## 3.1环境的搭建与可行性测试：

下载Cmake 3.14.1安装包：https://cmake.org/download/

下载libVA 2.4.1安装包: <https://github.com/intel/libva>

下载FFmpeg 4.1.3 ：http://www.ffmpeg.org/download.html

下载OpenVINO 2020 R3: <https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit/download.html>

请务必先安装好OpenVINO toolkit

本实验默认所有安装包下载至 /home/ 目录下，且所有系统路径为系统默认路径。

本方案实验环境：

硬件：Intel® Core™ i5-7200U 2500GHzx4, HD Graphics 620 (KBL GT2), MEM: 8G DDR4

软件版本：Linux 18.04.4 LTS, OpenVINO 2020.3，FFmpeg 4.3.1, LibVA: 2.1.0, VA-API version 1.6.0

安装步骤：

先安装Libva的所有依赖（遵循github上libVA的依赖项安装）：

*#安装Cmake*

*cd /home/CMake/*

*./configure*

*make -j8*

*sudo make install*

*#安装libva前， 首先删除系统里的libva*

*echo "remove other libva"*

*find /usr -name "libva\*" | xargs rm -rf*

*find /usr -name "lidrm\*" | xargs rm -rf*

*#安装mediasdk driver的依赖 以及保证 cmake版本为3.6.0及以上、*

*sudo apt install autoconf libtool libdrm-dev xorg xorg-dev openbox libx11-dev libgl1-mesa-glx libgl1-mesa-dev*

*#安装libva*

*cd /home/libva/*

*./configure*

*make*

*sudo make install*

在安装好LIBVA之后，需要设置相关的环境变量

*#设置libva环境变量*

*export LIBVA\_DRIVER\_NAME=iHD*

*export LIBVA\_DRIVERS\_PATH=/usr/local/lib/dri （请务必确保media driver编译出来的iHD\_drv\_video.so在此路径下）*

*export LD\_LIBRARY\_PATH=/usr/local/lib:/usr/lib64:$LD\_LIBRARY\_PATHsudo make install*

运行

*vainfo*

如果出现以下结果，则说明LibVA成功安装

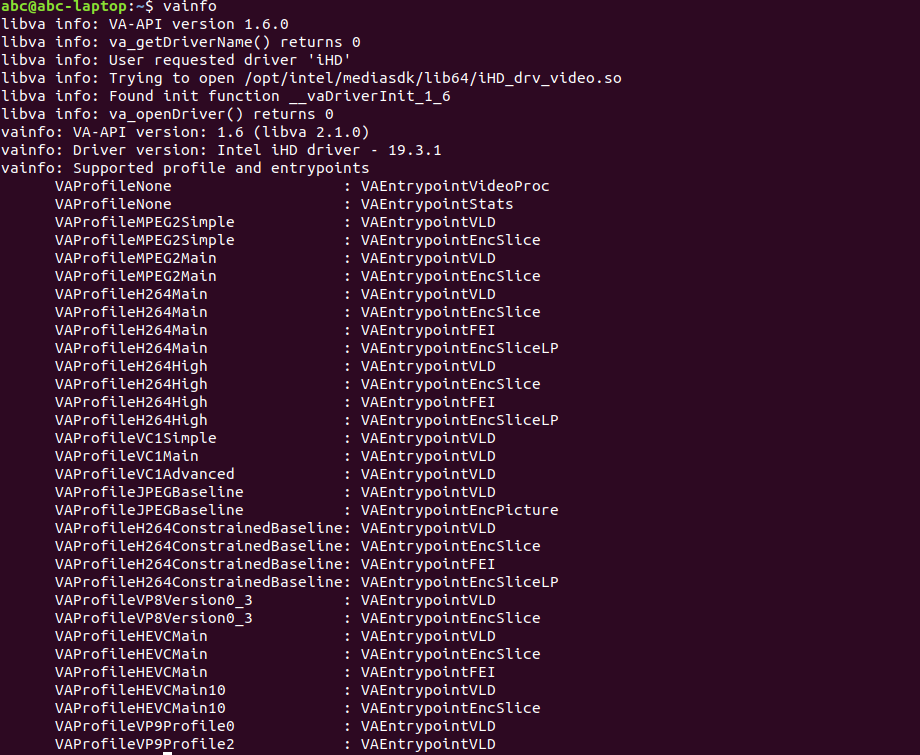


图12：vainfo 示例图

之后我们进行FFmpeg的安装，并且需要在编译的时候链接VAAPI。

首先，安装FFmpeg的一些必要依赖

*sudo apt-get update -qq && sudo apt-get -y install \*

*autoconf \*

*automake \*

*build-essential \*

*cmake \*

*git-core \*

*libass-dev \*

*libfreetype6-dev \*

*libsdl2-dev \*

*libtool \*

*libva-dev \*

*libvdpau-dev \*

*libvorbis-dev \*

*libxcb1-dev \*

*libxcb-shm0-dev \*

*libxcb-xfixes0-dev \*

*pkg-config \*

*texinfo \*

*wget \*

*zlib1g-dev*

*sudo apt-get install nasm*

*sudo apt-get install yasm*

安装FFmpeg，并且集成VAAPI

*# cd 至安装文件夹 安装FFmpeg with VAAPI*

*cd /home/FFmpeg-4.1.3/*

*./configure \*

*--disable-lzma \*

*--enable-pic \*

*--enable-nonfree\*

*--disable-stripping\*

*--enable-hwaccel=h264\_vaapi*

*make -j8*

*sudo make install*

*make -j8*

*sudo make install*

这个值得注意的是，我们必须要带有，一定要加--disable-stripping， 如果不加此选项，FFmpeg在编译时，会使用strip去掉符号信息，导致无法进行内部步进调试。H264\_vaapi则是我们要集成的项。

安装完成之后我们分别查看安装是否成功

*# 查看FFmpeg 中集成的带有vaapi字符的编解码器*

*FFmpeg -codecs | grep vaapi*

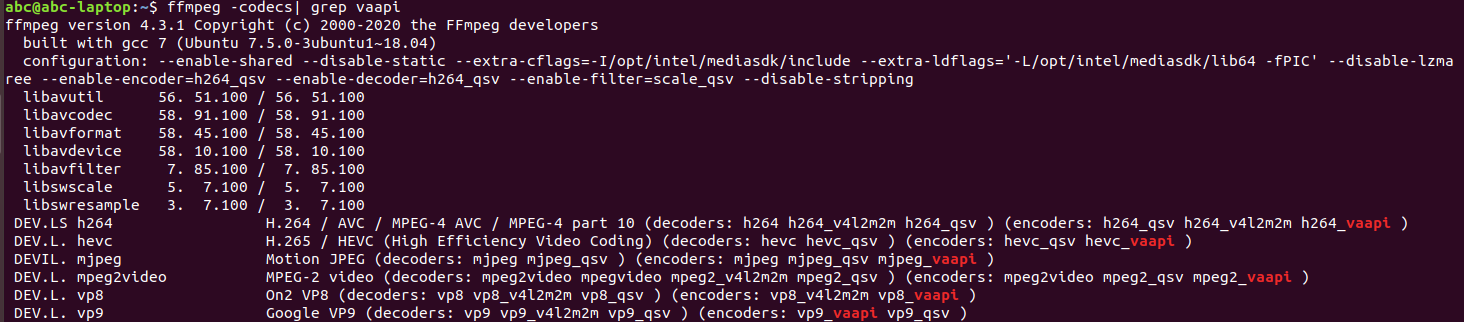
你将会看到类似如下的输出结果：

图13：VAAPI编解码器示例图

我们可以看到在例如h264格式下，已经成功集成了h264\_VAAPI的编码器

*# 查看FFmpeg 中集成的带有vaapi字符的滤镜*

*FFmpeg -filters | grep vaapi*

你将会看到类似如下的输出结果：

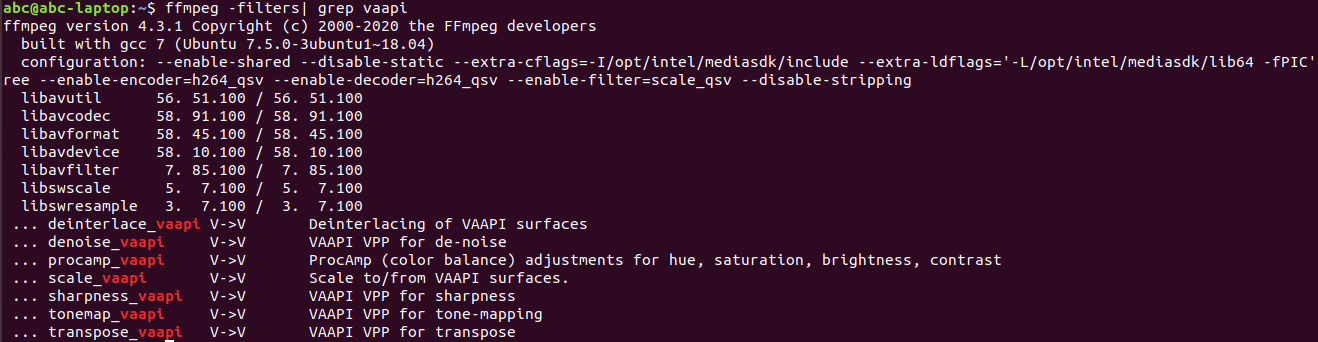
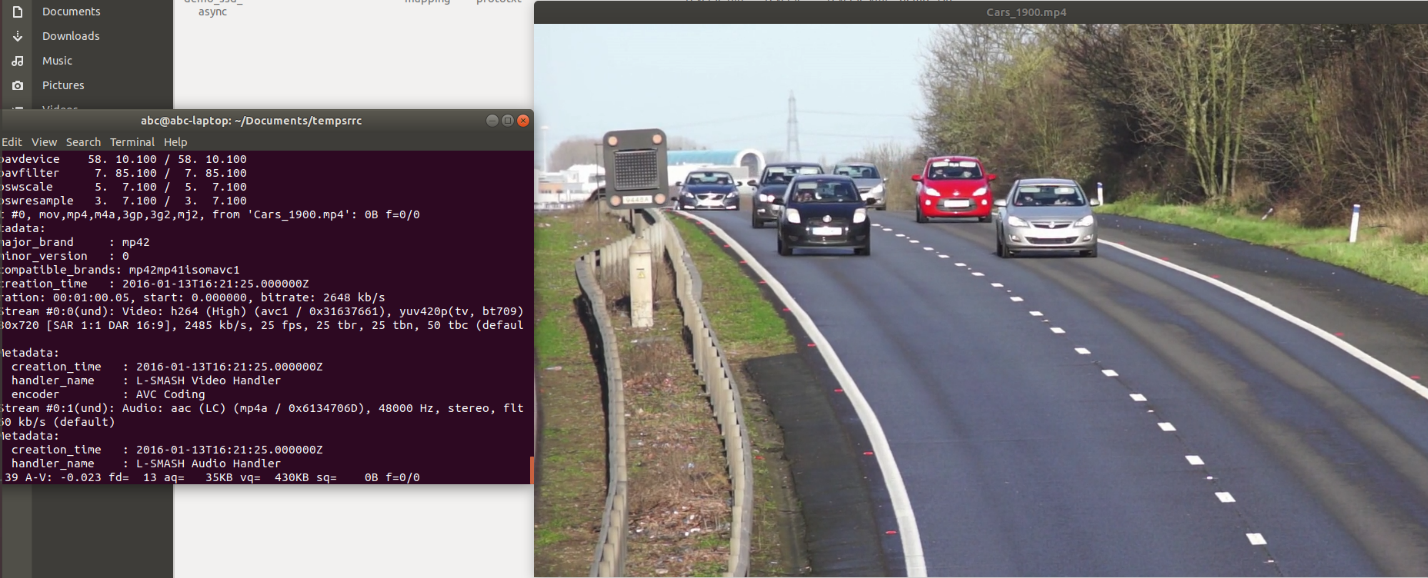


图14：VAAPI滤镜示例图

我们知道FFmpeg本身就是一款非常好用的命令行工具，它可以直接使用命令行的相关参数，以达到视频处理的目的。当我们成功安装好FFmpeg并且将VAAPI 成功集成之后，我们可以使用命令行来测试我们的整体工作流程，也就是使用VAAPI解码，然后使用VAAPI所属的scale来进行解码之后视频的缩放等操作。

通过命令行的方式，我们实现了将原先的视频通过VAAPI解码然后经过scale\_vaapi长款缩小一半。

*# 使用VAAPI 命令行 操作 进行解码和所发*

*FFmpeg -hwaccel vaapi -hwaccel\_device /dev/dri/renderD128 -hwaccel\_output\_format vaapi -i Cars\_1900.mp4 -vf ‘scale\_Vaap=iw/2:ih/2 -c:v h264\_vaapi -b:v 2M output.mp4’*

图15：实验视频缩放原图

可以看到下图的尺寸已经从1280x720变成了640x360。

图16：实验视频缩放结果图

如果，系统中没有安装cl/cl2（include cl.h时找不到文件），需要安装 cl/cl2，

*apt install opencl-clhpp-headers opencl-c-headers ocl-icd-opencl-dev*

当你完成main.cpp的代码后，修改编译的makefile使之包含ffmpeg的所有库：

*#ffmpeg*

*set (FFMPEG\_LIBS "avdevice avformat avfilter avcodec swresample swscale avutil")*

*set (CFLAGS "$(shell pkg-config --cflags ${FFMPEG\_LIBS})" ${CFLAGS})*

*set (LDLIBS "$(shell pkg-config --libs ${FFMPEG\_LIBS})" ${LDLIBS})*

*INCLUDE\_DIRECTORIES("/usr/local/include")*

*LINK\_DIRECTORIES("/usr/local/lib")*

备注：

* **若 编译中config.h 不能被找到，请将/ffmpeg/config.h复制至检索目录**
* **我测试的环境比较简单，我直接将以上#ffmpeg这段makefile脚本，添加至OpenVINO的Cpp sample下的Cmakelist文件中进行使用，使用原先OpenVINO原生的Cmakelist文件进行编译。若没有时间进行编译文件开发的同学，也可以按照这种发现进行编译**

Tips：也可以通过命令行的方式直接使用ffmpeg和openVINO，但是没有使用remoteblob会涉及到iGPu读数据时两次无用的拷贝。

在FFMPEG VIDEO ANALYTICS(FFVA) plugin实现了将OpenVINO™的推理功能集成于FFmpeg的filter之中，具体的项目的地址是<https://github.com/VCDP/FFmpeg-patch>。通过这个项目可以相对简单验证，命令行中的VAAPI缩放加上OpenVINO™推理。

*# 使用VAAPI 命令行 操作 进行解码和所发*

*FFmpeg -hwaccel vaapi -hwaccel\_device /dev/dri/renderD128 -hwaccel\_output\_format vaapi -i Cars\_1900.mp4 -vf ‘scale\_Vaap=iw/2:ih/2,detect=model=/root/intel\_models/mobilenet-ssd.xml:device=GPU:nireq=2,ocv\_overlay,format=nv12,hwupload’ -y -c:v h264\_vaapi output.mp4’*

**

图17：实验缩放结果推理图

可以看到输出图像的分辨率是640x360，并且图中的小车可以被OpenVINO™所识别，然后标记了一个绿色的方框在图像上。所以这两个小实验就证明了我们整个workload就是可以被实现的，第一部分是使用集成好VAAPI的FFmpeg进行硬解，得到裸帧数据保存在GPU的缓存中，第二部分由于命令行我们需要使用filter：hwdownload，hwupload，来移动数据从CPU移动到GPU，或者是从GPU移动到CPU,这势必会造成资源的浪费。于是我们使用source code的方式，并且借助于Remote blob的高效性，我们可以直接通过Remote blob把帧数据给到GPU中，整个程序使用C语言进行程序编写。这样的方法可以直接在GPU解码并且经过缩放之后直接将帧的缓存数据给GPU进行推理而不需要将数据拷贝回CPU做处理。