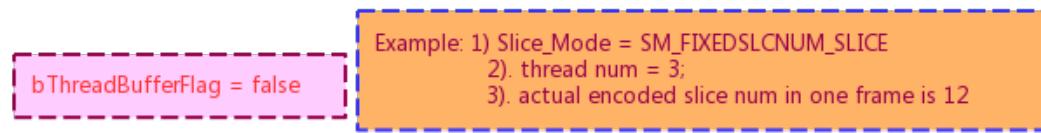
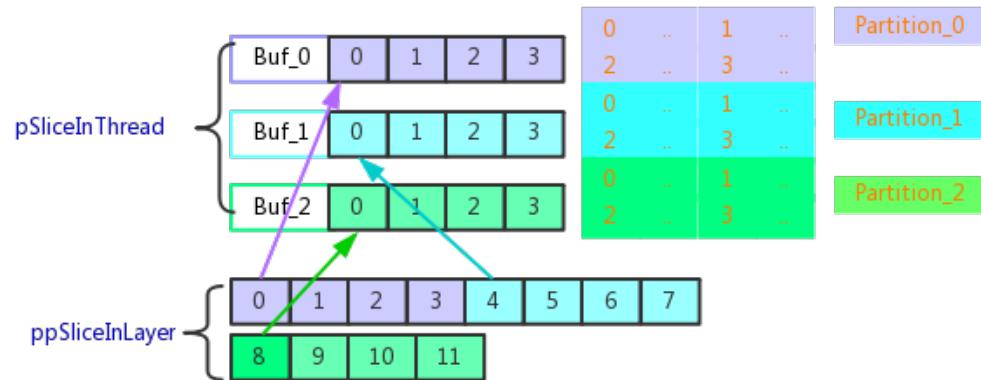
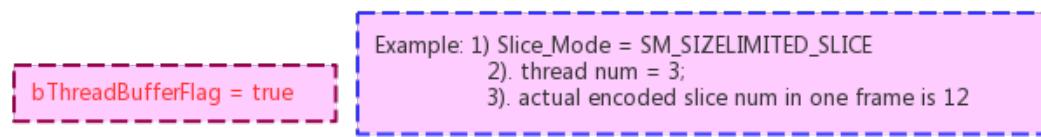
```
pDqLayer->bThreadSlcBufferFlag =
    (pCtx->pSvcParam->iMultipleThreadIdc > 1 &&
     SM_SIZELIMITED_SLICE == pSliceArgument->uiSliceMode) ? true : false;
```

- bThreadSlcBufferFlag = false, keep origin design
- bThreadSlcBufferFlag = true, using new buffer design



Example

```
pDqLayer->bThreadSlcBufferFlag =
    (pCtx->pSvcParam->iMultipleThreadIdc > 1 &&
     SM_SIZELIMITED_SLICE == pSliceArgument->uiSliceMode) ? true : false;
```



4.2. functions for new design

4.2.1 Reallocate module

```
bNeedReallocate = (pCurDq->sSliceBufferInfo[m_iThreadId].iCodedSliceNum >=
                    pCurDq->sSliceBufferInfo[m_iThreadId].iMaxSliceNum -1)
                    ? true : false;
if (bNeedReallocate) {
    WelsMutexLock (&m_pCtx->pSliceThreading->mutexThreadSlcBuffReallocate);
    //for memory statistic variable
    iReturn = ReallocateSliceInThread(m_pCtx, pCurDq, m_pCtx->uiDependencyId,
                                       m_iThreadId);
    WelsMutexUnlock (&m_pCtx->pSliceThreading->mutexThreadSlcBuffReallocate);
    if (ENC_RETURN_SUCCESS != iReturn) {
        return iReturn;
    }
}
```

4.2.2 reallocate step

at least $\frac{1}{2} * \text{MaxSliceNumOld}$ when reallocate new slice buffer

```
int32_t iIncreaseSlicNum = (iLeftMBNum * INT_MULTIPLY / iMBNumInPartition) *
                            iMaxSliceNumOld;

iIncreaseSlicNum = ( 0 == (iIncreaseSlicNum / INT_MULTIPLY) ) ? 1 :
                    (iIncreaseSlicNum / INT_MULTIPLY);
iIncreaseSlicNum = (iIncreaseSlicNum < iMaxSliceNumOld / 2) ?
                    (iMaxSliceNumOld / 2) : iIncreaseSlicNum;
iMaxSliceNumNew = iMaxSliceNumOld + iIncreaseSlicNum;
```

4.2.3 Extend layer buffer module

```
//reallocating ppSliceInLayer if total encoded slice num exceed max slice num
if (iMaxSliceNum > pCtx->pCurDqLayer->iMaxSliceNum) {
    iRet = ExtendLayerBuffer(pCtx, pCtx->pCurDqLayer->iMaxSliceNum, iMaxSliceNum);
    if (ENC_RETURN_SUCCESS != iRet) {
        return iRet;
    }
    pCtx->pCurDqLayer->iMaxSliceNum = iMaxSliceNum;
}
```

4.2.4. Reorder slice buffer module

```
//update ppSliceInLayer based on pSliceBuffer, reordering based on slice index
iRet = ReOrderSliceInLayer (pCtx, kuiSliceMode, pCtx->iActiveThreadsNum);
if (ENC_RETURN_SUCCESS != iRet) {
    WelsLog (&(pCtx->sLogCtx), WELS_LOG_ERROR,
        "CWelsH264SVCEncoder::SliceLayerInfoUpdate: ReOrderSliceInLayer
failed");
    return iRet;
}
```

4.2.5 other functions for new design

```
int32_t InitSliceList (SSlice*& pSliceList,
                      SBitStringAux* pBsWrite,
                      const int32_t kiMaxSliceNum,
                      const int32_t kiMaxSliceBufferSize,
                      const bool bIndependenceBsBuffer,
                      CMemoryAlign* pMa);

int32_t InitAllSlicesInThread (sWelsEncCtx* pCtx);

int32_t InitOneSliceInThread (sWelsEncCtx* pCtx,
                             SSlice*& pSlice,
                             const int32_t kiSlcBuffIdx,
                             const int32_t kiDlayerIdx,
                             const int32_t kiSliceIdx);

int32_t InitSliceInLayer (sWelsEncCtx* pCtx,
                         SDqLayer* pDqLayer,
                         const int32_t kiDlayerIndex,
                         CMemoryAlign* pMa);

int32_t ReallocateSliceList (sWelsEncCtx* pCtx,
                            SSliceArgument* pSliceArgument,
                            SSlice*& pSliceList,
                            const int32_t kiMaxSliceNumOld,
                            const int32_t kiMaxSliceNumNew);

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

int32_t ReallocSliceBuffer (sWelsEncCtx* pCtx);

int32_t GetCurLayerNalCount(const SDqLayer* pCurDq, const int32_t kiCodedSliceNum);
int32_t GetTotalCodedNalCount(SFrameBSInfo* pFbi);

int32_t FrameBsRealloc (sWelsEncCtx* pCtx,
                       SFrameBSInfo* pFrameBsInfo,
                       SLayerBSInfo* pLayerBsInfo,
                       const int32_t kiMaxSliceNumOld);

int32_t ReOrderSliceInLayer(sWelsEncCtx* pCtx,
                           const SliceModeEnum kuiSliceMode,
                           const int32_t kiThreadNum);

int32_t SliceLayerInfoUpdate (sWelsEncCtx* pCtx,
                            SFrameBSInfo* pFrameBsInfo,
                            SLayerBSInfo* pLayerBsInfo,
                            const SliceModeEnum kuiSliceMode);
```

5. TestData

5.1 Test Script

5.1.1 Test script brief introduction

Git Repos:

<https://sqbu-github.cisco.com/huashi/OpenH264MemCheckTool>

Test Script:

For more detail about scripts, please refer to 5.1.2~5.1.4

- + [run_ParseMemCheckLog.sh](#)
parse memory allocate/free/leak size for one test case
- + [run_MememAnalyseForOneYUV.sh](#)
test all cases for one YUV and output memory/FPS statistic info
- + [run_Main.sh](#)
test all YUVs for all cases and output all memory/FPS statistic info
- + Other scripts as test tool called by above script
 - [run_GetYUVList.sh](#)
 - [run_GetYUVPath.sh](#)
 - [run_ParseYUVInfo.sh](#)

Test case in configuration file:

Using [default](#) setting in below cfg files which copied from openh264/testbin/

- + [Welsenc.cfg](#)
- + [Layer2.cfg](#)

And combine with below parameters in [run_MememAnalyseForOneYUV.sh](#)

[SliceMode=\(0 1 2 3\)](#)
[SliceNum=\(1 4 4 0 \)](#)
[ThreadNum=\(1 2 3 4\)](#)
[SliceSize=\(1500 600\)](#)

Test codec:

- + Origin design
[Codec version info, refer to Codec_OriginDesign/Codec_Info.txt](#)
- + New design
[Codec version info, refer to Codec_NewDesign/Codec_Info.txt](#)
- + New design with reallocate step ¼ MaxSliceNumOld
[Codec version info, refer to Codec_NewDesign_Step_1-4/Codec_Info.txt](#)
- + New design with reallocate step 1/3 MaxSliceNumOld
[Codec version info, refer to Codec_NewDesign_Step_1-3/Codec_Info.txt](#)

Test data:

- + Overall data structure
`TestSpace ="TestData/${Codec}/${Arch}"`
Example:
`TestData/Codec_OriginDesign/x86
TestData/Codec_OriginDesign/x64
TestData/Codec_NewDesign/x86
TestData/Codec_NewDesign/x64
TestData/Codec_NewDesign_Step_1-3/x86
TestData/Codec_NewDesign_Step_1-3/x64
TestData/Codec_NewDesign_Step_1-4/x86
TestData/Codec_NewDesign_Step_1-4/x64`
- + Test data file generate by **encoder** for one case
`TestMemLogFile="${TestSpace}/enc_mem_check_point_${YUVName}_${SliceMode[$i]}_${SliceNum[$i]}_${iThrdNum}_${iSlcSize}.txt"`
- + Test data file generate by **run_ParseMemCheckLog.sh** for one case
`TestAnalyseResult="${TestSpace}/MemAnalyseResut_${YUVName}_${SliceMode[$i]}_${SliceNum[$i]}_${iThrdNum}_${iSlcSize}.txt"`
- + Test data file for one YUV by **run_MememAnalyseForOneYUV.sh**
`TestReport="${TestSpace}/MemReport_For_${YUVName}.csv"`
- + Test data file for all YUVs by **run_Main.sh**
`TestReportForAllYUV="${TestSpace}/MemReport_For_AllYUVs.csv"`

Test data example:

For **new design** with **x64** encoder,
`TestSpace= TestData/Codec_NewDesign/x64/`

case is: slice mode = 1, slice num = 4,
thread num = 3, target bit rate = 600
YUV is : Zhuling_1280x720.yuv

- + memory file generated by encoder, rename and remove to:
`TestData/Codec_NewDesign/x64/enc_mem_check_point_Zhuling_1280x720.yuv_1_4_3_600.txt`
- + Memory analyse result by **run_ParseMemCheckLog.sh** is:
`TestData/Codec_NewDesign/x64/MemAnalyseResut_Zhuling_1280x720.yuv_1_4_3_600.txt`
- + Test report by **run_MememAnalyseForOneYUV.sh** for
Zhuling_1280x720.yuv is:
`TestData/Codec_NewDesign/x64/MemReport_For_Zhuling_1280x720.yuv.csv`
- + After test all YUV by **run_Main.sh**; report for all YUVs is:
`TestData/Codec_NewDesign/x64/MemReport_For_AllYUVs.csv`

5.1.2 Memory analyse for encode one case

```
run_ParseMemCheckLog.sh
*****
# brief: 1. parse memory allocate/free info based on memory statistic log
#         ./enc_mem_check_point.txt
#
#         2. to generate above log file,
#             need to enable below macro in codec/common/memory_align.h
#             #define MEMORY_CHECK 1
#
#         3. the output may contain the summary of memory allocate/free
#             statistic info during encoding process, and detail info of all
#             buffers which have been allocated/freed during/after'
#             encoding process
#
# usage: ./run_ParseMemCheckLog.sh $LogFile $OutFile $Option
#        ----LogFile: should be enc_mem_check_point.txt
#                  or other rename log file
#        ----OutFile: the final report file for memory statistic info
#        ----Option:
#                  1)MemCheck      : all detail info
#                  2)OverallCheck : summary info only
#
#
*****
```

Example:

Analyse new design x64 encoder's memory info, with
Case slice mode = 1, slice num = 4,
thread num = 3, target bit rate = 600
LogFile="TestData/Codec_NewDesign/x64/enc_mem_check_point_Zhuling_1280x720.yuv_1_4_3_600.txt"

OutFile="TestData/Codec_NewDesign/x64/MemAnalyseResut_Zhuling_1280x720.yuv_1_4_3_600.txt"

```
+ ./run_ParseMemCheckLog.sh ${LogFile} ${OutFile} OverallCheck
+ ./run_ParseMemCheckLog.sh ${LogFile} ${OutFile} MemCheck
```

if using OverallCheck, will only output below summary info

```
TestData/Codec_NewDesign/x64/MemAnalyseResut_Zhuling_1280x720.yuv_1_4_3_600.txt
MemLeakStatus False: AllocatedNum--FreeNum (116--116) :
AllocateSize==FreeSize==LeakSize (25615532==25615532==0)
```

If using OverallCheck, will output detail info about all buffer
allocate size, free size, allocate num, free num etc

5.1.3 memory analyse for one YUV

```
run_MememAnalyseForOneYUV.sh
*****
# brief: Analyse memroy allocate statistic info for one YUV with
#         all test cases
#
# usage: ./run_MememAnalyseForOneYUV.sh \$YUVName \$YUVDir \$TestSpace
#        ----YUVName: Test YUV's name
#        ----YUVDir: Test YUV's folder, script will search YUV
#                     within this folder
#        ----TestSpace: Test data output folder
#
*****
```

Test case for one YUV in script(you can modify it if you want):

```
SliceMode=(0 1 2 3)
SliceNum=(1 4 4 0 )
ThreadNum=(1 2 3 4)
SliceSize=(1500 600)
```

Example:

Test yuv with new design with arch=x64 encoder,

- ↳ **Step 1:**
copy encoder h264enc from Codec_NewDesign/x64/
- ↳ **Step 2:**
YUVName="Zhuling_1280x720.yuv"
YUVDir="../../YUV"
TestSpace="TestData/Codec_NewDesign/x64"
- ↳ **Step 3:**
../run_MememAnalyseForOneYUV.sh \${YUVName} \${YUVDir} \${TestSpace}

**Then all test data for Zhuling_1280x720.yuv is under folder
TestData/Codec_NewDesign/x64/**

Test report file for zhuling is:

TestData/Codec_NewDesign/x64/MemReport_For_Zhuling_1280x720.yuv.csv

5.1.4 Memory analyse for all YUVs

```
run_Main.sh
*****
# brief: Analyse memory allocate statistic info for all YUVs with
#         all test cases
#
# usage: ./run_Main.sh \$TestYUVListDir
#         ----TestYUVListDir: Test YUV's folder, script will search YUV
#                           in this folder
#
*****
```

Test all YUVs with all test cases using all encoder in encoder set

You can modify below test sets if you want

```
TestYUVList=(CiscoVT2people_320x192_12fps.yuv" \
             "xuemei_640x360.yuv" \
             "Zhuling_1280x720.yuv" \
             "desktop_dialog_1920x1080_i420.yuv" \
             "SlideShowFast_SCC_2880x1800.yuv")

TestCodecList=(Codec_NewDesign" \
               "Codec_NewDesign_Step_1-3" \
               "Codec_NewDesign_Step_1-4" \
               "Codec_OriginDesign")

TestCodecArchList=("x64" "x86")
```

And combine with cases in [run_MememAnalyseForOneYUV.sh](#)

```
SliceMode=(0 1 2 3)
SliceNum=(1 4 4 0 )
ThreadNum=(1 2 3 4)
SliceSize=(1500 600)
```

Example:

[./run_Main.sh .../.../YUV](#)

Then all test data will be generated under all below folders:

```
TestData/Codec_OriginDesign/x86
TestData/Codec_OriginDesign/x64
TestData/Codec_NewDesign/x86
TestData/Codec_NewDesign/x64
TestData/Codec_NewDesign_Step_1-3/x86
TestData/Codec_NewDesign_Step_1-3/x64
TestData/Codec_NewDesign_Step_1-4/x86
TestData/Codec_NewDesign_Step_1-4/x64
```

5.2 Final Test Data

All test data for all test encoder all arch wit all test YUVs for all cases:

<https://sqbu-github.cisco.com/huashi/OpenH264MemCheckTool/tree/master/TestData>

Final Test report for all YUVs:

<https://sqbu-github.cisco.com/huashi/OpenH264MemCheckTool/blob/master/NewDesignTestData.xlsx>

example for SlideShowFast_SCC_2880x1800.yuv

SICM	SlcN	PicW	PicH	ThrdN	SlcSz	Origin design		New design		Delta(%)	
						AllocSize	FPS	AllocSize	FPS	deltaSiz	deltaFPS(%)
3	0	2880	1800	1	1500	99424308	85.17	99171165	88.36	-0.25	3.75
3	0	2880	1800	1	600	99738905	80.77	100121689	86.73	0.38	7.38
3	0	2880	1800	2	1500	162263844	123.39	137297154	122.54	-15.39	-0.69
3	0	2880	1800	2	600	162263844	124.79	199850984	127.57	23.16	2.23
3	0	2880	1800	3	1500	170984608	151.67	149679510	160.58	-12.46	5.87
3	0	2880	1800	3	600	170984608	157.59	244683892	156.55	43.10	-0.66
3	0	2880	1800	4	1500	178796354	165.30	168923895	175.24	-5.52	6.01
3	0	2880	1800	4	600	178796354	166.49	221484277	180.50	23.88	8.41

-15.39% means that new design decrease overall allocate size;

43.10% means that new design increase overall allocate size due to reallocate modue which is not allowed in origin design

Final test summary:

New design most of time save the overall memory size,
FPS is the same with origin disign

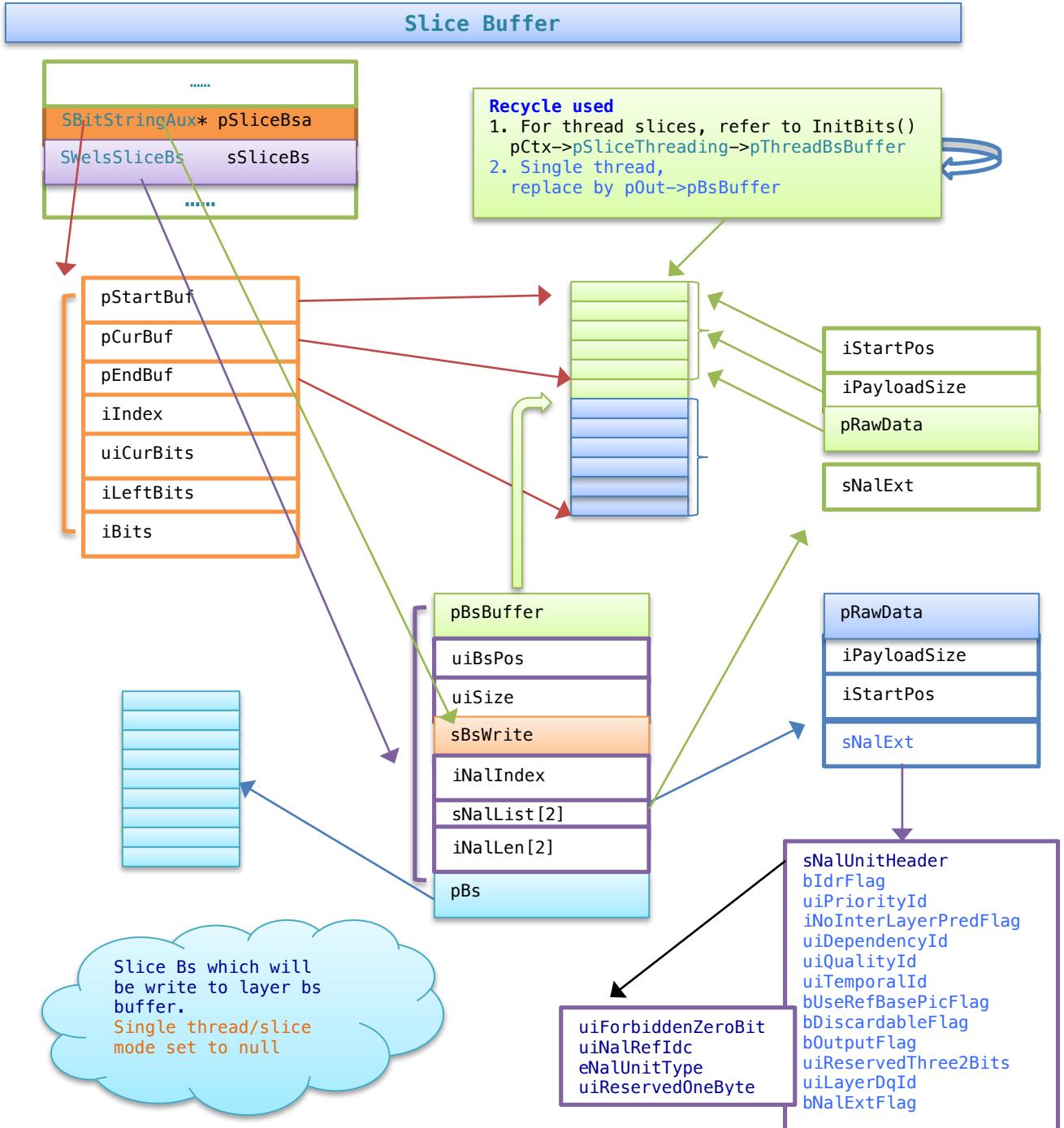
6. Code review and PRs

All new design pull requests list below:

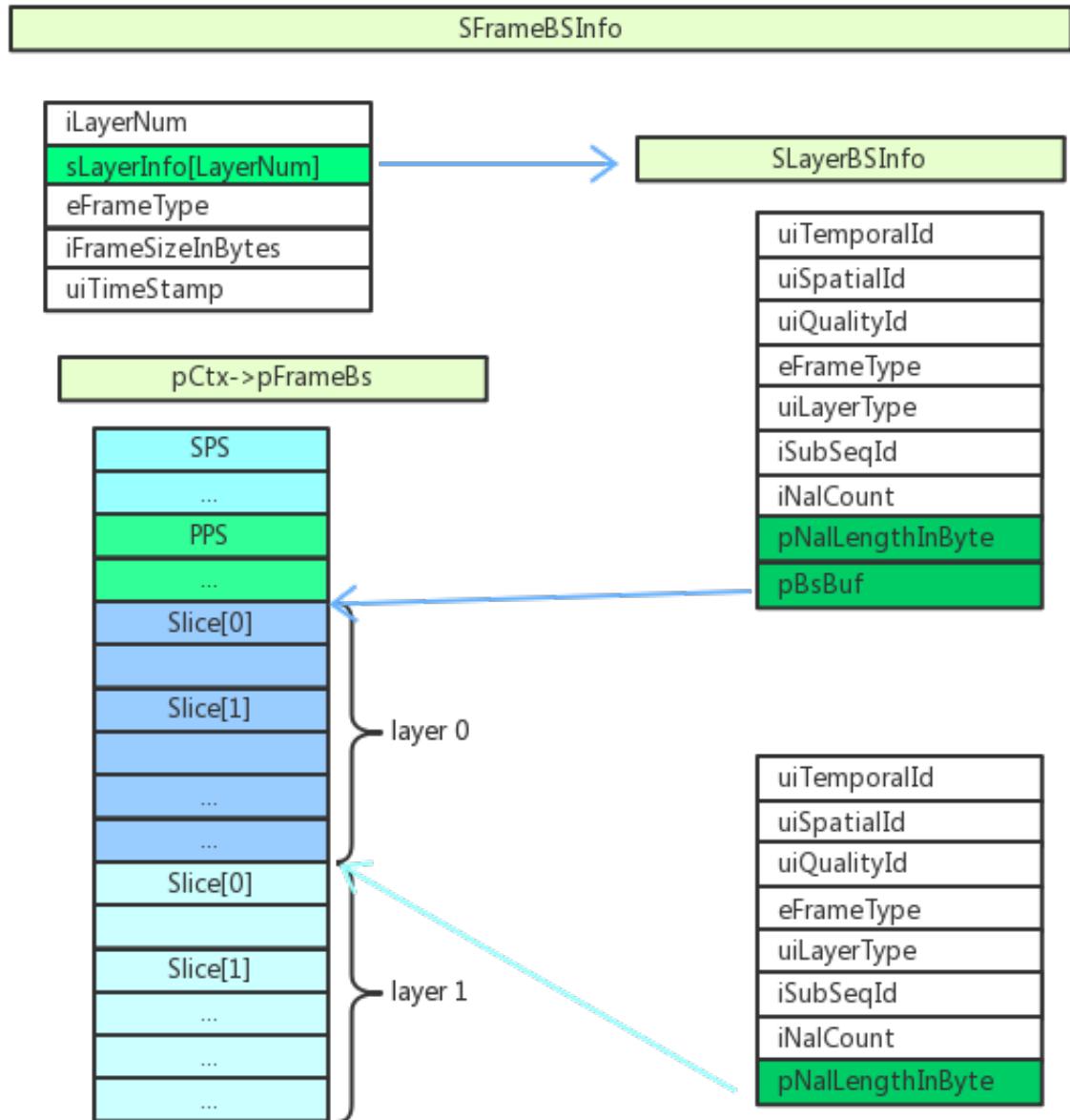
<https://github.com/cisco/openh264/pull/2698>
<https://github.com/cisco/openh264/pull/2699>
<https://github.com/cisco/openh264/pull/2700>
<https://github.com/cisco/openh264/pull/2704>
<https://github.com/cisco/openh264/pull/2705>
<https://github.com/cisco/openh264/pull/2706>
<https://github.com/cisco/openh264/pull/2714>
<https://github.com/cisco/openh264/pull/2714>
<https://github.com/cisco/openh264/pull/2716>
<https://github.com/cisco/openh264/pull/2719>
<https://github.com/cisco/openh264/pull/2720>

7. Buffer Structure

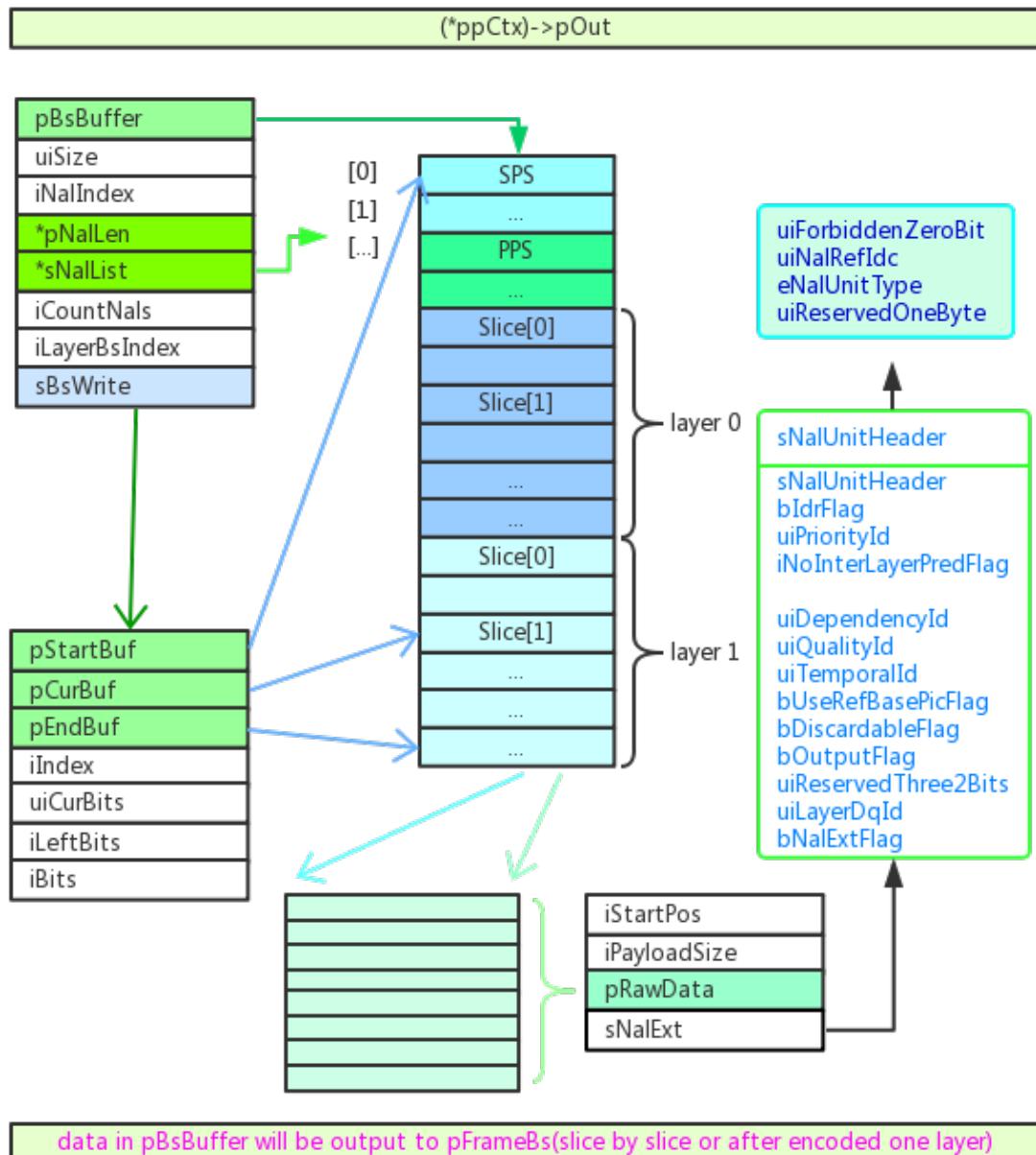
7.1. Slice Bs buffer



7.2. Frame bs buffer



7.3 pCtx-pOut



8. Other design

8.1. dynamic slice, slice boundary strategy

