Multimedia FAQ

Release 0.10.0

SOPHGO

Jun 19, 2024

目录

CHAPTER 1

Disclaimer

Legal Disclaimer

Notice

The purchased products, services and features are stipulated by the contract made between SOPHGO and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in

this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Technical Support

Address Floor 6, Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Website https://www.sophgo.com/

Email sales@sophgo.com

Phone +86-10-57590723 +86-10-57590724

CHAPTER 2

Release note

Version	Date of Release	Description
V0.1.0	2022.08.10	First release, contains SOPHON ffmpeg and SOPHON opencv
V0.2.4	2022.08.30	Official release, supplement documents
V0.6.0	2023.02.28	Added new features: watermark, mosaic, unified memory allocation
V0.6.1	2023.03.31	Add sample installation and use method
V0.7.0	2023.05.16	1, bm_opencv support imshow; 2, fix bug.
V0.7.1	2023.07.11	1, bm_ffmpeg supports jpeg loop dec; 2, new operator bayer2rgb.
V0.7.3	2023.10.18	1, fix bug.
V0.8.0	2023.08.01	1, fix bug.
V0.10.0	2024.04.11	1, Open the bmvid interface; 2, VPU dec supports external allocation of physical memory; 3, Update the video codec firmware.

CHAPTER 3

Disclaimer

Legal Disclaimer

Notice

The purchased products, services and features are stipulated by the contract made between SOPHGO and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in

this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Technical Support

Address Floor 6, Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Website https://www.sophgo.com/

Email sales@sophgo.com

Phone +86-10-57590723 +86-10-57590724

CHAPTER 4

Multimedia Customer FAQ

4.1 4K Image Problem

A: Some users need images with large resolutions, such as 8K. Since VPP only supports images with 4K resolution, after reading the images through OpenCV, it will automatically maintain the scale and zoom to size within 4K.

4.2 After Opencv reads the image and cvMat is converted to bmimage, users will call bmcv_image_vpp_convert to zoom and converse color space. However, the images received are inconsistent.

Analysis: the internal conversion matrix of opencv and the conversion matrix of bmcv_image_vpp_convert are inconsistent, so it requires to call bmcv_image_vpp_csc_matrix_convert and specify CSC_YPbPr2RGB_BT601 to maintain consistency.

4.3 Opencv imread has a performance problem when reading images.

Analysis: if the image size is less than 16x16, or jpeg in the progressive format, the processor cannot speed up. As a result, the processor path is taken, and the users find that the image decoding is not accelerated.

4.4 There is a performance problem with VideoWriter.write. Some users complain that saving files is slow.

Analysis: for now, it is normal to use YUV acquisition and write a frame of data between 10-20ms.

4.5 Blocking problem of ffmpeg

Analysis: if the avframe is not released in time, it will lead to blocking. There is a circular buffer inside the vpu.

4.6 The time to call the uploadMat/downloadMat interface

Analysis: after creating a cv::Mat and calling the interface in cv::bmcv to make some processing, the contents of the device memory are changed. At this time, downloadMat is needed to synchronize the device memory and system memory. After calling opencv native interfaces such as cv::resize, the content of system memory changes. uploadMat is required to synchronize device memory and system memory.

4.7 How to get the timestamp of video frames under opencv?

A: Opencv native provides an interface to obtain timestamp. The timestamp of the current frame can be obtained after each frame processed by cap.read().

The code is as follows:

```
Mat frame;
cap.read(frame);
*/ Obtain timestamp, and the return value is double */
int64_t timestamp = (int64_t)cap.getProperty(CAP_PROP_TIMESTAMP);
```

4.8 Solution of frequent network disconnection of videocapture in about 5 minutes under SA3 opencv

A: The problem of "connection timeout" often occurs when the RTSP data is connected for 3-5 minutes in UDP mode. The final solution to this problem is to update the latest routing board software. The verification method can be tested with TCP. If there is no problem with TCP, it can be confirmed that it is such a problem.

The method of using TCP is shown below:

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;tcp"
```

Execute the application (if sudo is used for execution, sudo - E is required to bring the environment variable) Note: the latest middleware-soc will use TCP as the default protocol. If the original users need to use UDP transmission protocol, it is necessary to guide the user to set it in the following way.

The method of using UDP:

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;udp"
```

Execute the application (if sudo is used for execution, sudo - E is required to bring the environment variable)

UDP applicable environment: when the network bandwidth is narrow, such as 4G/ 3G and other mobile communication systems, UDP is more appropriate.

TCP applicable environment: application scenarios with sufficient network bandwidth, high requirements for video blurred screen and small delay requirements are suitable for TCP.

4.9 How to get the original timestamp in RTSP

A: The default RTSP timestamp obtained in opencv starts from 0. If you want to obtain the original timestamp in RTSP, you can use the environment variable to control it, and then obtain it according to question 1

```
export OPENCV_FFMPEG_CAPTURE_OPTOINS="keep_rtsp_timestamp;1"
```

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="keep_rtsp_timestamp;1|buffer_size;
↪1024000|max_delay;500000|stimeout;20000000"
```

4.10 How to judge the cause of video blurred screen

A: The video blurred screen mentioned here is a long-time blurred screen. The occasional blurred screen may be caused by network data transmission error, which is not a controllable position of the application code. If the video blurred screen occurs for a long time, it is very likely that the video frame is not read in time, resulting in the overflow of socket recv buffer after the internal cache is full.

1. increase rmem_max to 2M. If the blurred screen disappears at this time, it indicates that the data processing jitter of the application is very large, and the buffer queue should be increased for smoothing

```
echo 2097152 > /proc/sys/net/core/rmem_max
```

2. use netstat -na, which is generally in the following format. Find the port of RTSP (UDP will print in the application, and the target RTSP address can be used for TCP). Recv-Q and Send-Q here should be 0 or not full under normal conditions. If Recv-Q often has a large number, it means to overflow. Generally, there will be no problem with Send-Q. If Send-Q is also large, the network driver is likely to fail.

```
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN
```

4.11 Unable to connect to RTSP

A: You can use the ffmpeg inherent command to test the connection: (the URL is the RTSP connection address)

```
ffmpeg -rtsp_transport tcp -i url -f rawvideo -y /dev/null
or
ffmpeg -rtsp_transport udp -i url -f rawvideo -y /dev/null
If the above cannot be connected successfully, please check the network.
```

4.12 Confirm whether the decoder works normally: (URL is the file name or RTSP connection address)

A:

```
ffmpeg -i url -f rawvideo -y /dev/null
```


4.13 Confirm whether the OpenCV interface between the decoder and vpp works normally:

A:

```
vidmulti number_of_instances url1 url2
```

4.14 Final debugging method with incorrect decoding or unable to decode

A: If the problem still cannot be solved on site after various debugging, you can dump the data before and after the problem by opening the environment variable for further analysis

In PCIE mode

```
export BMVID_PCIE_DUMP_STREAM_NUM=1000
```

The data of dump is under /data/ (the folder needs to be created with writing permission). The data of dump is stored according to core index and instance index.

In SoC mode

```
echo “0 0 1000 100” > /proc/vpuinfo
```

(dump the bitstream data of the first instance in the first core)

This configuration will circularly store 1000 frames of data between the two files. When a problem occurs, copy the 1000 frames before and after the two files. The two files are stored in /data/core% dinst%d_stream%d.bin. (Note: PCIe needs to prepare the written permission of /data/ directory in advance)

4.15 Determine whether RTSP works normally

A:

Method 1: play video through VLC (recommended), and set TCP and UDP modes respectively

Method 2: run vidmutil executable file. The default mode of vidmutil is UDP. Use TCP mode by setting the environment variable.

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;tcp|buffer_size;1024000|max_
↪delay;50000"
sudo -E ./vidmulti thread_num input_video [card] [enc_enable] input_video [card] [enc_
↪enable]...
```

4.16 Verification of disconnection of RTSP stream played by high-density server

A: Users can use VLC to play the same video. At the same time, check whether there is a disconnection in VLC playback. Pay attention to setting the buffer size of VLC.

4.17 Verify whether the video output from the current RTSP service has a blurred screen

A: Use VLC to play the video for a period to see if the video has a blurred screen

4.18 Check whether the RTSP service is streaming in real-time

A: Check whether the currently playing file is being sent through the RTSP server log.

4.19 Support status of old OpenCV interfaces such as cvQueryFrame

A:

Some users use the C interface of the old version of OpenCV. The interface list is as follows

```
void cvReleaseCapture( CvCapture** pcapture )

IplImage* cvQueryFrame( CvCapture* capture )

int cvGrabFrame( CvCapture* capture )

IplImage\* cvRetrieveFrame( CvCapture* capture, int idx )

double cvGetCaptureProperty( CvCapture* capture, int id )

int cvSetCaptureProperty( CvCapture* capture, int id, double value )

int cvGetCaptureDomain( CvCapture* capture)

CvCapture * cvCreateCameraCapture (int index)

CvCapture * cvCreateFileCaptureWithPreference (const char * filename, int apiPreference)

CvCapture * cvCreateFileCapture (const char * filename)
```

Most of these interfaces are supported. Only the interface whose return value is iplimage cannot be supported. This is because the ion memory type at the bottom of our hardware is saved in the uMatData type of MAT, while the iplImage type has no uMatData data structure.

Therefore, for users who currently use cvQueryFrame interface, it is recommended that users use cap.read() interface seals a C function interface whose return value is mat data. Do not directly call the interface of the old version of OpenCV.

4.20 For YUV format conversion not supported by VPP hardware, what software method is the fastest?

A:

An internally enhanced version of libyuv is recommended.

Compared with the original version, many NEON64 optimized format conversion API functions are added. It contains many functions related to JPEG YCbCr.

Location: /opt/sophon/libsophon-current/lib/

4.21 Where is the corresponding format in libyuv of BGR format in OpenCv? What about the RGB format in OpenCv?

A:

- The corresponding format in libyuv of BGR format in OpenCV is RGB24.
- The corresponding format in libyuv of RGB format in OpenCV is RAW.

4.22 What should be paid attention to when using libyuv to process JPEG output or input?

A:

If dealing with JPEG output or input, you need to use functions such as J400, J420, J422 and j444, otherwise the output result will have color difference.

The reason is that the format conversion matrix of JPEG is different from that of video.

4.23 ffmpeg & opencv supports gb28181 protocol, and the incoming URL address is in the form of

A:

UDP Real-time Stream Address

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?
↪deviceid=35018284001310090010#localid=1247879287116362497 9#localip=172.10.18.201
↪#localmediaport=20108
34020000002019000001:123456@35.26.240.99:5666: sip server national standard code:␣
↪password of
sip server @IP address of sip server: port of sip server
deviceid：20-bit code of the front device
localid：local 20-bit code, optional
localip：local IP, optional If you do not set it, you will get the ip of eth0. If there is no eth0,
you need to set it manually
localmediaport：the video stream port of the media receiving end. Port mapping is required.
Map two ports (rtp:11801, rtcp:11802), the in and out of the two port mappings should be the␣
↪same. One core board port cannot be repeated.
```

UDP playback stream address

```
gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid=\
↪35018284001310090010#devicetype=3#localid=1247879287116362497 9#localip=172.10.18.
↪201#localmediaport=20108#begtime=20191018160000#endtime=20191026163713
34020000002019000001:123456@35.26.240.99:5666：sip server national standard code: sip␣
↪server
password@sip server IP address: SIP server port
deviceid：20-bit code of the front device
devicetype：recording storage type
localid：local 20-bit code, optional. If you do not set it, you will get the IP of eth0. If there is no
eth0, you need to set it manually
localip：local IP, optional
localmediaport：the video stream port of the media receiver. Port mapping is required. Two␣
↪ports
(rtp:11801, rtcp:11802) are mapped. The in and out of the two ports should be the same. One
core board port cannot be repeated.
begtime：start time of recording
endtime：end time of recording
```

TCP Real-time Stream Address

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?
↪deviceid=35018284001310090010#localid=1247879287116362497 9#localip=172.10.18.201
34020000002019000001:123456@35.26.240.99:5666：sip server national standard code: sip␣
↪server
password@sip server ip address: sip server port
deviceid：20-bit code of the front device
localid：local 20-bit code, optional
```

(continues on next page)

(continued from previous page)

localip: local IP, optional. If you do not set it, you will get the IP of eth0. If there is no eth0, you need to set it manually

TCP playback stream address

```
gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?
↪deviceid=35018284001310090010#devicetype=3#localid=12478792871163624979
↪#localip=172.10.18.201#begtime=20191018160000#endtime=20191026163713
34020000002019000001:123456@35.26.240.99:5666：sip server national standard code: sip␣
↪server
password@sip server IP address: sip server port
deviceid：20-bit code of the front device
devicetype :recording storage type
localid :local 20-bit code, optional
localip :local IP, optional. If you do not set it, you will get the IP of eth0. If there is no eth0, you
need to set it manually
begtime :start time of recording
endtime :end time of recording
```

Attention:

1. The default mode of streaming media transmission is UDP. If you use TCP to obtain real-time stream or playback stream, you need to specify the display.

   Ffmpeg specifies the TCP mode for interface calls. Through av_dict_set to set gb28181_transport_rtp to tcp.

   Opencv is specified by setting environment variables

   ```
   export OPENCV_FFMPEG_CAPTURE_OPTIONS="gb28181_transport_rtp;tcp"
   ```

2. If the internal ip/port cannot be accessed externally by using UDP, the local media port needs to do port mapping, which requires two rtp and rtcp.

3. When doing port mapping, try not to use a port number that is too large, we recommend a port number in the range 10000-20000 and a maximum port number of 65536 for socket. But in many cases, the port number is limited by many factors. If the port number is too large, a [bind failed] error may occur.

4.24 Now the default in opencv is to use ION memory as the data space of MAT. How to specify that MAT objects are created and used based on system memory?

A:

```
using namespace cv;

Mat m; m.allocator = m.getDefaultAllocator();    // get system allocator
```

Then you can call various mat functions normally, such as m.create() m.copyto(), and then allocate memory according to the specified allocator.

```
m.allocator = hal::getDefaultAllocator();  // get ion allocator
```

You can resume using the ion memory allocator to allocate memory.

4.25 FFMPEG JPEG example for encode and transcode.

A:

- **a jpeg encoder example for command line**

```
ffmpeg -c:v jpeg_bm -i src/5.jpg -c:v jpeg_bm  -is_dma_buffer 1 -y 5nx.jpg
```

- **mjpeg transcoder examples for command line**

```
ffmpeg -c:v jpeg_bm -num_extra_framebuffers 2 -i in_mjpeg.avi -c:v jpeg_bm -is_dma_
↪buffer 1 -y test_avi.mov

ffmpeg -c:v jpeg_bm -num_extra_framebuffers 2 -i in_mjpeg.mov -c:v jpeg_bm -is_dma_
↪buffer 1 -y test_mov.mov
```

4.26 How to read bitstream from input buffer in FFMPEG?

A:

There is an example in FFMPEG source code. /opt/sophon/sophon-ffmpeg-latest/share/ffmpeg/examples/avio_reading.c (or http://www.ffmpeg.org/doxygen/trunk/doc_2examples_2avio_reading_8c-example.html)

This case makes libavformat demuxer access media content through a **custom AVIOContext read callback** instread of file, rtsp, etc protocols defined in FFMPEG.

This is an example in middleware-soc about using avio + jpeg_bm to decode still jpeg picture. (/opt/sophon/sophon-ffmpeg-latest/share/ffmpeg/examples/avio_decode_jpeg.c)

4.27 Read images from memory and use AVIOContext * avio =avio_alloc_context() and avformat_open_input() to initialize. It is found that the initialization time is 290ms; However, if the image is read locally, it is only 3ms. Why does initialization take so long? How to reduce initialization time?

A:

```
ret = avformat_open_input(&fmt_ctx, NULL, NULL, NULL);
```

Here is the simplest call. Therefore, avformat will read the data internally and traverse all the data to confirm the data format in avio.

On the premise that you already know the demuxer you need to use (for example, you know that the demuxer of JPEG is MJPEG), if you want to avoid reading data in this function and this kind of matching, you can try changing the code to the following.

```
AVInputFormat* input_format = av_find_input_format("mjpeg");

ret = avformat_open_input(&fmt_ctx, NULL, input_format, NULL);
```

4.28 how to know the jpeg demuxer name supported in FFMPEG?

A:

root@linaro-developer:~# ffmpeg -demuxers | grep jpeg
D jpeg_pipe piped jpeg sequence
D jpegls_pipe piped jpegls sequence
D mjpeg raw MJPEG video
D mjpeg_2000 raw MJPEG 2000 video
D mpjpeg MIME multipart JPEG
D smjpeg Loki SDL MJPEG

4.29 how to know all bm hardware decoder names supported in FFMPEG?

A:

```
root@linaro-developer:/home/sophgo/test# ffmpeg -decoders | grep _bm
V..... avs_bm bm AVS decoder wrapper (codec avs)
V..... cavs_bm bm CAVS decoder wrapper (codec cavs)
V..... flv1_bm bm FLV1 decoder wrapper (codec flv1)
V..... h263_bm bm H.263 decoder wrapper (codec h263)
V..... h264_bm bm H.264 decoder wrapper (codec h264)
V..... hevc_bm bm HEVC decoder wrapper (codec hevc)
V..... jpeg_bm BM JPEG DECODER (codec mjpeg)
V..... mpeg1_bm bm MPEG1 decoder wrapper (codec mpeg1video)
V..... mpeg2_bm bm MPEG2 decoder wrapper (codec mpeg2video)
V..... mpeg4_bm bm MPEG4 decoder wrapper (codec mpeg4)
V..... mpeg4v3_bm bm MPEG4V3 decoder wrapper (codec msmpeg4v3)
V..... vc1_bm bm VC1 decoder wrapper (codec vc1)
V..... vp3_bm bm VP3 decoder wrapper (codec vp3)
V..... vp8_bm bm VP8 decoder wrapper (codec vp8)
V..... wmv1_bm bm WMV1 decoder wrapper (codec wmv1)
V..... wmv2_bm bm WMV2 decoder wrapper (codec wmv2)
V..... wmv3_bm bm WMV3 decoder wrapper (codec wmv3)
```

4.30 How to show the decoder info, for example, jpeg_bm, in FFMPEG?

A:

```
root@linaro-developer:/home/sophgo/test# ffmpeg -h decoder=jpeg_bm
Decoder jpeg_bm [BM JPEG DECODER]:
General capabilities: avoidprobe
Threading capabilities: none
jpeg_bm_decoder AVOptions:
-bs_buffer_size <int> .D.V..... the bitstream buffer size (Kbytes) for bm jpeg decoder
(from 0 to INT_MAX) (default 5120)
-chroma_interleave <flags> .D.V..... chroma interleave of output frame for bm jpeg
decoder (default 0)
-num_extra_framebuffers <int> .D.V..... the number of extra frame buffer for jpeg
decoder (0 for still jpeg, at least 2 for motion jpeg) (from 0 to INT_MAX) (default 0)
```

4.31 How to show the encoder info, for example, jpeg_bm, in FFMPEG?

A:

root@linaro-developer:/home/sophgo/test# ffmpeg -h encoder=jpeg_bm
Encoder jpeg_bm [BM JPEG ENCODER]:
General capabilities: none
Threading capabilities: none
Supported pixel formats: yuvj420p yuvj422p yuvj444p
jpeg_bm_encoder AVOptions:
-is_dma_buffer <flags> E..V..... flag to indicate if the input frame buffer is DMA buffer (default 0)

4.32 A jpeg encoder example for calling api function

A:

```
AVDictionary* dict = NULL;

av_dict_set_int(&dict, "is_dma_buffer", 1, 0);

ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

4.33 example to set decoder jpeg_bm when calling ffmpeg api for decoding still jpeg picture

A:

```
AVDictionary* dict = NULL;

/* The output of bm jpeg decoder is chroma-interleaved,for example, NV12 */

av_dict_set_int(&dict, "chroma_interleave", 1, 0);

/* The bitstream buffer is 3100KB(less than 1920x1080x3),
/* assuming the max dimension is 1920x1080 */
av_dict_set_int(&dict, "bs_buffer_size", 3100, 0);
/* Extra frame buffers: 0 for still jpeg, at least 2 for mjpeg */
av_dict_set_int(&dict, "num_extra_framebuffers", 0, 0);

ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

4.34 example to set decoder jpeg_bm when calling ffmpeg api for decoding motion jpeg picture

A:

```
AVDictionary* dict = NULL;

/* The output of bm jpeg decoder is chroma-separated, for example, YUVJ420 */

av_dict_set_int(&dict, "chroma_interleave", 0, 0);

/* The bitstream buffer is 3100KB,
/* assuming the max dimension is 1920x1080 */
av_dict_set_int(&dict, "bs_buffer_size", 3100, 0);
/* Extra frame buffers: 0 for still jpeg, at least 2 for mjpeg */
av_dict_set_int(&dict, "num_extra_framebuffers", 2, 0);

ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

4.35 Is the decoding performance of BM168x different in H264/H265? If users adjust the bit rate, how many channels can they solve at most? Is there a corresponding data reference?

A:

264 and 265 have the same number of decoding channels.

The bit rate will affect the decoding frame rate. This change needs to be measured. For example，The BM1684 decoding can reach 960fps for the monitoring code stream. This kind of monitoring code stream has no B frame, the scene fluctuation is small, and the bit rate is basically 2~4Mbps. If it is a movie or other high bit rate, such as 10Mbps, 20MBps or more, it will have a significant impact. The specific size needs to be measured.

The influence of resolution on decoding frame rate can be converted in proportion. 960fps is for 1920x1080 HD resolution.

4.36 Whether the number of decoding channels of BM168x can be increased by frame extraction

A:

The frame extraction provided in OpenCV is done in the decoded result, not only the frame extraction concept of decoding I/P frame. The frame extraction decoding here is mainly to ensure the uniformity of the number of frames, so that the subsequent analysis and processing are carried out at equal intervals. This is a solution designed for when

the subsequent model analysis is complex and cannot detect every frame, but it cannot achieve the effect of increasing the number of decoding channels.

Here to explain the reason why we do not provide the frame extraction function of only decoding I/P frames. If only I and P frames are solved, the interval of frame extraction completely depends on the coding structure of the code stream, so it is difficult to control the performance. For example, if there is no B frame in the monitoring code stream, it is equivalent to no frame extraction. If the user can control the encoder, the more practical way is to directly reduce the coding frame rate of the encoder, for example, to 15fps, so that the number of decoding channels can be directly increased; On the contrary, if the user has no way to control the encoder, the frame extraction method of only decoding IP frames cannot achieve the effect of increasing the number of decoding channels.

4.37 Does it support H264/H265 video parsing in avi, f4v, mov, 3gp, mp4, ts, asf, flv, mkv package format?

A: We use ffmpeg to support these encapsulation formats, and we also support what ffmpeg supports. After investigation, these encapsulation formats are supported by ffmpeg. However, the support of the packaging format for H265/264 depends on the standard definition of the packaging format. For example, there is no support for h265 in flv standard. At present, domestic ones are expanded by themselves.

4.38 Whether it supports PNG, JPG, BMP, JPEG and other image formats

A: In addition to JPEG 2000, the jpg/jpeg format has many levels of its own standards. We support it in a combination of software and hardware. Except for a few parts of JPEG baseline, they are supported by hardware acceleration, and other JPEG/JPG/BMP/PNG are supported by software acceleration. The main interface is opencv/ffmpeg library.

4.39 Why are there so many warnings in Valgrind memory check that affect the debugging of applications?

A:

Every time our version is released, we will check the memory leak with Valgrind. If there is a memory leak, we will check and fix it. The reason why some warnings are not removed is that most of these warnings are memory uninitialized. If we add initialization to each of these memories, it will significantly reduce the speed and performance. Moreover, we have confirmed that the subsequent operations are carried out after the hardware assigns a value to them. For such warnings, we will not eliminate them.

In order to avoid the impact of too many warnings on the upper application debugging, it is recommended to use the suppression function of Valgrind. You can filter the configuration file to avoid the Valgrind warnings generated by our module, facilitating the upper application debugging its own program.

4.40 The physical memory (heap2) allocation failure occurs when using the video write encoding of OpenCV

A:

Confirm the size of heap2. If the default size of heap2 is tens of MB, you need to set heap2 size to 1G. At present, the factory default configuration is 1G.

Update_boot_info can be called to query heap2 size update_boot_info –heap2_size=0x40000000 –dev=0x0 sets heap2 size to 1G. Please reinstall the drive after setting.

4.41 The imread jpeg decoding result of Bm_opencv is different from that of native OpenCV, with error

A: Yes. Native opencv uses libjpeg-turbo, whereas bm_opencv uses jpeg hardware acceleration unit in bm168x to decode.

The error mainly comes from the process of converting the decoded output YUV to BGR. 1) YUV needs up sampling of YUV444 before BGR conversion. There is no standard to enforce the unified practice of this upsample. Jpeg-turbo provides the default Fancy upsample and the fast replication upsampling algorithm. Original opencv uses the fast copy upsampling algorithm in the cvtColor function, while the default fancy upsample of libjpeg-turbo is used in imread and imdecode; The BM168x hardware acceleration unit adopts the fast replication algorithm. 2) The conversion from YUV444 to BGR is a floating-point operation, and the accuracy of floating-point coefficients will have +/-1 error. Among them, 1) is the main source of error.

This error is not an error, it is caused by the fact that both sides use different upsample algorithms, even libjpeg-turbo provides both upsample algorithms.

The above errors can lead to a decrease in the accuracy of the AI model, and if users are concerned about the difference between the two, we suggest two solutions:

1) Set export USE_SOFT_JPGDEC=1 can specify that libjpeg-turbo is still used for decoding. However, this will cause the processor loading to go up. Not recommended.
2) It is possible that the model relied too much on the open source opencv decoding results in the past, and the decoding results of bm_opencv can be used to retrain

the model to improve the applicability of the model parameters. to improve the applicability of the model parameters.

3) The data from the test toolset can be reprocessed using the djpeg tool provided by libjpeg-turbo, and then the model can be trained with the processed data. The command of djpeg is as follows: ./djpeg -nosmooth -bmp -outfile xxxxx.bmp input.jpg

Just run the above command and use the regenerated bmp file as the training data set.

4.42 How to check the memory, usage and other status of vpu/jpu

A:

In PCIe mode, you can use the following method to view:

```
cat /proc/bmsophon/card0/bmsophon0/media

cat /proc/bmsophon/card0/bmsophon0/jpu
```

In SoC mode, you can use the following method to view:

```
cat /proc/vpuinfo

cat /proc/jpuinfo
```

4.43 When the video supports 32 channels or more, it is reported that the video memory is not enough. How to optimize the memory space?

A:

Under the PCIe board, there is 3G video memory. It is more than enough to support 32 or more channels. However, in SoC mode, the default configuration of video memory is 2G, which is more than enough for normal use in 16 channels. However, in 32 channels, video needs to be carefully designed at the application level without any waste.

If the decoding uses FFMPEG framework, first ensure that the video output format uses the compression format, i.e., output_format 101. If the Opencv framework is used, the internal compression format has been used by default;

Secondly, if the application does not directly press into the queue after obtaining the decoded output avFrame, but converts it to RGB or uncompressed data and then caches it, av_dict_set extra_frame_buffer_num is 1 (the default is 2). Opencv internally will be optimized by default in the latest version.

Finally, if the above optimization is still not enough, you can consider changing the dtb setting in SoC mode to allocate more memory for video misappropriation. Of course,

other modules should reduce memory accordingly. This should be allocated from the perspective of the system.

4.44 How does mat allocate device memory and system memory in Opencv?

A:

Due to the influence of design, there are many details of this problem, which can be explained from three aspects.

1. In SoC mode, the device memory and system memory are the same physical memory, which is managed through the ION memory of the operating system, and the system memory is obtained by mmap of ION memory. In PCIe mode, device memory and system memory are two physical memories. Device memory refers to the memory on the BM168x card, and system memory refers to the operating system memory on the server. If users want to only open system memory and keep consistent with open-source opencv, please refer to the answer of FAQ26.
2. In SOPHON opencv, by default, both device memory and system memory will be opened up, in which the system memory is placed in mat.u->data or mat.data, and the device memory is placed in mat.u->addr. Only the following situations will not open the device memory, but only provide the system memory:
   - When data is developed externally and provided to mat. That is, when declared in the following way:

     ```
     Mat mat(h, w, type, data); 或 mat.create(h, w, type, data);
     ```

   - In SoC mode, when type does not belong to one of (CV_8UC3, CV_32FC1, CV_32FC3). Pay special attention that here CV_8UC1 is not open, which is to ensure that our OpenCV can pass the opencv_test_core consistency verification.
   - When the width or height is less than 16. Because the hardware does not support this kind of width and height.
3. In SoC mode of BM1682、BM1684 and BM1684x, the device memory of CV_8UC3 assigned by mat will be automatically 64 aligned, that is, the allocated memory size must be 64 aligned (Note: only for cv_8uc3 in SoC mode, and only for BM1682、BM1684 and BM1684x). In PCIe mode, the allocated memory is byte aligned.

4.45 How to deal with the situation that the image format/size conversion in ffmpeg leads to backoff or wrong sequence during video playback?

A: The bottom layer of ffmpeg maintains a queue of avframes as the input source of the encoder during encoding. The data in the queue should be valid during encoding. If scaling or pixel format conversion is required after decoding, attention should be paid to the data validity and release of the avframe sent to the encoder.

For example, ff_bmcv_transcode decodes to output src-avframe, converts to src-bm_image, converts to pixel format or scales to dst-bm_image and finally transfers back to dst-avframe to encode. In this process, the device memory of src-avframe and src-bm_image is the same, and the device memory of dst-avframe and dst-bm_image is the same. After getting dst-bm_image, the device memory of src_avframe and src-bm_image can be release (one of them can be released to release the device memory). After the input dst-bm_image as the encoder converting to dst-avframe, its device memory cannot be released (the common exceptional situation is that it is released when the reference count of dts-bm_image is 0 at the end of the function). If dst-bm_image is released. At this time, there will be problems with encoding with dst-avframe.

Solution: the pointer of dst-bm_image is malloc, a piece of memory, and then passes it to av_buffer_create, which ensures dst-bm_image at the end of the function will not be decremented by 1. The method of releasing is to transfer the pointer of dst-bm_image of malloc via av_buffer_create is passed to the released callback function. When the dst-avframe reference count is 0, the callback function will be called to connect the pointer of malloc with dst-bm_image to release together. Please refer to example ff_bmcv_transcode/ff_avframe_convert.cpp for details.

4.46 The startup device executes a function slowly for the first time. The restart process runs normally again

Description: After the equipment is powered on and the program is executed for the first time, the function processing time is long. When the program is executed again, the operation is normal.

Solution: Make a verification first. If it can be reproduced without restarting, it means that the file cache slows it down.

1. After power on, if the first execution is slow and the second execution is normal, enter the root user
2. Clear cache echo 3 > /proc/sys/vm/drop_caches
3. If the program is executed again and runs slowly, it can be determined that it is caused by cache.

4.47 Opencv mat creation fails, showing "terminate called after throwing an instance of 'CV:: exception' what(): opencv (4.1.0)··· matrix.cpp:452: error: (- 215:Assertion failed) u! = 0 in function 'create' "

A:

This error is mainly due to the failure of device memory allocation. There are two reasons for failure:

1. The number of handles exceeds the system limit, which may be due to handle leakage or the number of system handles is set too small. You can confirm it with the following methods:

   View the maximum number of system defined handles:

   ```
   ulimit -n
   ```

   View the number of handles used by the current process:

   ```
   lsof -n|awk 'print $2' |sort|uniq -c|sort -nr|more
   ```

2. The device is out of memory. It can be confirmed by the following methods:

   · In SoC mode

   ```
   cat /sys/kernel/debug/ion/bm_vpp_heap_dump/summary
   ```

   · In PCIe mode, bm-smi tool can view the memory space of the device

Solution: after eliminating the memory leakage or handle leakage of the code itself, you can solve the problem of handle limitation by increasing the maximum number of handles in the system: ulimit -HSn 65536

If the device memory is not enough, you need to optimize the program to reduce the occupation of device memory, or modify the memory layout in the dts file to increase the corresponding device memory. For details, please refer to the instructions in the SM5 User Guide.

4.48 When opencv transfers bm_image, it reports the error "Memory allocated by user, no device memory assigned. Not support BMCV!"

A: This kind of error usually occurs in soc mode. The converted Mat only allocates system memory and does not allocate device memory, and bm_image requires device memory, so the conversion fails. In pcie mode, the device memory will be automatically allocated inside the interface, so this error will not be reported.

Mats that cause such problems are usually attached by externally allocated data memory, that is, call Mat(h, w, data) or Mat.create(h, w, data) to create, and data!=NULL, assigned externally.

For this case, since bm_image must require device memory, the solution is:

1. Generate a new Mat, create device memory by default, then copy it once with copyTo(), move the data to the device memory, and then re-use this Mat to convert to bm_image
2. Create bm_image directly, and then use bm_image_copy_host_to_device to copy the data in Mat.data to the device memory of bm_image.

4.49 After Opencv uses the memory data, width and height of the existing Mat to create a new Mat, the image data saved by the new Mat is wrong and the display is abnormal.

A: The wrong line of the saved image is usually caused by the loss of step information in Mat.

Generally, an existing Mat is used to generate a new Mat, and when memory reuse is required, it can be directly assigned to the new Mat for simple implementation, such as Mat1 = Mat2.

But in some cases, such as some clients are limited by the architecture, function parameters can only be passed by C-style pointers, and only the data pointer, rows, and cols members in the Mat can be used to restore the Mat. At this time, you need to pay attention to the setting of the step variable. By default, it is the AUTO_STEP configuration, that is, each row of data is not filled with data. But in many cases, after opencv processing, it will cause padding data in each row. like,

1. In soc mode, our Mat considers the execution efficiency. When creating the memory of the Mat, each row of data will be aligned to 64 bytes to meet the needs of hardware acceleration (only in soc mode)
2. Intrinsic operations of opencv, like this Mat is a submatrix of another Mat (i.e. the selected area of the rect), or other operations that may result in padding.

Therefore, according to the definition of opencv, the general processing method is to specify step when generating a new Mat, as shown below:

```
cv::Mat image_mat = cv::imread(filename,IMREAD_COLOR,0);
cv::Mat image_mat_temp(image_mat.rows,image_mat.cols,CV_8UC3,image_mat.data,image_
↪mat.step[0]);
cv::imwrite("sophgo1.jpg",image_mat_temp);
```

4.50 In soc mode, the client gets AVframe when decoding with ffmpeg, and copies data[0-3] to the system memory. It is found that the copy time is about 20ms, and the copy of the same amount of data in two addresses in the system memory only takes 1-3ms.

A: The reason for the above problem is that the system disables cache by default in ffmpeg, so the performance of processor copy is very low, and enabling cache can achieve the same speed as system memory copying each other. The cache can be enabled using the following interface.

```
av_dict_set_int(&opts, "enable_cache", 1, 0);
```

However, this direct copy data storage takes up a lot of memory, bandwidth and processor computing power. We recommend a zero-copy method to save the original decoded data:

1. It is recommended to use the extra_frame_buffer_num parameter to increase the number of hardware frame buffers. You can choose the number of buffered frames according to your needs. The disadvantages of this method are that one is occupying the decoder memory, which may reduce the number of video decoding channels; the other is that when the cached frames are not released in time, the video hardware decoding will be blocked when all the buffered frames are used up.

   ```
   av_dict_set_int(&opts, "extra_frame_buffer_num", extra_frame_buffer_num, 0);
   ```

2. It is recommended to use the output_format parameter to set the decoder to output compressed format data, and then use vpp to process the output uncompressed yuv data (when scaling and cropping are required, it can be done synchronously), Then directly zero-copy reference uncompressed yuv data. This method will not affect the hardware decoding performance, and the data space that can be cached is much larger.

   ```
   av_dict_set_int(&opts, "output_format", 101, 0);
   ```

4.51 Warning when opencv VideoCapture decodes video: maybe grab ends normally, retry count = 513

The above problem is because there is a timeout detection in VideoCapture. If a valid packet is not received within a certain period of time, the above log will be output. At this time, if the video source is a network stream, you can use vlc to pull the stream to verify whether the stream is normal. If the file is played to the end, you need to call VidoeCapture.release and then re-VideoCapture.open

4.52 [Problem Analysis]Customer feedback encountered the following error message "VPU_DecRegisterFrameBuffer failed Error code is 0x3" , and then warned Allocate Frame Buffer memory failure.

This Warning message indicates that the allocated number of frames in the decoder buffer exceeds the maximum allowable decoded frames. The cause of this problem may be:

1. For unsupported video encoding formats, such as field format, you can use the method in FAQ14 to record the code stream data and submit it to us for analysis.
2. Extra_frame_buffer_num is set too large. In theory, extra_frame_buffer_num cannot exceed 15, and if it exceeds the maximum number of buffered frames required by the standard, it may not be able to meet the standard. Because most code streams do not use the maximum value, when extra_frame_buffer_num is greater than 15, it can still work for most code streams.

At present, it has been found that there are two possible reasons for this problem, and new cases will continue to be added in the future.

4.53 In SOC mode, opencv reports an error when using 8UC1 Mat, but when the Mat format is 8UC3, the same program works perfectly.

There are many customers who have encountered this problem. This time, a FAQ is specially set up for searching. Its core content has been introduced in FAQ46 "How does mat allocate device memory and system memory in Opencv", you can continue to refer to FAQ46

In soc mode, the 8UC1 Mat created by default does not allocate device memory. Therefore, when hardware acceleration is required, such as inference, bmcv operation, etc., various memory exception errors will be caused.

For the solution, please refer to FAQ26 "How to specify the Mat object to be created and used based on the system memory", specifying that when the 8UC1 Mat is created, the ion allocator is used internally to allocate memory. As follows.

```
cv::Mat gray_mat;
gray_mat.allocator = hal::getAllocator();
gray_mat.create(h, w, CV_8UC1);
```

4.54 When calling the bmcv_image_vpp_convert_padding API, an error of the scaling ratio exceeding 32 times is reported: "vpp not support: scaling ratio greater than 32" .

In the vpp of bm1684, the hardware limit image scaling cannot exceed 32 times (the hardware limit image scaling cannot exceed 128 times in the vpp of bm1684). That should satisfy dst_crop_w <= src_crop_w * 32， src_crop_w <= dst_crop_w * 32， dst_crop_h <= src_crop_h * 32 , src_crop_h <= dst_crop_h * 32。

The cause of this problem may be:

1. Crop_w, crop_h in input crop_rect and dst_crop_w, dst_crop_h in output padding_attr are scaled more than 32 times.
2. The number of crop_rect, padding_attr values should be the same as the number of output_num.

4.55 [Problem Analysis]What is the program warning "VPU_DecGetOutputInfo decode fail framdIdx xxx error(0x00000000) reason(0x00400000), reasonExt(0x00000000)" , the specific value of reason here may be different

This warning is usually caused by a code stream error, the meaning of the prompt is that the xxx frame is decoded incorrectly, and the reason for the error is…. For the upper-layer application, the specific reason here does not need to be concerned, just know that it is caused by the code stream error.

Further analysis, the causes of code stream errors can usually be divided into two categories, and we have to deal with them accordingly. Because once this kind of warning occurs frequently, it means that the decoded data is incorrect. At this time, there may be various mosaics or image blurring, which will cause various abnormal situations for subsequent processing, so we must minimize this situation.

1. Packet loss caused by network conditions. At this time, we can use our test program vidmulti to verify, if there is no decoding error in vidmulti, then this situation can be ruled out. If it is confirmed that the network is lost, it is necessary to distinguish whether the network bandwidth itself is not enough. If the bandwidth itself is not enough, there is no way but to reduce the bit rate of the video stream. If the bandwidth is sufficient, check the network cable. When the number of stream connections exceeds more than 20, it may have exceeded 100M at this time. At this time, the network cable must also be changed to CAT6 to match the Gigabit network.

2. The decoding performance reaches the upper limit, resulting in packet loss. This happens in a streaming environment and doesn't happen for file playback. At this time, we can also use our vidmulti to run it as a comparison. If vidmulti also has an error, it means that the performance has indeed reached the upper limit, otherwise it means that the application itself still has room for optimization.

4.56 [Problem Analysis]The program warns "coreIdx 0 InstIdx 0: VPU interrupt wait timeout", what's going on?

This prompt indicates that the video decoding or encoding interrupt timed out. This prompt is only a warning and will try again, so it can be ignored as long as it doesn't appear in a row. This situation is generally caused by decoding data errors or overloading. For example, in the case of a board, because the data exchange of the board is too frequent, the decoding or encoding data transmission is blocked, and the interrupt times out.

4.57 When using TCP to transmit the stream, if the stream server stops pushing the stream, ffmpeg blocks in av_read_frame

This is because the timeout period is too long. You can use the following parameters to set the timeout period to be shorter.

av_dict_set(&options, "stimeout", "1000000", 0);

4.58 [Problem Analysis]When using ffmpeg jpeg_bm to decode large JPEG images, sometimes it will report "ERROR:DMA buffer size(5242880) is less than input data size(xxxxxxx)", how to solve it?

When decoding JPEG images with FFMPEG's jpeg_bm hardware decoder, the default input buffer is 5120K. Allocate the input buffer before getting the JPEG file to avoid creating and destroying memory frequently when decoding the MJPG file. When the default input buffer size is smaller than the input jpeg file, you can use the following command to increase the input buffer size.

av_dict_set_int(&opts, "bs_buffer_size", 8192, 0); //Note: bs_buffer_size is in Kbyte

4.59 How to fill in csc_type_t, csc_type and csc_matrix_t* matrix when calling the bmcv_image_vpp_basic interface?

When vpp in bmcv does csc color conversion, it provides 4 groups of 601 coefficients and 4 groups of 709 coefficients by default, as shown in csc_type_t.

1. csc_type can be filled with CSC_MAX_ENUM, matrix filled with NULL, and 601 YCbCr and RGB conversion coefficient will be configured by default.
2. csc_type can be filled with the parameters in csc_type_t, such as YCbCr2RGB_BT709, matrix filled with NULL, and the corresponding coefficients will be configured according to the selected type.
3. CSC_type can be filled with CSC_USER_DEFINED_MATRIX, and matrix can be filled with custom coefficients. It will be configured according to the custom coefficient.

The coefficients in csc_matrix_t refer to the following formulas:

Y = csc_coe00 * R + csc_coe01 * G + csc_coe02 * B + csc_add0;

U = csc_coe10 * R + csc_coe11 * G + csc_coe12 * B + csc_add1;

V = csc_coe20 * R + csc_coe21 * G + csc_coe22 * B + csc_add2;

Since 1684 vpp has 10 bits of precision(BM1684x reached FP32), integer handling.

The calculation method of csc_coe and csc_add is: csc_coe = round (float number * 1024) and then take the complement of the integer.

csc_coe takes the lower 13 bit, that is, csc_coe = csc_coe & 0x1fff, csc_add takes the lower 21 bit, that is, csc_add = csc_add & 0x1fffff.

For example:

floating-point coe matrix => fixed-point coe matrix

0.1826 0.6142 0.0620 16.0000 => 0x00bb 0x0275 0x003f 0x004000

4.60 [Problem Analysis]The program crashes when different threads call bm_image_destroy on the same bm_image.

When the bm_image_destroy(bm_image image) interface was designed, the struct was used as the formal parameter, and the memory pointed to by image.image_private was released internally, but the modification of the pointer image.image_private could not be passed to the outside of the function, resulting in a wild pointer problem on the second call.

In order to maximize the compatibility of the client code with the sdk, the interface is not modified at present. It is recommended to set image.image_private = NULL after using bm_image_destroy(image) to avoid wild pointer problems when multi-threaded.

4.61 How to pass Mat information across processes so that different processes can share device memory data in Mat with zero copy?

The obstacle to sharing Mat across processes is that it is very difficult to share virtual memory and handles between processes, so the essence of solving this problem is: how to reconstruct the same Mat data structure from a piece of device memory with zero copy.

To solve this problem, the following three interfaces will be used. The first two interfaces are used to reconstruct the data of yuvMat, and the latter interface is used to reconstruct the data of opencv standard Mat.

```
cv::av::create(int height, int width, int color_format, void *data, long addr, int fd, int* plane_
↪stride, int* plane_size, int color_space = AVCOL_SPC_BT709, int color_range = AVCOL_
↪RANGE_MMPEG, int id = 0)
cv::Mat(AVFrame* frame, int id)
Mat::create(int height, int width, int total, int _type, const size_t* _steps, void* _data,␣
↪unsigned long addr, int fd, int id = 0)
```

```
/* The complete code for sharing Mat across processes is shown below. There are many ways␣
↪to share across processes. The purpose of the following example is to show how to use the␣
↪above functions to reconstruct Mat data. Other codes are for reference only.
where image is the Mat that needs to be shared. */

 union ipc_mat{
  struct{
    unsigned long long addr;
    int total;
    int type;
    size_t step[2];
    int plane_size[4];
    int plane_step[4];
    int pix_fmt;
    int height;
    int width;
    int color_space;
    int color_range;
    int dev_id;
    int isYuvMat;
  }message;
  unsigned char data[128];
}signal;
memset(signal.data, 0, sizeof(signal));

if (isSender){  // The code behind is send
  int fd = open("./ipc_sample", O_WRONLY);
  signal.message.addr = image.u->addr;
  signal.message.height = image.rows;
  signal.message.width = image.cols;
  signal.message.isYuvMat = image.avOK() ? 1 : 0;
```

(continues on next page)

(continued from previous page)

```
    if (signal.message.isYuvMat){  // deal with yuvMat
        signal.message.plane_size[0] = image.avAddr(5) - image.avAddr(4); //avAddr(4~6)␣
↪correspond to the device memory
        signal.message.plane_step[0] = image.avStep(4);

        signal.message.pix_fmt = image.avFormat();
        if (signal.message.pix_fmt == AV_PIX_FMT_YUV420P){
            signal.message.plane_size[2] =
            signal.message.plane_size[1] = image.avAddr(6) - image.avAddr(5);
            signal.message.plane_step[1] = image.avStep(5);
            signal.message.plane_step[2] = image.avStep(6);
        } else if (signal.message.pix_fmt == AV_PIX_FMT_NV12){
            signal.message.plane_size[1] = signal.message.plane_size[0] / 2;
            signal.message.plane_step[1] = image.avStep(5);
        }  // This is for demonstration only, can continue to expand more color formats

        signal.message.color_space = image.u->frame->colorspace;
        signal.message.color_range = image.u->frame->color_range;

        signal.message.dev_id = image.card;
    } else { // deal with bgrMat
        signal.message.total = image.total();
        signal.message.type = image.type();
        signal.message.step[0] = image.step[0];
        signal.message.step[1] = image.step[1];
    }

    write(fd, signal.data, 128);
    /* Here while(1) is for example only, it should be noted that close(fd) is
       required later in practical applications */
    while(1) sleep(1);
} else if (!isSender){
    if ((mkfifo("./ipc_example", 0600) == -1) && errno != EEXIST){  // ipc sharing is for example␣
↪only
        printf("mkfifo failed\n");
        perror("reason");
    }

    int fd = open("./ipc", O_RDONLY);
    Mat f_mat;  // Shared Mat to be reconstructed
    int cnt = 0;

    while (cnt < 128){
        cnt += read(fd, signal.data+cnt, 128-cnt);
    }

    if(signal.message.isYuvMat) { // yuvMat
        AVFrame* f = cv::av::create(signal.message.height,
                                    signal.message.width,
                                    signal.message.pix_fmt,
```

(continues on next page)

(continued from previous page)

```
                    NULL,
                    signal.message.addr,
                    0,
  /* Here, fd can be directly given 0, and its function is only to indicate
    that there is an externally given device memory address addr */
                    signal.message.plane_step,
                    signal.message.plane_size,
                    signal.message.color_space,
                    signal.message.color_range,
                    signal.message.dev_id);
    f_mat.create(f, signal.message.dev_id);
  } else {
    f_mat.create(signal.message.height,
            signal.message.width,
            signal.message.total,
            signal.message.type,
            signal.message.step,
            NULL,
            signal.message.addr,
            0,
  /* Here, fd can be directly given 0, and its function is only to indicate
    that there is an externally given device memory address addr */
            signal.message.dev_id);
    bmcv::downloadMat(f_mat);
  /* Note that the data in the device memory needs to be synchronized to the
    system memory in time. Because the device memory data is always the latest
    in the use of yuvMat, and the system memory data is always the latest in
    the use of bgrMat, which is the design principle we follow in opencv */
  }
  close(fd);
}

  /* The above code is for reference only, please modify and customize according
    to your actual needs. */
```

4.62 [Important Note] When using hardware acceleration decoding in FFMPEG, you need to release all AVFrames and release all AVFrames from av_frame_alloc() before the current channel AVCodecContext is released.

The above AVFrames all come from avcodec_receive_frame, avcodec_decode_video2.

4.63 Failed to apply for device memory, the error returns -24.

There will be an fd for each application of device memory, and the maximum is 1024 on ubuntu. If the application continues and is not released, and the number of fds exceeds 1024, the application for device memory will fail, and an error of -24 will be returned. If you want to expand the number of fds in ubuntu, you can modify the limit through the ulimit command. For example, ulimit -n 10000 can expand the number of fds in ubuntu to 10000.

4.64 Questions about bm_image_create, bm_image_alloc_dev_mem, bm_image_attach.

1. bm_image_create: used to create the bm_image structure.
2. bm_image_alloc_dev_mem: apply for device memory, and bm_image will be attached internally.
3. bm_image_attach: used to bind the device memory obtained from opencv, etc. with the bm_image applied by bm_image_create.

4.65 Questions about bm_image_destroy and bm_image_detach.

1. bm_image_detach: used to unbind the device memory associated with bm_image. If the device memory is automatically applied for internally, the device memory will be released; if bm_image is not bound to the device memory, it will return directly.

2. bm_image_destroy: bm_image_detach is called nested inside this function. That is to say, when this function is called, if the device memory bound by bm_image is applied for through bm_image_alloc_dev_mem, it will be released; If it is the device memory bound by bm_image_attach, it will not be released. At this time, it is necessary to pay attention to whether there is a memory leak in the device memory. If there are other modules that continue to use it, the corresponding module will release it.

3. In general, whoever applies for the device memory will release it. If the device memory bound by attach is used, bm_image_destroy cannot be called to release it. If it is determined that the device memory bound by attach is no longer used, it can be released through bm_free_device interface to release.

4.66 Is it necessary to call bm_create_image before bmcv::toBMI, and if so, will using bm_image_destroy at the end cause memory leaks?

1. bmcv::toBMI calls bm_image_create internally, no need to call bm_create_image again.
2. If bm_create_image is called before bmcv::toBMI, it will cause a memory leak.
3. After calling bmcv::toBMI, in addition to calling bm_image_destroy, you also need image.image_private = NULL.

4.67 Use ffmpeg to encode, decode and filter in pcie mode. Be sure to specify sophon_idx.

1. The default sophon_idx for decoding, encoding and filtering is 0.
2. In HWAccel mode: specify the card ID through hwaccel_device. eg:

```
ffmpeg -hwaccel bmcodec -hwaccel_device 1 -c:v hevc_bm -output_format 0
    -i video.265 -vf "scale_bm=352:288:sophon_idx=1"
    -c:v h264_bm -g 256 -b:v 256K -sophon_idx 1 -y out.264
```

3. In normal mode: specify the card id through sophon_idx. eg:

```
ffmpeg -c:v hevc_bm  -sophon_idx 1 -i video.265
    -vf "scale_bm=2560:1440:opt=crop:zero_copy=1:sophon_idx=1"
    -c:v h264_bm -g 256 -b:v 8M -sophon_idx 1 -y out.264
```