



RK3399Pro入门教程 (6) 硬件编解码器MPP库的使用

发表于 2019-4-16 11:57:37 查看: 265460 | 回复: 202 | [复制链接] | 只看该作者

楼主

本帖最后由 jefferyzhang 于 2022-12-9 08:54 编辑



jefferyzhang

版主

积分 11966

MPP库安装方式

1. dnf 安装(详见wiki)

```
1. sudo dnf install librockchip_mpp-devel
复制代码
```

2. 源码编译

MPP库源码下载地址

```
1. https://github.com/rockchip-linux/mpp
2. 或 https://github.com/HermaChen/mpp
复制代码
```

MPP库简介

MPP库是Rockchip根据自己的硬编解码器开发的应用程序编解码库, 如果想达到最好的效果, 必须要通过librockchip_mpp来直接编码实现编解码。其他第三方库都会因为兼容api的原因, 使增加几次无用的跳转动作, 并且使用的都是虚拟地址。在上一篇RGAA的教学中, 我们知道纯物理连续地址的硬件操作是非常快的, 转到虚拟地址后效率就会降低。如果选择Toybrick的性能, 开发最完美的代码, 纯连续的物理Buffer, mpp+rga是离不开的。

Mpp的API思路其实跟目前绝大多数的编解码库是一致的, 都是queue/dequeue的队列操作方式, 先设置好编解码状态, 然后不停的queue/dequeue input/output buffer就可以实现编解码控制了。

Mpp库自带的test基本可以带大家入手。

MPP编译

1. (没有交叉编译环境的建议还是直接放在板子上编译) cd 到build目录里对应平台的目录

```
1. cd build/linux/aarch64
复制代码
```

2. 如果是交叉编译环境, 需要修改该目录下编译链的配置。然后执行编译脚本。

```
1. ./make-Makefiles.bash
复制代码
```

MPP库函数讲解

解码范例在mpp源码内: test/mpi_dec_test.c

编码范例在mpp源码内: test/mpi_enc_test.c

1. 创建 MPP context 和 MPP api 接口。(注意, 和RGA一样, 多个线程多个实例需要多个独立的context)

```
1. ret = mpp_create(&ctx, &mpi);
2.
3. if (MPP_OK != ret) {
4.     mpp_err("mpp_create failed\n");
5.     goto MPP_TEST_OUT;
6. }
复制代码
```

2. 设置一些MPP的模式(这里设置的是 MPP_DEC_SET_PARSER_SPLIT_MODE)

```
1. mpi_cmd = MPP_DEC_SET_PARSER_SPLIT_MODE;
2. param = &need_split;
3. ret = mpi->control(ctx, mpi_cmd, param);
4. if (MPP_OK != ret) {
5.     mpp_err("mpi->control failed\n");
6.     goto MPP_TEST_OUT;
7. }
复制代码
```

常用设置的一些模式解释如下:(其余的可以看MPP自带的开发文档, 在doc目录下有详细说明)

MPP_DEC_SET_PARSER_SPLIT_MODE: (仅限解码)

自动拼包(建议开启), 硬编解码器每次解码就是一个Frame, 所以如果输入的数据不确定是不是一个Frame(例如可能是一个Slice, 一个Nalu或者一个FU-A分包, 甚至可能随意读的任意长度数据), 那就必须把该模式打开, MPP会自动分包拼包成一个完整Frame送给硬编解码器。

MPP_DEC_SET_IMMEDIATE_OUT: (仅限解码)

立即输出模式(不建议开启), 如果未开启立即输出模式, MPP会按预先设定的节奏间隔输出解码的帧(例如33ms输出一帧), 但是实际硬件解码过程并不是均匀输出的, 有时候两帧间隔可能就1ms一下子输出2-3帧, 有时候两帧间又会有较长的间隔。如果打开立即输出模式, MPP就会在解码成功后立即输出一帧, 那后续处理显示的节奏就需要用户自己控制。该模式适用于一些对实时性要求比较高的客户产品, 需要自己把握输出节奏。

MPP_SET_INPUT_BLOCK:

MPP_SET_INPUT_BLOCK_TIMEOUT:

MPP_SET_OUTPUT_BLOCK:

MPP_SET_OUTPUT_BLOCK_TIMEOUT:

设置输入输出的block模式, 如果block模式打开, 喂数据时候会block住直到编解码成功入队列或者出队列或者达到TIMEOUT时间, 才会返回。

3. 初始化 MPP

```
1. ret = mpp_init(ctx, MPP_CTX_DEC, MppCodingType::MPP_VIDEO_CodingAVC);
2. if (MPP_OK != ret) {
3.     mpp_err("mpp_init failed\n");
4.     goto MPP_TEST_OUT;
5. }
复制代码
```

初始化编码还是解码, 以及编解码的格式。

MPP_CTX_DEC: 解码

MPP_CTX_ENC: 编码

MPP_VIDEO_CodingAVC: H.264

MPP_VIDEO_CodingHEVC: H.265

MPP_VIDEO_CodingVP8: VP8

MPP_VIDEO_CodingVP9: VP9

MPP_VIDEO_CodingMJPEG: MJPEG

等等, 详细参看rk_mpi.h定义

4. 解码的话到这里初始化就完成了, 编码的话需要多设置一些参数

设置编码宽高、对齐宽高参数

```
1. mPrepCfg.change = MPP_ENC_PREP_CFG_CHANGE_INPUT | MPP_ENC_PREP_CFG_CHANGE_FORMAT;
2. mPrepCfg.width = mWidth;
3. mPrepCfg.height = mHeight;
4. mPrepCfg.hor_stride = mHStride;
5. mPrepCfg.ver_stride = mVStride;
6. mPrepCfg.format = mFrameFormat;
7.
8. int ret = mAppApi->control(mAppCtx, MPP_ENC_SET_PREP_CFG, &mPrepCfg);
复制代码
```

设置编码码率、质量、定码率变码率

```
1. mRcCfg.change = MPP_ENC_RC_CFG_CHANGE_ALL;
2.
3. /*
4.  * rc_mode = rate control mode
5.  */
复制代码
```

```

5.  * Mpp balances quality and bit rate by the mode index
6.  * Mpp provide 5 level of balance mode of quality and bit rate
7.  * 1 - only quality mode: only quality parameter takes effect
8.  * 2 - more quality mode: quality parameter takes more effect
9.  * 3 - balance mode      : balance quality and bitrate 50 to 50
10. * 4 - more bitrate mode: bitrate parameter takes more effect
11. * 5 - only bitrate mode: only bitrate parameter takes effect
12. */
13. if (mIsCBR) {
14.     mRcCfg.rc_mode = (MppEncRcMode) MPP_ENC_RC_MODE_CBR;
15. } else {
16.     mRcCfg.rc_mode = (MppEncRcMode) MPP_ENC_RC_MODE_VBR;
17. }
18.
19. /*
20.  * quality - quality parameter
21.  * mpp does not give the direct parameter in different protocol.
22.  * mpp provide total 5 quality level 1 ~ 5
23.  * 0 - auto
24.  * 1 - worst
25.  * 2 - worse
26.  * 3 - medium
27.  * 4 - better
28.  * 5 - best
29.  */
30. if (mQuality > 4) {
31.     mRcCfg.quality = (MppEncRcQuality)MPP_ENC_RC_QUALITY_BEST;
32. } else {
33.     mRcCfg.quality = (MppEncRcQuality)mQuality;
34. }
35.
36.
37. int bps = mBps;
38. switch (mRcCfg.rc_mode) {
39.     case MPP_ENC_RC_MODE_CBR:
40.         // constant bitrate has very small bps range of 1/16 bps
41.         mRcCfg.bps_target = bps;
42.         mRcCfg.bps_max = bps * 17 / 16;
43.         mRcCfg.bps_min = bps * 15 / 16;
44.         break;
45.     case MPP_ENC_RC_MODE_VBR:
46.         // variable bitrate has large bps range
47.         mRcCfg.bps_target = bps;
48.         mRcCfg.bps_max = bps * 3 / 2;
49.         mRcCfg.bps_min = bps * 1 / 2;
50.         break;
51.     default:
52.         abort();
53. }
54.
55. /* fix input / output frame rate */
56. mRcCfg.fps_in_flex = 0;
57. mRcCfg.fps_in_num = mFps;
58. mRcCfg.fps_in_denorm = 1;
59. mRcCfg.fps_out_flex = 0;
60. mRcCfg.fps_out_num = mFps;
61. mRcCfg.fps_out_denorm = 1;
62.
63. mRcCfg.gop = mIInterval; /* i frame interval */
64. mRcCfg.skip_cnt = 0;
65.
66. int ret = mMppApi->control(mMppCtx, MPP_ENC_SET_RC_CFG, &mRcCfg);

```

复制代码

设置264相关的其他编码参数

```

1.  mCodecCfg.h264.change = MPP_ENC_H264_CFG_CHANGE_PROFILE |
2.      MPP_ENC_H264_CFG_CHANGE_ENTROPY |
3.      MPP_ENC_H264_CFG_CHANGE_TRANS_8x8 |
4.      MPP_ENC_H264_CFG_CHANGE_QP_LIMIT;
5.
6.  /*
7.   * H.264 profile_idc parameter
8.   * 66 - Baseline profile
9.   * 77 - Main profile
10.  * 100 - High profile
11.  */
12.  mCodecCfg.h264.profile = 100;
13.
14.  /*
15.   * H.264 level_idc parameter
16.   * 10 / 11 / 12 / 13 - cif@15fps / cif@7.5fps / cif@15fps / cif@30fps
17.   * 20 / 21 / 22 - cif@30fps / half-D1@25fps / D1@12.5fps
18.   * 30 / 31 / 32 - D1@25fps / 720p@30fps / 720p@60fps
19.   * 40 / 41 / 42 - 1080p@30fps / 1080p@30fps / 1080p@60fps
20.   * 50 / 51 / 52 - 4K@30fps
21.  */
22.  mCodecCfg.h264.level = 40;
23.  mCodecCfg.h264.entropy_coding_mode = 1;
24.  mCodecCfg.h264.cabac_init_idc = 0;
25.  mCodecCfg.h264.transform8x8_mode = 1;
26.
27.  if (mRcCfg.rc_mode == MPP_ENC_RC_MODE_CBR) {
28.      mCodecCfg.h264.qp_init = 10;
29.      mCodecCfg.h264.qp_min = 4;
30.      mCodecCfg.h264.qp_max = 30;
31.      mCodecCfg.h264.qp_max_step = 16;
32.  }
33.
34.  int ret = mMppApi->control(mMppCtx, MPP_ENC_SET_CODEC_CFG, &mCodecCfg);

```

复制代码


5. 接下来就是喂数据和拿输出数据的过程了。具体可以直接看sample代码。这里解释下一些基本概念。方便大家看Sample代码时不借温。


MppPacket : 存放编码数据。例如264、265数据
MppFrame : 存放解码的数据。例如YUV、RGB数据
MppTask : 一次编码或者解码的session

编码就是喂MppFrame, 输出MppPacket;
解码就是喂MppPacket, 输出MppFrame;

MPV包含两套接口做编解码:
一套是简易接口。类似 decode_put_packet / decode_get_frame 这样put/get即可
一套是高级接口。类似 poll / enqueue/ dequeue 这样的对input output队列进行操作

解码得到的output buffer一般都拥有虚拟地址和物理地址的fd。紧接着就可以通过RGA做对应操作或者拷贝。速度是相当快的。

**本帖子中包含更多资源**
您需要 [登录](#) 才可以下载或查看, 没有帐号? [立即注册](#)



★ 收藏 5

回复

举报



ronyuzhang

注册会员

积分 73

发表于 2019-4-17 10:08:59 | 只看该作者

沙发

大赞的教程, 良心, 有求必应@jefferyzhang 希望再接再厉, 嘉惠码农。

回复

举报



yaowei

中级会员

积分 375

发表于 2019-4-18 11:44:52 | 只看该作者

板凳

那么视频解码必须要用C++写的? python代码是没有的吗?

回复

举报



jefferyzhang

版主

积分 11966

楼主 | 发表于 2019-4-18 12:28:50 | 只看该作者

地板

“ yaowei 发表于 2019-4-18 11:44
那么视频解码必须要用C++写的? python代码是没有的吗? ”

python可以使用gstreamer-rockchip来转接。例如opencv内部调用的是gstreamer的接口, 是可以调用到硬解码的。具体可以问下其他开发者, 他们有成功用起来过

回复

举报



yaowei

中级会员

积分 375

发表于 2019-4-18 17:17:58 | 只看该作者

5#

我也成功用起来了, 不需要gstreamer-rockchip, 还需要其他一些库, 现在可以解码视频和rtsp摄像头。

回复

举报



ronyuzhang

注册会员

积分 73

发表于 2019-4-30 18:42:30 | 只看该作者

6#

官方mipi的camera 什么时候发售

回复

举报



jefferyzhang

版主

积分 11966

楼主 | 发表于 2019-5-1 19:12:37 | 只看该作者

7#

“ ronyuzhang 发表于 2019-4-30 18:42
官方mipi的camera 什么时候发售 ”

快了...

回复

举报



kiwi

中级会员

积分 418

QQ交谈

发表于 2019-5-8 18:18:57 | 只看该作者

8#

MPP兼容的ffmpeg, 是指用这个ffmpeg去编解码会调用到vpu吗

回复

举报



jefferyzhang

版主

积分 11966

楼主 | 发表于 2019-5-9 08:50:07 | 只看该作者

9#

本帖最后由 jefferyzhang 于 2022-7-21 20:54 编辑

“ kiwi 发表于 2019-5-8 18:18
MPP兼容的ffmpeg, 是指用这个ffmpeg去编解码会调用到vpu吗 ”

(已删除)

回复

举报

发表于 2019-5-16 11:21:49 | 只看该作者

10#



积分 16

下一页 ▶

高级模式

Toybrick