# 寒武纪 FFmpeg-MLU 用户手册

**版本 *3.1.0***

**2023 年 01 月 03 日**

Cambricon 寒武纪

# 目录



 

 

# 表格目录



 

# 1 版权声明

**免责声明**

中科寒武纪科技股份有限公司（下称“寒武纪”）不代表、担保（明示、暗示或法定的）或保证本文件所含信息，并明示放弃对可销售性、所有权、不侵犯知识产权或特定目的适用性做出任何和所有暗示担保，且寒武纪不承担因应用或使用任何产品或服务而产生的任何责任。寒武纪不应对因下列原因产生的任何违约、损害赔偿、成本或问题承担任何责任：（1）使用寒武纪产品的任何方式违背本指南；或（2）客户产品设计。

**责任限制**

在任何情况下，寒武纪都不对因使用或无法使用本指南而导致的任何损害（包括但不限于利润损失、业务中断和信息损失等损害）承担责任，即便寒武纪已被告知可能遭受该等损害。尽管客户可能因任何理由遭受任何损害，根据寒武纪的产品销售条款与条件，寒武纪为本指南所述产品对客户承担的总共和累计责任应受到限制。

**信息准确性**

本文件提供的信息属于寒武纪所有，且寒武纪保留不经通知随时对本文件信息或对任何产品和服务做出任何更改的权利。本指南所含信息和本指南所引用寒武纪文档的所有其他信息均“按原样”提供。寒武纪不担保信息、文本、图案、链接或本指南内所含其他项目的准确性或完整性。寒武纪可不经通知随时对本指南或本指南所述产品做出更改，但不承诺更新本指南。

本指南列出的性能测试和等级要使用特定芯片或计算机系统或组件来测量。经该等测试，本指南所示结果反映了寒武纪产品的大概性能。系统硬件或软件设计或配置的任何不同会影响实际性能。如上所述，寒武纪不代表、担保或保证本指南所述产品将适用于任何特定用途。寒武纪不代表或担保测试每种产品的所有参数。客户全权承担确保产品适合并适用于客户计划的应用以及对应用程序进行必要测试的责任，以期避免应用程序或产品的默认情况。

客户产品设计的脆弱性会影响寒武纪产品的质量和可靠性并导致超出本指南范围的额外或不同的情况和/或要求。

**知识产权通知**

寒武纪和寒武纪的标志是中科寒武纪科技股份有限公司在中国和其他国家的商标和/或注册商标。其他公司和产品名称应为与其关联的各自公司的商标。

式复制、重制、修改、出版、上传、发布、传输或分发本指南。除了客户使用本指南信息和产品的权利，根据本指南，寒武纪不授予其他任何明示或暗示的权利或许可。未免疑义，寒武纪不根据任何专利、版权、商标、商业秘密或任何其他寒武纪的知识产权或所有权对客户授予任何（明示或暗示的）权利或许可。

- 版权声明
- © 2023 中科寒武纪科技股份有限公司保留一切权利。

# 2 前言

## 2.1 版本记录

表 2.1: 版本记录

| | |
|---|---|
| **文档名称** | 寒武纪 FFmpeg-MLU 用户手册 |
| **版本号** | V3.1.0 |
| **作者** | Cambricon |
| **修改日期** | 2023.01.03 |

## 2.2 更新历史

- V3.1.0

  **更新时间**：2023 年 01 月 03 日

  **更新内容**：

  – 增加 MLU370 及 MLU370+ 视频解码器设置解码序输出功能。

  – 优化了 MLU370 及 MLU370+ 视频解码器延时。

  – 完善 **FFmpeg - MLU 编译及运行**章节

  – 完善 **FAQ** 章节。

- V3.0.0

  **更新时间**：2022 年 09 月 28 日

  **更新内容**：

  – 完善关于 MLU300 的描述。

  – 完善 FAQ 章节。

- V2.4.0

  **更新时间**：2022 年 08 月 18 日

  **更新内容**：

  – 修改关于 MLU 的描述。

 

  - 增加 avs2 解码器。
  - 完善 FAQ 章节。
- V2.3.0

  **更新时间**：2022 年 04 月 28 日

  **更新内容**：

  - 将 FFmpeg-MLU 多个参考文档整合为一个文档。

服务联系: service@cambricon.com

 

# 3 FFmpeg-MLU 简介

FFmpeg 作为迄今最为流行的开源多媒体操作工具之一，提供了完整的录制、转换以及流化音视频的解决方案。因其具备丰富的音视频插件库和高度的可移植特性，以及多个音视频插件在同一 pipeline 框架上挂载从而构成完整的多媒体系统的可实现性，使其在世界范围内得到广泛的应用。

寒武纪 AI 加速卡上内置了视频、图像相关的硬件加速计算单元。为了利用硬件提高计算效率，同时保障产品的可用性和用户使用的便捷性，寒武纪提供了 FFmpeg-MLU SDK 软件解决方案。FFmpeg-MLU 集成了寒武纪硬件加速卡的视频、图像硬件编解码单元和硬件 AI 计算单元，实现了基于 Cambricon MLU 硬件加速的视频编码、解码和 AI 计算；其中硬件视频图像编解码单元基于寒武纪 CNCodec 加速库开发。依靠 FFmpeg 音视频编解码和流媒体协议等模块，Cambricon 视频、图像编解码单元及 AI 加速单元可以很便捷地实现高性能硬件加速的多媒体处理 pipeline。

寒武纪 FFmpeg-MLU SDK 使用纯 C 接口实现硬件加速的图像、视频编解码功能和常见图像算法处理，完全兼容社区 FFmpeg；符合社区 FFmpeg 代码开发及命令行使用规范，同时也符合社区 FFmpeg hwaccel 硬件加速框架规范 (https://trac.ffmpeg.org/wiki/HWAccelIntro)，实现了硬件内存管理、硬件加速处理模块与 CPU 模块的流程化兼容处理等。

 

# 4 FFmpeg-MLU 编译及运行

## 4.1 前提条件

编译及运行 FFmpeg-MLU 之前，需要确保以下环境和依赖已正确安装：

- 支持的操作系统如下：
  - Ubuntu 16.04/18.04
  - Centos 7/8
  - Kylin(详细请查看寒武纪软件栈支持的系统版本)
- 支持的 MLU 硬件形态：- MLU200：MLU220-m.2、MLU220-SOM、MLU220-edge、MLU270-all。
  - MLU300：MLU370-all 和 MLU370+。
- 寒武纪 MLU 驱动：
  - MLU200 平台：
    - Driver-4.7.0 或更高版本。具体安装方法，请参考《寒武纪驱动安装包使用手册》。
  - MLU370 平台：
    - Driver-4.15.12 或更高版本。具体安装方法，请参考《寒武纪驱动安装包使用手册》。
- 寒武纪 MLU SDK：
  - MLU200 平台：
    - cntookit-1.5.0 或更高版本。具体安装方法，请参考《寒武纪 CNToolkit 安装升级使用手册》。
    - cncv-0.4.0 或更高版本。具体安装方法，请参考《寒武纪计算机视觉库用户手册》。
  - MLU370 平台：
    - cntookit-2.2.0 或更高版本 (cntookit-3.0.1 除外)。具体安装方法，请参考《寒武纪 CNToolkit 安装升级使用手册》。
    - cncv-0.7.0 或更高版本。具体安装方法，请参考《寒武纪计算机视觉库用户手册》。

 

## 4.2 下载 FFmpeg-MLU 源代码及编译

```
git clone https://github.com/Cambricon/ffmpeg-mlu
cd ffmpeg-mlu

# 编译 MLU200 系列硬件平台版本
## 若使用系统 Bangware 环境
source compile_ffmpeg.sh MLU200
## 若使用指定路径 BangWare 环境
source compile_ffmpeg.sh MLU200 ${BANGWARE_HOME}

# 编译 MLU370 硬件平台版本
## 若使用系统 BangWare 环境
source compile_ffmpeg.sh MLU370
## 若使用指定路径 BangWare 环境
source compile_ffmpeg.sh MLU370 ${BANGWARE_HOME}
```

**注意：**

- ${BANGWARE_HOME} 表示 Cambricon cntoolkit SDK 软件包安装的位置。
- 请根据不同的硬件平台选择对应的平台名字。此处 MLU270 和 MLU220 统称为 MLU200，MLU370 各种硬件平台统称为 MLU370。
- 若 MLU220-SOM、MLU220 Edge 及 MLU370 等平台需要运行在 arm 平台上，则需要进行交叉编译，具体交叉编译方法请参考 Cambricon 社区教程链接: https://forum.cambricon.com/index.php?m=content&c=index&a=show&catid=47&id=1617 。
- 若仅需要 MLU 编解码器功能，则直接使用 compile_ffmpeg.sh 脚本即可，默认仅使用 --enable-mlu 和 --enable-mlumpp 编译选项；可参考编译指令：../configure ... **--enable-ffmpeg --enable-mlu --enable-mlumpp** ...。
- 若还需要使用 mlu-filter 功能，则需要在 compile_ffmpeg.sh 脚本中增加 --enable-mlufilter 编译选项；可参考编译指令：../configure ... **--enable-mlu --enable-mlumpp --enable-mlufilter** ...。

## 4.3 设置 LD_LIBRARY_PATH 路径

运行下面命令，设置 LD_LIBRARY_PATH 路径：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${BANGWARE_HOME}/lib64
```

 

## 4.4 (可选) 编译 mlu_op

如果想要通过 MLU 加速实现视频缩放及颜色空间转换等功能，需先安装 **MLU-CNCV 硬件加速库** (版本可参考前提条件)，然后编译 mlu_op 并拷贝 libeasyOP.so 到 ${BANGWARE_HOME}/lib64 目录下。

```
#1: 安装完成 cncv 库
...
#2: 编译 mlu_op 算子
cd mlu_op & mkdir build
cd build & cmake .. & make -j
#3: 拷贝 libeasyOP.so 到 BANGWARE_HOME 路径下
cp ../lib/libeasyOP.so ${BANGWARE_HOME}/lib64
```

**注意：**

- 若要使用 mlu-filter 功能，则需要在 compile_ffmpeg.sh``脚本中增加``--enable-mlufilter``编译选项，默认仅使用--enable-mlu “和 “--enable-mlumpp “编译选项。

## 4.5 运行 FFmpeg-MLU

可以通过以下任意一种方式运行 FFmpe-MLU：

- 通过动态库或者静态库在基于社区通用 API 开发的工程中使用 FFmpeg-MLU。
- 直接通过社区通用命令行使用 FFmpeg-MLU。

下边分别通过两种方式进行运行演示。

### 4.5.1 API 示例运行

通过调用 API 方式来使用 MLU 硬件功能，具体请参考 ffmpeg-mlu/doc/examples 文件夹下 decode_mlu、encode_mlu 和 hw_decode_mlu 三个示例。

**示例 1:** 该示例基于社区 FFmpeg 通用 API 开发；读取 input.mp4 的 H.264 视频流文件，解码输出 out.yuv 的输出文件，参数 dev_id 表示设备号。sample 位于 ffmpeg-mlu/doc/examples/decode_mlu.c 路径下。

```
./build/doc/examples/decode_mlu input.mp4 out.yuv dev_id
# 参数为：输入文件 输出文件 设备号
```

**示例 2:** 该示例基于社区 FFmpeg hwaccel 通用硬件加速 API 开发；读取 input.mp4 的 H.264 视频流文件，解码输出 out.yuv 的输出文件，参数 dev_id 表示设备号。sample 位于

 

ffmpeg-mlu/doc/examples/hw_decode_mlu.c 路径下。

```
./build/doc/examples/hw_decode_mlu input.mp4 out.yuv dev_id
# 参数为：输入文件 输出文件 设备号
```

**示例 3:** 该示例基于社区 FFmpeg 通用 API 开发；自动生成 raw 数据送给编码器，编码器编完的码流输出到 out.h264 文件中，参数 dev_id 表示设备号。sample 位于 ffmpeg-mlu/doc/examples/encode_mlu.c 路径下。

```
./build/doc/examples/encode_mlu out.h264 dev_id enc_type
# 参数为：输出文件 设备号 编码类型 (h264 或者 hevc)
```

### 4.5.2 命令行示例

通过调用命令行的方式来使用 MLU 硬件功能，可参考如下示例及FFmpeg-MLU 硬件模块 章节中的说明来使用。也可以结合社区 FFmpeg 进行更高级的用法，具体可参考社区 FFmpeg。

**示例 1:** MLU 硬件解码运行示例。读取包含 H.264 码流的 input.mp4 文件，将文件解码并存储到 out.yuv 文件。

```
ffmpeg -vsync 0 -c:v h264_mludec -i input.mp4 output.yuv
```

**示例 2:** MLU 硬件解码运行示例。读取包含 H.264 码流的 input.mp4 文件，通过 MLU 解码，输出到 MLU 内存中，并通过 hwdownload_mlu mlu-filter 将解码结果存储到 out.yuv 文件。

```
ffmpeg -vsync 0 -hwaccel mlu -hwaccel_output_format mlu -hwaccel_device 0 -c:v h264_mludec -i␣
↪input.mp4 -vf hwdownload_mlu out.yuv
```

**示例 3:** MLU 硬件编码运行示例。读取分辨率为 1920x1080 像素格式为 nv12 的 input.yuv 文件，将文件编码为 H.264 码流，并存储为 output.h264 视频流文件。

```
ffmpeg -vsync 0 -s 1920x1080 -pix_fmt nv12 -i input.yuv -c:v h264_mluenc output.h264
```

 

# 5 FFmpeg-MLU 硬件模块

## 5.1 硬件视频、图像解码器

### 5.1.1 FFmpeg-MLU 支持的硬件视频及图像解码格式

- H.264/AVC Codec 名称：`h264_mludec`
- HEVC Codec 名称：`hevc_mludec`
- VP8 Codec 名称：`vp8_mludec`
- VP9 Codec 名称：`vp9_mludec`
- JPEG/MJEPG Codec 名称：`mjpeg_mludec`
- AVS2 Codec 名称：`avs2_mludec`

具体规格细节，请查看寒武纪《寒武纪 CNCodec-V3 用户手册》和《寒武纪 CNCodec-V3 开发者》。

### 5.1.2 硬件图像、视频解码器支持的配置选项

FFmpeg-MLU 的硬件图像、视频解码器提供了专属的配置选项，可通过社区 FFmpeg 通用指令查询：

```
# 查询支持的 MLU 硬件解码器类型
ffmpeg -decoders | grep mlu
# 查询专属配置选项
ffmpeg -h decoder=h264_mludec
ffmpeg -h decoder=hevc_mludec
...
```

在基于 API 开发的工程中，上述专属选项可以通过 `av_dict_set` 或 `av_dict_set_int` API 设置；在命令行调用中，上述专属选项可以通过在解码器后作为参数设置；具体使用可参考社区 FFmpeg(https://ffmpeg.org/documentation.html)。

图像解码器仅支持 jpeg 格式的图像或 mjpeg 格式的视频。解码专属选项具体介绍如下：

 

表 5.1: MLU200 硬件平台解码器参数

| 参数 | 类型 | 描述 |
| --- | --- | --- |
| device_id | int | 选择使用的加速卡，设置多卡时支持设置范围为：**0** - **MAX_DEVICE_COUNT**；其中 **MAX_DEVICE_COUNT** 值为加速卡总数减 1，默认值为 **0**。 |
| instance_id | int | 解码使用的 VPU/JPU 硬件编号，支持设置范围为：**-1** - **MAX_INSTANCE_COUNT**。其中 **MAX_INSTANCE_COUNT** 值为 VPU/JPU 硬件总数减 1，**-1** 表示自动选择，默认值为 **-1**。 |
| input_buf_num | int | 输入缓冲器的数量。支持设置范围为：**1** - **18**，默认值为 **4**。 |
| output_buf_num | int | 输出缓冲器的数量；支持设置范围为：**1** - **18**，默认值为 **4**。 |
| stride_align | int | 输出对齐的步长；支持设置范围为：**1/2/4/8/16/32/64/128**，默认值为 **1**。 |
| output_pixfmt | int | 输出的像素格式，可设置的值为: **nv12/nv21/p010/yuv420p**，解码器默认设置 **nv12**。 |
| resize | string | 只支持 **视频解码器**设置，调整视频大小 **(width)x(height)**；使用该功能，需确保寒武纪 cncv 加速库安装成功，同时需确保 libeasyOP.so 置于指定目录下 (可参考(可选) 编译 mlu_op)。默认不打开。 |
| post_pixfmt | string | 只支持 **视频解码器**设置，输出格式为 rgbx 格式，使用该参数时建议不设置 outout_pixfmt 参数; 支持设置的格式为：**rgb24、bgr24、rgba、bgra、argb、abgr**；仅在 output_pixfmt 设置为 **nv12/nv21** 时生效，默认不打开；**推荐使用默认值**。 |
| frame_extract | int | 只支持 **视频解码器**设置，输出解码 frame 的 IBP 帧类型；支持的值为 **0 或 1**，仅在设置为 1 时生效，默认设置为 0；**推荐使用默认值**。 |

表 5.2: MLU370 硬件平台解码器参数

| 参数 | 类型 | 描述 |
| --- | --- | --- |

下页继续

 

表 5.2 – 续上页

| | | |
|---|---|---|
| device_id | int | 选择使用的加速卡, 设置多卡时使用; 支持设置范围为: **0** - **MAX_DEVICE_COUNT** 。其中 **MAX_DEVICE_COUNT** 值为加速卡总数减 1, 默认值为 **0** 。 |
| output_buf_num | int | 解码器输出缓冲器的数量；支持设置范围为 **1** - **18** ，默认值为 **4** 。 |
| stride_align | int | 输出对齐的步长；支持设置范围为 **1/2/4/8/16/32/64/128** ，默认值为 **1** 。 |
| output_pixfmt | int | 输出像素的格式。支持设置的值为: **nv12/nv21/p010/yuv420p** , 默认值为 **nv12** 。 |
| crop | string | 对解码输出图像进行 crop 功能，默认关闭。输入格式为 **(left)x(top)x(width)x(height)** ，视频解码器支持 **48x48** - **8192x4320** 分辨率 crop，但 crop 的图不能超过原始图像边界。 |
| resize | string | 对解码输出图像进行 resize 功能，输出格式为 **(width)x(height)** ，默认关闭。**视频解码器**支持原始分辨率缩至 **48x48** 。宽高只能同时缩小, 且需要 2 像素对齐。而 **图像解码器**仅支持相对于原图 1/2、1/4 和 1/8 的缩放。 |
| backend | int | 图像解码参数，选择不同硬件 backend 模式进行图像解码，支持设置的值为：0,1,2。 |
| output_order | int | 视频解码参数，设置视频解码器输出顺序支持设置的值为：display 和 decode，默认为 display |

## 5.2 硬件视频编码器

### 5.2.1 FFmpeg-MLU 支持的硬件视频编码格式

- H.264/AVC Codec 名称：`h264_mluenc`
- HEVC Codec 名称：`hevc_mluenc`

 

### 5.2.2 硬件视频编码器支持的配置选项

FFmpeg-MLU 的硬件视频编码器提供了专属的配置选项，可通过社区通用指令查询：

```
# 查询支持的 MLU 硬件编码器类型
ffmpeg -encoders | grep mlu
# 查询专属配置选项
ffmpeg -h encoder=h264_mluenc
ffmpeg -h encoder=hevc_mluenc
```

在基于 API 开发的工程中，上述专属选项可以通过 `av_dict_set` 或 `av_dict_set_int` API 设置；在命令行调用中，上述专属选项可以通过在编码器后作为参数设置。**其他通用选项**除以下表格所列参数外，也支持社区 FFmpeg 的设置，如: `-b` , `-bf` , `-g` , `-qmin` , `-qmax` 等等，具体作用以及数值范围请参考社区 FFmpeg 官方文档，详见社区 FFmpeg(https://ffmpeg.org/documentation.html) 。在对视频编码器专属选项详细介绍如下：

表 5.3: MLU200 硬件平台视频编码器参数

| 参数 | 类型 | 描述 |
|---|---|---|
| device_id | int | 选择使用的加速卡，设置多卡时使用；支持设置的范围：**0** - **MAX_DEVICE_COUNT** ；**MAX_DEVICE_COUNT** 值为加速卡总数减 1，默认值为 **0** 。 |
| instance_id | int | 选择使用的 VPU 硬件编号。支持设置范围为：**-1** - **MAX_INSTANCE_COUNT** 。其中 **MAX_INSTANCE_COUNT** 值为 VPU 硬件总数减 1。**-1** 表示自动选择，默认值为 **-1** 。 |
| input_buf_num | int | 编码输入缓冲的数量。支持设置的值的范围为：**1** - **18** 。默认值为 **4** 。 |
| output_buf_num | int | 编码输出缓冲的数量支持设置的值的范围为：**1** - **18** ; 默认值为 **4** 。 |
| init_qpP | int | P 帧量化参数。支持设置的值的范围为：**0** - **51**，可选配置。设置为 **0** 时，编码器将用默认值，H264 默认值为 27，HEVC 默认值为 26。 |
| init_qpI | int | I 帧量化参数。支持设置的值的范围为：**0** - **51** ，可选配置。设置为 **0** 时，编码器将用默认值，H264 默认值为 27，HEVC 默认值为 26。 |

下页继续

 

表 5.3 – 续上页

| | | |
|---|---|---|
| init_qpB | int | B 帧量化参数。支持设置的值的范围为：**0** - **51** ，可选配置。设置为 **0** 时，编码器将用默认值，H264 默认值为 27，HEVC 默认值为 26。 |
| qp | int | 恒定 QP 参数设置方法。同社区 FFmpeg cqp，支持设置的值的范围为：**0** - **51** ，默认值为 **0** ，表示不设置 qp。 |
| vbr_minqp | int | 可变比特率模式，并提供 MinQP，同社区 FFmepg qmin。支持设置的值的范围为：**0** - **51** ，可选配置，设置为 0 时，编码器将用默认值 1。 |
| vbr_maxqp | int | 可变比特率模式，并提供 MaxQP，同社区 FFmpeg qmax。支持设置的值的范围为：**0** - **51** ，可选配置。设置为 0 时，编码器将用默认值 1。 |
| sar | string | 设置编码器 sar **(width):(height)** 。可以设置为: **16:11** 或直接读取视频流的值，默认为 **NULL** 。 |
| rc | string | 码率控制模式参数。可以设置为: **vbr/cbr/cqp** ，默认为 **vbr** 。 |
| profile | const | 设置编码档次。**h264 编码器**支持设置的值为：baseline、main、high 和 high444p，默认值为 high。**hevc 编码器**支持设置的值为：main、main_still、main_intra 和 main10，默认值为 **main** 。 |
| level | const | 设置编码级别。**h264 编码器**支持设置的范围为：1、1.0、1b、1.0b、1.1、1.2、1.3、2、2.0、2.1、2.2、3、3.1、3.2、4、4.0、4.1、4.2、5、5.0、5.1、5.2、auto。默认值为 4.1。**hevc 编码器**支持设置的范围为：1、1.0、2、2.0、2.1、3、3.1、4、4.0、4.1、5、5.0、5.1、5.2、6、6.0、6.1、6.2、auto。默认值为 1。 |
| coder | const | 仅 h264 视频编码器支持，设置 **h264 编码**熵 (entropy) 模式。支持设置的值为：**cabac** 和 **cavlc** ，默认值为 **cavlc** 。 |

表 5.4: MLU370 硬件平台视频编码器参数

| 参数 | 类型 | 描述 |
|---|---|---|

下页继续

 

表 5.4 – 续上页

| | | |
|---|---|---|
| device_id | int | 选择使用的加速卡，设置多卡时使用。支持取值范围为：**0** - **MAX_DEVICE_COUNT** 。其中 **MAX_DEVICE_COUNT** 为加速卡总数减 1，默认值为 **0** 。 |
| stride_align | int | 编码器输出对齐的步长。支持设置范围为：1/2/4/8/16/32/64/128。默认值为 **1** 。 |
| rdo_level | int | 编码器 rdo 等级。等级越低，速度越快，质量越低。支持取值范围为：**0～4** ；默认值为 **0** 。 |
| stream_type | int | 编码器视频流的类型。支持设置的值为：**byte_stream/nalu_stream** 。默认值为 **byte_stream** 。 |
| rc | int | 编码器的码率控制模式。目前支持设置的值为：crf/cbr/vbr/cvbr/fixedqp，默认值为 **vbr** 。 |
| init_qpI | int | fixedqp 码流控制模式下的 I 帧 qp。支持取值范围为：**0** - **51** ，默认值为 **30** 。 |
| init_qpB | int | fixedqp 码流控制模式下的 B 帧 qp。支持取值范围为：**0** - **51** ，默认值为 **32** 。 |
| init_qpP | int | fixedqp 码流控制模式下的 P 帧 qp。支持取值范围为：**0** - **51** ，默认值为 **30** 。 |
| qp | int | 码流控制中的 qp 值。支持取值范围为：**0** - **51** ，默认值为 **51** 。 |
| qmin | int | 码流控制中的最小 qp。支持取值范围为：**0** - **51** ，默认值为 **0** 。 |
| qmax | int | 码流控制中的最大 qp。支持取值范围为：**0** - **51** ，默认值为 **51** 。 |
| ctb_rc | int | 块级码率控制开关，可以开启编码的块级码率控制。支持取值为：disable/sub/obj/both；默认值为 **disable** ，即不开启。sub 表示主观模式；obj 表示客观模式。both 表示同时兼顾主观和客观模式以及用户自定义模式。 |
| target_quality | int | 编码的目标质量，仅在 rc 设置为 crf 时生效。支持取值范围为：**0** - **51** ，默认值为 **0** 。 |

下页继续

 

表 5.4 – 续上页

| | | |
|---|---|---|
| lookahead | int | 编码时先行深度的大小，开启之后，进行 2pass 编码，lookahead 值越大则画质更好，编码延迟越高。支持取值范围为：**0** - **40**；在设置为 0~3 时都视为 0，不开启 2pass 编码；在 rc 设置为 crf 时必须开启。默认值为 **0**，不开启。 |
| block_size | int | 块级码控中的宏块颗粒度大小。**仅在 ctb_rc 的值不为 disable 时生效**。块级码率控制的宏块大小，宏块颗粒度越小，QP 调整的越精细，相应的码率会越高；宏块颗粒度越大，QP 调整的越粗，相应的码率会越低。支持取值范围为：**64/32/16**，默认值为 **64**。 |
| ctb_row | int | 块级码控中设置行级的 qp 步长，只在 ctb rc 为 obj 和 both 时生效。支持取值范围为：**0** - **64**，默认值为 **0**。 |
| rc_qp_delta_range | int | 进行码控时宏块与当前帧的最大 qp 差值。支持取值范围为：**0** - **15**，默认值为 **0**。 |
| base_ctb | int | 进行块级码控时的宏块基础复杂度，仅在 ctb rc 设置为 sub 以及 obj 时才能生效。宏块基础复杂度越高，低复杂度宏块 QP 越低；宏块基础复杂度越低，低复杂度宏块 QP 越高。支持取值范围为：**0** - **30**，默认值为 **0**。 |
| tier | int | **HEVC 编码器**中的 tier。支持取值范围为：main/high，默认值为 main。**H264 编码器**不支持该属性。 |
| profile | int | 编码器的 profile 等级。**HEVC** 目前取值范围为：main/main_still/prof_max，默认值为 main。**H264** 目前取值范围为：baseline/main/high，默认值为 main。 |
| level | int | 编码等级控制。**H264** 支持设置的值为：1、1.0、1b、1.0b、1.1、1.2、1.3、2、2.0、2.1、2.2、2.3、3、3.1、3.2、4、4.0、4.1、4.2、5、5.0、5.1、5.2、6、6.0、6.1、6.2、auto。**HEVC** 支持设置的值为：1、1.0、2、2.0、2.1、3、3.1、4、4.0、4.1、5、5.0、5.1、5.2、6、6.0、6.1、6.2、auto。默认值均为 **5.1**。 |

 

## 5.3 硬件图像编码器

### 5.3.1 FFmpeg-MLU 支持的硬件图像编码格式

- JPEG/MJPEG Codec 名称：`mjpeg_mluenc`

### 5.3.2 硬件图像编码器支持的配置选项

FFmpeg-MLU 的硬件图像编码器提供了专属的配置选项，可通过社区通用指令查询：

```
# 查询支持的 MLU 硬件编码器类型
ffmpeg -encoders | grep mlu
# 查询专属配置选项
ffmpeg -h encoder=mjpeg_mluenc
```

在基于 API 开发的工程中，上述专属选项可以通过 `av_dict_set` 或 `av_dict_set_int` API 设置；在命令行调用中，上述专属选项可以通过在编码器后作为参数设置；详见社区 FFmpeg(https://ffmpeg.org/documentation.html)。

图像编码器仅支持 jpeg 格式的图像或 mjpeg 格式的视频；在对图像编码器专属选项详细介绍如下：

表 5.5: MLU200 硬件平台图像编码器参数

| 参数 | 类型 | 描述 |
| --- | --- | --- |
| device_id | int | 选择使用的加速卡，设置多卡时使用。支持取值范围为：**0** - **MAX_DEVICE_COUNT** 。其中 **MAX_DEVICE_COUNT** 为加速卡总数减 1，默认值为 0。 |
| instance_id | int | 选择使用的图像编码器序号。支持取值范围为：**-1** - **MAX_INSTANCE_COUNT** 。其中 **MAX_INSTANCE_COUNT** 为 JPU 硬件总数减 1。-1 表示自动选择，默认值为 -1。 |
| quality | int | 编码器编码 jpeg 图片的质量，数值越大编码越慢，质量越高。支持取值范围为：**1** - **100** ，默认值为 **80** 。 |

表 5.6: MLU370 硬件平台图像编码器参数

| 参数 | 类型 | 描述 |
| --- | --- | --- |

下页继续

 

表 5.6 – 续上页

| | | |
|---|---|---|
| device_id | int | 选择使用的加速卡。支持取值范围为：**0** - **MAX_DEVICE_COUNT**。其中 **MAX_DEVICE_COUNT** 为加速卡总数减 1，默认值为 **0**。 |
| stride_align | int | 编码器输出对齐的步长。支持设置范围为：1/2/4/8/16/32/64/128，默认值为 **1**。 |
| quality | int | 编码器编码 jpeg 图片的质量，数值越大编码越慢，质量越高。支持取值范围为：**1** - **100**，默认值为 **75**。 |

## 5.4 硬件 mlu-filter

### 5.4.1 FFmpeg-MLU 支持的硬件 mlu-filter

FFmpeg-MLU 开发了各种 **颜色空间转换**和 **尺寸变换**功能的 **mlu-filter**；**mlu-filter** 基于 **MLU-CNCV 硬件加速库**实现，在 MLU200 和 MLU300 平台支持的硬件 **mlu-filter** 类型和参数一样，API 接口完全兼容；**mlu-filter** 提供了专属的配置选项，可通过社区通用指令查询：

**注意：**

- 要启用 **mlu-filter** 需要在编译 FFmpeg-MLU 时设置编译选项 **--enable-mlufilter**，默认关闭。
- 可参考编译指令：- ../configure ... **--enable-mlu --enable-mlumpp --enable-mlufilter** ...。

```
 # 查询支持的 MLU 硬件 filter 类型
ffmpeg -filters | grep mlu
# 查询专属配置选项
ffmpeg -h filter=scale_yuv2yuv_mlu
...
```

在基于 API 开发的工程中，上述专属选项可以通过 `av_dict_set` 或 `av_dict_set_int` API 设置；在命令行调用中，上述专属选项可以通过在调用 filter 的时候作为参数设置，MLU200 和 MLU370 硬件平台 **mlu-filter** 详细介绍如下：

表 5.7: MLU200 和 MLU370 硬件平台 mlu-filter 说明

| filter 名称 | 功能说明 | 使用 |
|---|---|---|

下页继续

 

表 5.7 – 续上页

| | | |
|---|---|---|
| scale_yuv2yuv_mlu | 图像尺寸变换 filter，仅支持 YUV 格式图像 (NV12/NV21)，同时还支持 crop 功能。最后两个参数 x 和 y 分别表示起始点的横纵坐标。要求输出宽度和高度必须为偶数，图像像素深度为 8u。 | -vf filter=w : h : n : x : y。w: 输出图像宽度；h: 输出图像高度；x:crop 的 x 坐标；y:crop 的 y 坐标。 |
| scale_rgbx2rgbx_mlu | 图像尺寸变换 filter，仅支持 RGBX 格式图像，同时还支持 crop 功能。最后两个参数 x 和 y 分别表示起始点的横纵坐标。要求输入图像的宽高均不超过 8192 像素，图像像素深度为 8u。 | -vf filter=w : h : n : x : y。w: 输出图像宽度；h: 输出图像高度；n: 设备号；x:crop 的 x 坐标；y:crop 的 y 坐标。 |
| cvt_yuv2rgbx_mlu | 颜色空间转换 filter。输入 YUV 格式 (NV12/NV21)，输出 RGBX(RGB24、BGR24、RGBA、BGRA、ARGB、ABGR) 格式。要求输入图像的宽高均不超过 8192 像素，且均为偶数，图像像素深度为 8u。 | -vf filter=<out_fmt>:<dev_id>。o_fmt: 输出像素格式；n: 设备号。 |
| cvt_rgbx2yuv_mlu | 颜色空间转换 filter。输入 RGBX(RGB24、BGR24、RGBA、BGRA、ARGB、ABGR) 格式，输出 YUV 格式 (NV12/NV21)。要求输入图像的宽高均不小于 2，不超过 8192 像素，且均为偶数，图像像素深度为 8u。 | -vf filter=<out_fmt>:<dev_id>。o_fmt: 输出像素格式；n: 设备号。 |
| cvt_rgbx2rgbx_mlu | 颜色空间转换 filter。输入 RGBX(RGB24、BGR24、RGBA、BGRA、ARGB、ABGR) 格式，输出 RGBX(RGB24、BGR24、RGBA、BGRA、ARGB、ABGR) 格式。要求输入图像的宽高均不小于 2，不超过 8192 像素，且均为偶数，图像像素深度为 8u，输入和输出图像 RGB 格式不相同。 | -vf filter=<out_fmt>:<dev_id>。o_fmt: 输出像素格式；n: 设备号。 |

下页继续

 

表 5.7 – 续上页

| | | |
|---|---|---|
| scale_cvt_yuv2rgbx_mlu | 图像颜色空间转换和图像尺寸变换融合 filter，先将图像颜色空间转换为指定 RGBX 系列，再调整到指定尺寸。输入 YUV 图像 (NV12/NV21)，输出 RGBX 图像 RGB24、BGR24、RGBA、BGRA、ARGB、ABGR。要求输入图像的宽高均不小于 2，不超过 8192 像素，且均为偶数，图像像素深度为 8u，且输入与输出的数据类型必须相同。 | -vf filter=<dst_width>:<dst_height>:<out_fmt>:<dev_id>。w: 输出图像宽度；h: 输出图像高度；o_fmt: 输出像素格式；n: 设备号。 |

 

# 6 常用命令行

## 6.1 Baseline 编码测试

```
ffmpeg -benchmark -re -i input.mkv -c:v h264_mluenc -f null -
```

## 6.2 视频质量测试

这里介绍了如何通过 PSNR 和 SSIM 方法测试视频质量：

```
# PSNR 测试
ffmpeg -i src.h264  -i dst.h264  -lavfi psnr="stats_file=psnr.log" -f null -
# SSIM 测试
ffmpeg -i src.h264  -i dst.h264  -lavfi ssim="stats_file=ssim.log" -f null -
```

## 6.3 解码缩放

```
# 解码 h264 码流，使用 cpu-filter 缩放成 cif，然后使用默认 cpu 编码器编成 h264 码流
ffmpeg -vsync 0 -c:v h264_mludec -i input_h264_1920x1080.mkv -vf scale=352:288 output_352x288.
↪h264

# 解码 h264 码流，使用 mlu-filter 缩放成 cif，然后使用 mlu 硬件编码器编成 hevc 码流
ffmpeg -vsync 0 -c:v h264_mludec -i input_h264_1920x1080.mkv -vf scale_yuv2yuv_mlu=352:288:0 -
↪c:v hevc_mluenc output_352x288.h265

# hwaccel 硬件加速模式，解码 h264 码流，使用 mlu-filter 缩放成 cif，再使用 mlu 硬件编码器编成 hevc
码流，不涉及 host 侧和 device 侧之间的数据拷贝，全程在设备侧进行
ffmpeg -vsync 0 -hwaccel mlu -hwaccel_output_format mlu -hwaccel_device 0 -c:v h264_mludec -i␣
↪input.h264 -vf scale_yuv2yuv_mlu=352:288:0 -c:v hevc_mluenc output_cif.hevc
```

 

## 6.4 1:1 无缩放转码

```
# 无缩放转成指定码率的视频
ffmpeg -vsync 0 -c:v h264_mludec -i fhd_input.mkv -c:a copy -c:v h264_mluenc -b:v 5M output.mp4

# hwaccel 模式无缩放转成指定码率的视频
ffmpeg -hwaccel mlu -hwaccel_output_format mlu -c:v h264_mludec -i input_file -c:v h264_mluenc␣
↪-b:v 5M output_file.h264
```

## 6.5 1:N 可缩放转码

```
# 1:N 可变尺寸转码成指定码率的视频
ffmpeg -vsync 0 -c:v h264_mludec -i fhd_input.mkv -vf scale=1280:720 -c:a copy -c:v h264_
↪mluenc -b:v 2M output1.mp4 -vf scale=640:360 -c:a copy -c:v h264_mluenc -b:v 512K output2.mp4
```

## 6.6 解码 + MLU Filter + 编码

```
# mlu 解码 + mlu-filter + mlu 编码的转码 pipeline
ffmpeg -vsync 0 -c:v h264_mludec -i input_1920x1080.mkv -vf scale_yuv2yuv_mlu=320:240:0 -c:v␣
↪h264_mluenc output_320x240.h264

# mlu 编解码和多个 mlu-filer 联合使用的 pipeline
ffmpeg -vsync 0 -c:v h264_mludec -i input_1920x1080.mkv -vf cvt_yuv2rgbx_mlu=rgb24:0,cvt_
↪rgbx2yuv_mlu=nv12:0 output.h264

# hwaccel 模式，mlu 解码 + mlu-filter + mlu 编码的转码 pipeline
ffmpeg -hwaccel mlu -hwaccel_output_format mlu -c:v h264_mludec -i input_file -vf scale_
↪yuv2yuv_mlu=320:240:0 -c:v h264_mluenc output_file.h264
```

 

# 7 FAQ

## 7.1 常见的编译问题

(1) 找不到 BangWare 相关动态库，可能是未指定环境变量 ${BANGWARE_HOME} 的路径；请检查该环境变量是否正确设置，未设置的情况下请手动 export 指定 ${BANGWARE_HOME} 的路径，并确认相关动态库依赖安装正确。

(2) 动态库 codec 头文件出现错误，可能是 patch 选择错误或者执行 configure 时 extra-libs 设置错误，建议根据FFmpeg-MLU 编译及运行 重新进行编译及安装。

(3) 编译时出现有些变量未定义。可能是使用的 cntoolkit 和 driver 包版本与 FFmpeg-MLU 版本不匹配，建议根据《寒武纪 FFmpeg-MLU 用户手册》中规定的版本进行选择安装。

## 7.2 常见的运行时问题

(1) 解码输出帧数异常。解决办法：在编解码时加入 -vsync 0 或者 -vsync 2 选项，防止 FFmpeg 在编解码时生成重复帧和额外帧的 yuv 数据。

(2) 容器中运行报错找不到 MLU 设备，尝试 docker run 上使用 privileged 参数或者 device 参数将 MLU 设备映射到容器中。

(3) 运行中发现 cnmon info vpu 和 jpu 显示占用为 0，可能是指定了错误的设备号，建议修改再尝试。

(4) 运行编码器出错，查看是否设置了编译选项 --enable-mlu 和 --enable-mlumpp。

(5) 运行 mlu filter 出错，查看是否设置了编译选项 --enable-mlu 和 --enable-mlufilter。

## 7.3 常见功能和性能问题

(1) 编码性能影响因素很多，不同配置情况下，性能差异很大；FFmpeg-MLU 支持测试特定配置下的性能，用户可以自行修改不同参数组合进行测试。

(2) 解码器是否有 resize 后处理操作：MLU200 和 MLU370 视频解码器支持解码模块 resize 后处理操作，具体请见硬件视频、图像解码器 。

(3) 解码器是否有 crop 后处理操作：MLU200 解码器无 crop 后处理操作；MLU370 解码器支持 crop 操作，具体请见硬件视频、图像解码器 。

 

(4) 网络拉流失败无法播放或者路数不达标：可能是网络的问题，受网络带宽限制；若直接来自摄像头，建议一个摄像头最多不要超过 3 路拉流。

(5) 若使用的 CPU 本身性能较低，则可以通过控制编解码及 `filter` 的线程数来提升性能；使用 API 开发的工程可以在创建编解码器之前设置 `thread_count` 参数；使用命令行方式，可以在指定编解码器的时候设置 `-threads` 参数。

(6) 若对解码输出图像顺序无要求，则可以设置解码输出顺序为 decode；以为 decode 模式比 display 模式延时更低。

 