一种适用于异构视频编解码设备的应用层编程框架

一. 选题的背景及意义

1.1 视频编解码

1.1.1视频编解码介绍

随着计算机、多媒体和数据通信技术的迅速发展, 数字视频的应用越来越广, 由于数字视频数据量巨大, 不利于传输、存储和播放, 采用视频编解码技术, 将数据以压缩可以解决这一问题。数字视频编码技术是数字信息传输、存储等环节的前提, 也是数字视频产业的基础【1】。

1.1.2 解决方案简介

- 视频编解码解决方案，从底层硬件到上层应用开发，每一层次都有很多种实现方式。
- 针对底层硬件，可使用`CPU`直接处理，这就是软件转码，纯CPU计算，但浪费了CPU资源。或者是目前流行的`CPU+GPU`模式，这种方案应用较广。或者是`Intel`的`Intel Graphics`系列，这种是将显示核心集成在CPU中，还有很多种其它的异构模式的解决方案。
- 在算法的选择和使用上，可选择的方案也很多，包括`ISO/IEC`制定的`MPEG-x`和`ITu-T`制定的`H.26x`两大系列视频编码国际标准。及目前广泛使用的`H.264`.或者其它公司专有的算法`RealVideo`、`Nancy`与`WindowsMediaVideo(WMV)`等。
- 上层应用的实现方案也很多，包括特定公司开发出来的视频解决方案，比如`Intel Media SDK`就是针对`Intel Graphics`开发的应用实现方案，或者是开源的ffmpeg，这个应用就更广泛了，是一套完整的，跨平台的音视频处理方案。

1.2 当前主流解决方案

1.2.1 软件转码

主要是利用CPU来完成大量而复杂的计算，效率较低，占用CPU资源。但目前Intel推出的`Intel Graphics HD`系列芯片，里面集成了显示核心。但也只能应用于小需求的市场。本质上也相当于一个专用的视频处理设备,只是这个芯片被封装到了CPU中,称为显示核心.

1.2.2 异构编解码设备

- 异构编解码设备主要是指为了实现视频的编解码的专用设备，一般是指外置的芯片,可以是集成在主板上的显卡设备,也可以是像GPU那样独立的设备,还有可能集成到CPU中,也有的像外围的IO接口那样的设备.虽然都是为了实现特定的视频处理要求,但形态各异,底层的硬件和驱动也是大不相同的。而在应用层上，视频的处理逻辑可能会类似。
- 目前主流的异构编解码设备，包括GPU模型，`Intel Graphics HD`系列也可算作是其中的一种不同实现方案。还有众多的各大研究机构或公司开发出的视频编解码设备等一系列解决方案。

1.3 异构设备解决方案的不足

1.3.1 方案繁杂,标准众多

- 虽然异构设备解决方案很多,但大部分都是公司或研究机构所有,一般只给开发者或用户提供相应SDK.而提供的SDK也只适用于对应的硬件设备.不论从语言,接口定义都差异很大,具有很大的局限性.不同的设备没有移植的可能.
- 除了语言,硬件设备差异外,设备的连接方式,环境的搭建也是各不相同.
- 一般来说,上层应用的开发者也无法看到SDK或接口的细节,只可能获得一份开发文档.当出现问题,也不能从深入底层,从根本上解决.如果有一套标准,并且各异构设备提供者都能按此标准来提供接口,那么开发就会很容易.

1.3.2 开发难度大,维护困难

- 对于开发者,除了要熟悉接口和提供的文档外,还需要在此基础上开发上层应用程序.
- 若有更新,开发者需要对接口很熟悉才行,如果只了解上层应用,是无法完成对接的.
- 从一个设备移植到另一个设备,上层应用必须重新开发,这就需要开发者重新学习新异构设备的特性和文档.这样加大了开发难道.而程序的移植往往是比较常见的.
- 开发语言的差异,也导致了移植的困难.

二 国内外本学科领域的发展现状和趋势

2.1 视频编解码国内外研究重点

- 针对于视频编解码，大多数研究重点都是在算法的，包括新算法的提出和以前算法的改进（即如何用更高效的算法来处理视频）。典型的编解码器要么采用行业标准，
  如 `MPEG一2、MPEG一4、H．264／AVC与AVs` ．要么采用专有算法．
  如 `On2、ReaIVideo、Nancy与WindowsMediaVideo(WMV)` 等。目前应用较广的编解码技术(H．264／AVC与Vc一1)，压缩一解压(编解码)算法可以实现数字视频的存储与传输。这两种编解码技术利用可编程DSP与AsIc等低成本IC的处理能力．都能够达到极高的压缩比．不过，在具体的一些性能上这两种编解码技术仍然存在差异。
- 硬件实现（开发或改进硬件的体系结构以适用于视频处理）也是一大研究热点。但主要是各大公司或研究机构开发出自己的硬件设备来实现视频的编解码。
- 另一个研究重点是如何提高编解速度的相关研究，除了算法选用和硬件的实现外，还有其它的因素的研究，可能影响到编解码速度，包括上层应用的数据组织，线程调度等。

2.2 异构设备视频编解码的编程模式

- 异构设备视频编解码优点很多，可定制。异构视频转码设备主要是针对特定应用场景定制的，一般适用场景不多。对于用户，可实现的方案多，可选项多。这也是各公司推陈出新的原因之一.
- 异构设备解决方案太依赖于硬件提供商，底层接口差异性太大，没有一个通用统一的定义接口，导致异构设备编程因难。
- 逻辑层次上，没有一个统一的解决方案，特别是应用层。虽然有众多的异构设备支持OpenCL等类似语言简化编程，　基于 `CPU+GPU` 异构平台的性能优化及多核并行编程模型的研究【7】也是一大热点,但硬件依赖性太大。主要有:
  - 可移植性差，依赖特定的硬件设备。
  - 可编程性差，依赖对应的底层软件接口。整个程序框架规定好了之后，几乎无法更改，除非是硬件提供者，可看到底层代码实现才能改动，用户几无编程的可能，只能在小问题上修改。

- 维护升级困难，可扩展性差。这个是基于上面两个带来的问题

三．研究内容和预期目标

3.1. 研究的主要内容

3.1.1 视频编解码分层

为什么分层

异构视频编解码是一个依赖于特定设备，通过设备提供的驱动和接口来完成视频处理的过程．从设计到实现整个过程，要做到自底向上的全面考虑．如果没有一个层次的概念，而是仅仅按过程来处理，特别是底层的微小改动会产生牵一发动全身的效果，上层也要做相应的修改．这样会花费很大的代价

而目前大部分异构设备都是基于过程来处理的，给程序员提供的就是底层的驱动文档，或者是接口的API函数调用，最后再给一个简单的示例程序．这样，对于开发者来说，要想在不熟悉的异构设备上作开发，需要自底而上的了解各个函数调用和驱动，最后才能按这此示例编写特定的视频应用程序，耗费的时间过多，而且最大的问题就是移植和维护的困难．比如，当程序员设计了一个视频编解码的应用，那么根据逻辑，就要找到底层对应的功能接口，但是，当这套程序移植到另一台不同架构的异构设备的时候，程序就完全不能使用，不是仅仅修改对应的驱动接口这么简单，由于不同设备定义的接口不同，提供的调用也不同，所以更换环境后不得不重新开始再开发一套程序．

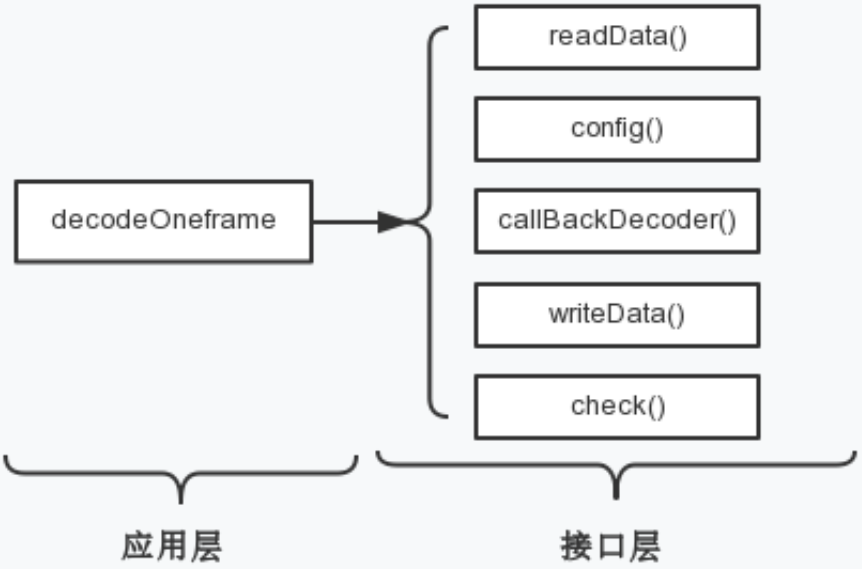
而事实上，这样的工作是不必要的．我们知道，虽然两个不同异构设备提供的接口驱动不同，但在应用方面，对于开发者来说，他要实现的功能是一样的，逻辑结构也大体相同，而对于上层的应用开发人员来说，完全没有必要知道硬件的实现细节甚至接口的调用．同样的，对于一个驱动开发者来说，只要按照特定的标准来开发出驱动程序即可，也完全不用考虑设备的物理特性和上层应用的逻辑．所有的开发和实现，都可以按功能和逻辑分层，这样可以有效的组织和开发．

> 分层如下图一：
>
> 
>
>
> ***图一 分层结构***

分层的依据

给异构视频处理设备自底而上分层，分层的依据是每一层所负责的功能，按功能性来分层。每一层只需要完成本层所规定的任务和提供给上一层可实用功能调用即可。而每一层的实现，都是基于下一层向上提供的功能调用的，并为上一层提供功能接口。这些可以是软件SDK或API，也可以是硬件驱动。从异构设备一直到应用层的视频编解码，有些类似网络中的TCP/IP协议分层的概念。每一层可以独立的看作是一个功能模块，而相邻的两层是依赖和调用的关系。如图一的分层结构.这里随着研究的深入，层次实际上还可以划分得更细，结构功能还可以更明确．目前研究的结果是暂时划分四层．

针对于每一层，都可以作独立的开发．因为当剥离了特定的设备之后，每一层从逻辑上来看，只和相信的上下两层相关，调用下层的接口，同时也为上层提供接口．相邻层的关系可以简单的接口调用来联系．

> 我们以一个最基本的视频处理，就是解码一帧为例，如图二：
>
> 
>
>
> ***图二：`decondeOneframe`(相邻层示例)***

针对于解码功能，在应用层，我们只需要完成解一帧的功能，有接口层，为完成这个功能，可能需要调用如下的接口：

- `readData()` 读取源视频数据到特定的内存区域，这个是事先分配好的区域
- `config()` 配置解码参数，这个也是在初始化时配置好了，此处要把值传递给解码器
- `callBackDecoder()` 针对上面的参数，调用对应的解码器，每个解码器针对了不同的视频格式或者不同的算法，这里就需要异构设备的计算，最后得到解码后的一帧`raw data`
- `writeData()` 对解码后的数据处理，一般是播放出来或者存储
- `check()` 非必需的步骤，但是确认会提高程序的健壮和解码的效率，这里主要检查数据是否足够解码，参数的检查等

分层的优点：

- 对于异构设备的构视频处理，某一层的开发者不需要通篇考虑整个编解码过程，而是根据自己的需求，实现某一层即可。
- 应用层的实现也不再需要从底层一步步完成，开发者可专注于自己所处的层次，而底层实现，可按照类似协议规定好．
- 各层相对独立，相邻层间通过定义好的接口调用交互．
- 便于定位问题．当出现问题或需要添加新的功能，可以很方便的找到问题所有和添加的部位．
- 移植性强．各层间的实现，不论是软件代码，还是硬件功能，只要符合事先规定好的接口调用，可以在不做修改过很少修改的情况下移植．在某些层次，逻辑功能往往是一致的，特别是在应用层．这样，上层的代码或实现的逻辑功能，可以很方便的移植．

层次如下：

- 物理层：
  对于物理层的，各公司或研究机构各有各的实现方式．可以是专用的异构设备，也可是集成于某个芯片的显示核心．目前主流的是显卡或者是显示核心．对于GPU，生产的公司也很多，如表一，这里仅仅列了一些较大的公司，目的是为了说明，异构设备中，光GPU就有如此多的厂家和公司．异构设备可以从属于主机也可独立的运行操作系统．而视频的编解码需要大量的计算，物理层仅是实现高性能计算的芯片，视频编解码的工作仅仅是芯片的一个应用场景．

供应商	产品	应用
Intel	集成显卡	Intel主板
AMD	独立显卡	HD系列
Nvida	独立显卡	Geforce系列
Matrox	2D绘图（工程）	GF5
sis和via	集成	自主生产主板

表一　GPU供应商

- 驱动层：
  驱动层一般由异构设备制造者提供，驱动层应涉及到对硬件的最底层调有用基本的驱动，计算模块的基本调用等．而一般上层的应用是看不到驱动层的实现细节．这一过程封装在接口层的API中，而且针对各个不同的异构设备，实现的细节也是不同的．而上层开发者往往不需要关注这些细节，驱动层只需要完成特定的功能，给上层提供必要的功能调用即可．而实际上，对于这一层，可能有些设备不仅仅是用于视频处理，视频编解码可能只是此设备的一项应用功能而已．而针对不用的驱动，可实现不用的应用方向．
- 接口层：
  利用驱动层提供的基本调用，完成视频处理的基本操作．包括异构设备的启动和关闭，设备的配置和初始化，内存分配和释放，数据结构的构造和析构，包括帧对象，源文件，生成文件的存放组织形式，文件类或结构体的定义，编解码算法步骤的实现．以H.264编码为例，其过程包括帧内预测，帧间预测，

宏块侵害，运动补偿，去块滤波等，在这一层，就要实现这一系列算法，这样上一层就可以直接调用实现了．

- 应用层：
  向用户提供一组常用的应用程序，程序员也可根据应用来完成视频的编解码．根据接口层提供的功能，按照视频处理流程，调用相关的接口，并处理好数据流等．
  异构设备的差异体现在物理层和驱动层，不论是实现方式还是组织模式，都大有不同．但是到接口层，功能性就大体相同了．特别是当开发人员进行视频编解码时，可能只有对接口层调用的API不同了．目前应用层中视频处理以下面几大框架为主，如表二

框架名	产品	优点	缺点	基于
MPC-HC系列	射手,windows系	兼容性	开发复杂，主要windows平台	Direct Show
mplay架构	mplay,mplay系	稳定，兼容	代码结构不清晰	ffmpeg
VLC系列	VLC系	稳定，兼容	纯C开发 硬件加速略有困难	ffmpeg

表二　应用层三大主流框架

3.1.2. 总结归纳出应用层视频处理的一般流程和要点，相当于对应用层分块

- 在应用层面上按功能分块.
  - 越到上层，异构设备视频处理的差异性就越小．如果我们按功能来分块，应用层就会划分成一个个很小的应用块．可以总结归纳出视频转码的通用框架，而在底层及各异构模型的具体实现则不关注．
  - 由前面的分析，接口层提供了关于视频处理的一般处理接口，而应用层只需要根据各类应用的特点，调用相关接口即可完成．而视频处理类的应用主要是以编码解码类的应用为主，目前也暂时关注这两类，至于多路处理，转码，变换缩放等都是在此基础上进行的．对应用层分块，参见本文4.2节．
- 应用层分块的实现过程
  总结一套类似协议的编程框架，类比于网络中的应用层，定义主要的接口和实现细节
  - 针对应用层的编程框架，定义主要的接口和协议，实现视频转码．针对各异构设备，底层的差异最大，越到上层，差异越小，到接口层时，各实现已经大体可统一了，这样为制定一套＂协议＂提供了基础．这是本文重点之一．
  - **＂协议＂**的制定，主要内容是按功能分块的方式，针对某一个基本功能，要具体介绍此功能的作用，需要的接口，需要的数据结构，实现的函数形式，函数的返回值，输入和输出，在分块图中所处的地位．而针对视频处理，基本的功能函数大约有十多种，不同的异构设备虽然接口函数名称各异，但功能却相同．参见本文4.3节介绍．
  - 实现视频编解码的主要过程．视频处理的模块有很多种，我们只关注主要的功能．并在各大平台实现．
  - 需要有一个类似协议的定义和规范．可以是一个草案的规定．并且需要实现一个编译器功能，这个编译器的作用是用于测试．
- 在几种主流异构设备上实现

    - 此乃实验过程，验证论文的观点可行可用．实现的过程包括已经实现的（本人工程中的三个不同异构设备上的实现）和目前常用的异构设备．性能上只要达到原生系统性能就可以证明方式的可行．
    - 所做的实验包括，接口函数的定义，接口函数的重写，实现应用程序即视频编码或解码过程，测试性能（帧率，速度，视频质量）．性能要求是不得低于原生系统．
- 对比各设备中视频转码的数据，分析并修正各接口与实现
    - 对文章论点和规范作出修正和改进．
    - 这里面可预测的困难有：改写接口后可能会出现调用错误，性能可能大幅下降，可能功能无法都完成，只能实现一些功能等．
- 视频其它处理

  其它处理是包括大小缩放，转置，编码算法的选择，视频截取，嵌入音频等，这些可在分层的层次中指出，并具体到应用场景

3.2 预期目标

3.2.1归纳总结出视频编解码的逻辑结构，提出基于应用层的视频编解码框架

编程框架包括：

- 视频编解码的一般流程，主要是在应用层，对应用层功能分块
- 实现视频编解码的接口函数的定义，实现，功能介绍
- 在现有异构设备上实现或对现有接口实现一个统一的修改，以符合本编程框架．

3.2.2 涉及的点应包含主要功能

- 视频处理有太多的功能模块和细节，可能无法找到通用统一的归纳，但针对主要的功能，需要有很明确清晰的研究。本文目前关注的是主要的编解码模块.
-

3.2.3 在主要的异构平台上实现

- 对改进后的接口，其性能可与原生的接口性能作对比，如果性能相近，可证明提出的编程框架是可行的
- 实现后可对比主要数据,比如速度,质量等

3.2.4 实验验证

- 实验对象是根据此编程框架得到的编解码速度与原来异构设备提供的demo进行对比,主要比较解码速度
- 改进后的编程框架解码速度应该与原demo相差不大,这样即达到了目的.如果逻辑功能正确,数据处理及时,可能效果还要好.

四. 研究方法，实验方案，技术路线及可行性分析

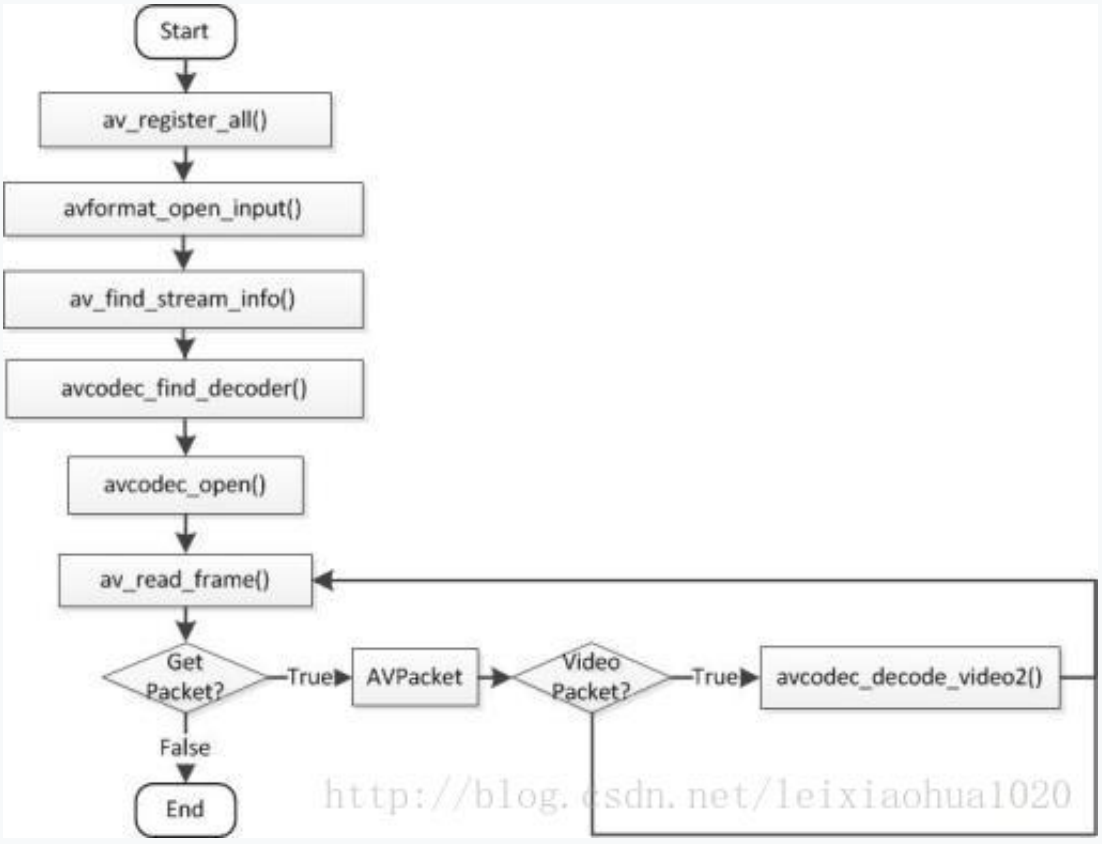
4.1 借鉴项目中已有成果

4.1.1 `Intel HD Graphics` 转码方案。

调用的是 `Intel Media SDK` 接口，可实现多路视频实时转码。在 `Intel® Core™ i3-3110M CPU @ 2.40GHz` 处理器上(显示核心是 `Intel HD Graphics 4000` 基本频率650MHz)，可实现5路1080p实时转码(h264转h264)

4.1.2 `ffmpeg` 开源转码框架

其中H.264格式视频编码，改写成调用 `Intel HD Graphics` 硬件编码方案，即为此编写硬件编码器



ffmpeg解码接口

4.1.3 `Godson-dm` 芯片，使用的是 `Coda980` 编程接口。

`Coda980 Video codec IP` 是 `Chips&Media【4】` 的视频处理方案，使用的异构设备是 `Godson-dm` 芯片。目前已基于此完成了多个相关项目，包括解码，编码，转码，多路解码等，达到了很好的效果。这套方案也是我们借鉴的重要依据。

4.2视频编解码应用层的一般逻辑

在应用层,各异构设备实现的功能是相同的,只是底层提供的接口或调用不同.这样就可以把应用层按功能模块统一起来,不论它们实现的编程语言是什么,大部分是C/C++类的,都可以按照这一框架统一起来.大体分下面几个模块.

4.2.1设备开启与初始化

- 包括打开设备，获取设备状态,分配设备号,初始化相关数据结构,建立线程等过程
- 不同硬件差异性比较大，但必须打开设备,初始化后才能使用，整个打开过程相对独立，打开后就不会影响后面的操作。

- 这里可以以`Coda980`为例

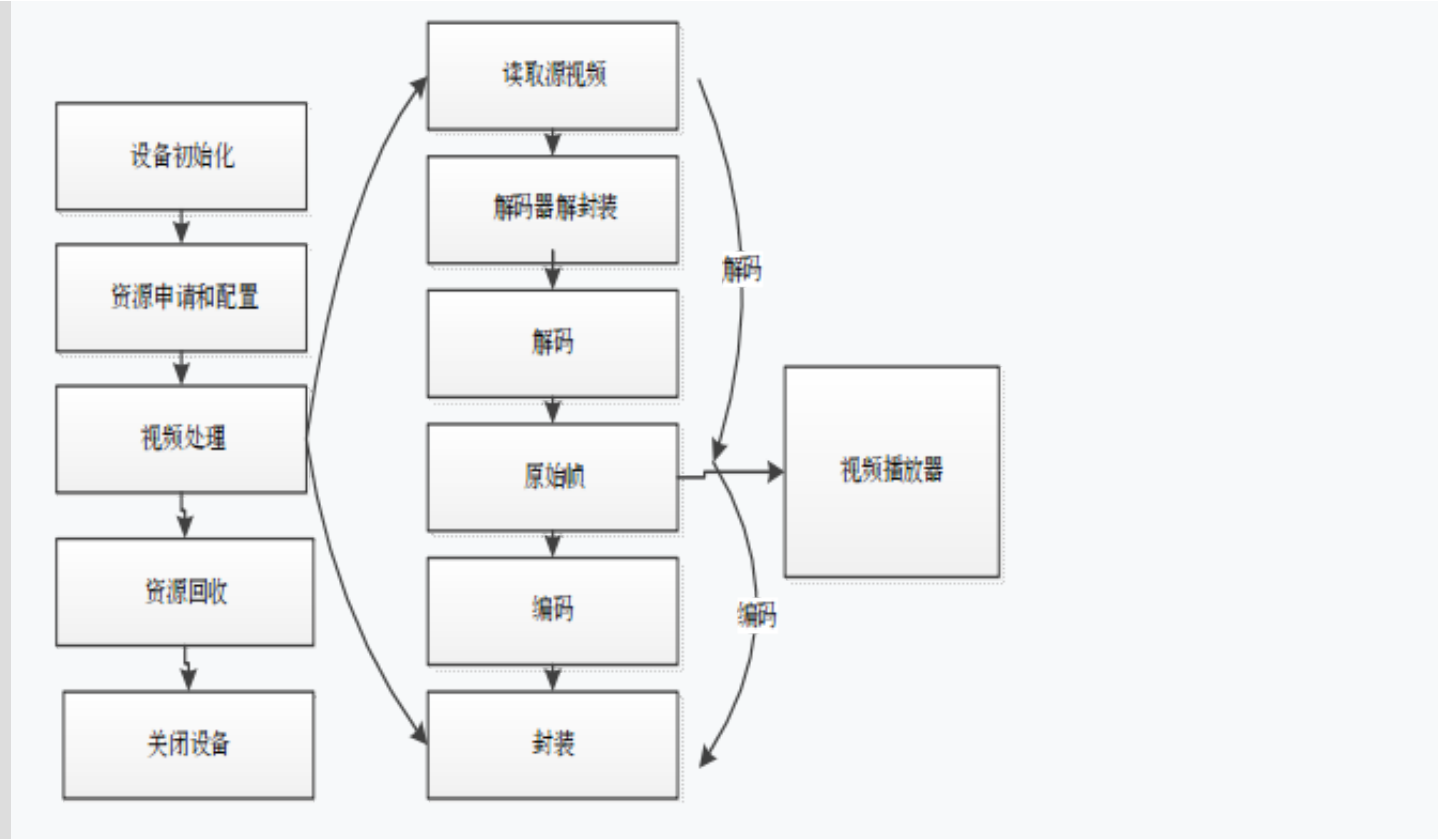
```
//整个设备开启自检过程已封成vdi_init()调用,而这个调用是应用层需要使用的,下面的代码是接口层的实现细节
int vdi_init(unsigned long coreIdx)
{
//打开前的自检状态判断等...
    pthread_mutexattr_t mutexattr;
    MUTEX_HANDLE mutex;
    vdi_info_t *vdi;
    vdi = &s_vdi_info[coreIdx];
//...还有很多判断...

//打开设备的底层调用
    vdi->vpu_fd = open(VPU_DEVICE_NAME, O_RDWR);
    if (vdi->vpu_fd < 0) {
        fprintf(stderr, "[VDI] Can't open vpu driver\n");
        return -1;
    }

//其它初始化,判断工作
    memset(&vdi->vpu_buffer_pool, 0x00, sizeof(vpudrv_buffer_pool_t)*MAX_VPU_BUFFER_POOL);
    vdi->vdb_register.size = VPU_BIT_REG_SIZE;
    vdi->vdb_register.virt_addr = (unsigned long)mmap(NULL, vdi->vdb_register.size, PROT_READ | PROT_WRITE, MAP_SHARED, vdi->vpu_fd, 0);

//....还有大量的判断初始化工作
    return 0;

}
```

4.2.2 资源的申请和配置

- 在上一个步骤过后,就需要对设备进行配置,这里的配置主要与视频相关,后面的解码或编码所需要的参数都是事先配置好的
- 内存资源的申请，相关数据结构的初始化，主要包括视频信息的数组结构，编解码器所需要的内存，编解码器的配置，内存的组织，线程的调度等一系列配置。只有这个过程正确完成，后面的视频处理才能进行。

4.2.3 视频处理（如图）

- 这个是主要的过程,在前面一切工作就绪后,就可以进行视频处理了
- 下图是视频解码后再编码的一个流程图, 大体分如下的几个主要步骤:



视频编解码流程图

- 具体如下:
  - 1.读取源视频
    这里可以是在本地读取,也可是从网络或其它来源.不同的用途实现起来不一样,一般是获取视频源后放到内存,等待解码
  - 2.解封装
    分析提取视频文件中的video流,因为视频中还可能有其它的流,比如音频,字幕等. 读取视频格式，分辨率,大小，码率等相关信息，存储到对应的数据结构中
  - 3.调用对应的解码器
    根据2中读取的数年信息,根据这些信息配置并调用对应的解码器,将相关信息存入对应的结构体中，保存下来.这里的解码器就是一个个算法的软件实现.可能对于一种解码有不同的算法来解决,这样对应不同的解码器.编码器也类似.
  - 4.对视频部分解码
    解码后,得到原始帧，到此完成了解码过程，原始帧可以根据不同的应用做处理,比如缩放,播放,识别,对比,去噪等过程都是在这儿进行
  - 5读入编码信息，设定编码器参数和格式
    这一步可能在前面的配置中设置好了，不用的实现方案中，这一步可能在编码之前就配置好,和解码器的配置类似
  - 6.调用编码器编码
    编码是解码的逆过程,和上面的处理大体类似
  - 7.封装相关数据，包括音频和文字
    这个也是解封闭的逆过程.根据不同的格式需要,封装成不同的视频
  - 8.结束
    这个就做一些收尾的工作.
- 这里仍以 `coda980` 为例,选取第四步,对视频部分解码,在应用层,必须封成一个功能模块,在 `coda980` 中就

是`ret = VPU_DecStartOneFrame(handle, &decParam);`,而这一步,是接口层所提供的调用,应用层只需要配置好参数decParam和设备号handle即可.

4.2.4 资源回收

- 这个过程与前面的对应,需要回收内存等资源,还要对最后一部分视频处理作判断.有时候,最后一部分视频可能不是完整的,不能完成一整帧的解码,这样也要回收处理
- 针对多路过其它应用,还有线程等资源的回收处理

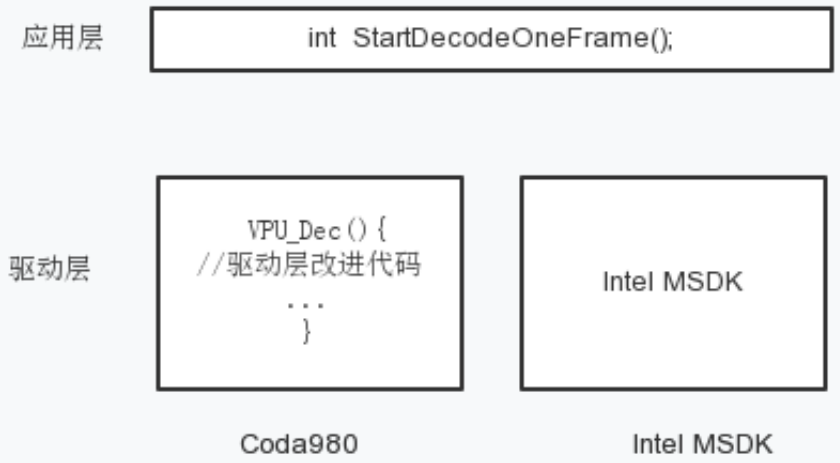
4.2.5 关闭设备

- 这个部分也比较重要,实际的功能可能对视频处理没那么重要.设备的正常关闭不同的地方要求各异.大体也是一系列流程化的操作.而最后会对关闭成功与否有一个返回值判断

4.3 实验方案

4.3.1 归纳总结应用层的编程框架

- 定义接口,返回值,参数等关键变量
  此处举例如下:
  在应用层,定义解码一帧的接口函数是
  `int StartDecodeOneFrame(int DeviceId, DecodeParam *parm);`,
  - 返回0表示解码成功,非零表示出现各类warning或error, 负值表示error, 正值表示warnings.
  - 参数表示设备号和解码配置.
  - 在`coda980`和`Intel MSDK`中的实现分别如下图



图四 接口定义示例与下层实现

1. 在上层统一了接口后,只需要实现一个类似编译器功能的转化,就可以在不同的设备上提供相同的功能
2. 针对不同的异构设备,需要有一个不同的转化过程
3. 带来的好处就是,应用层的代码,可在不同异构设备上使用, 开发人员可较少的关注底层

- 按功能划分不同的模块或函数
- 在项目中的三种异构设备上实现，得到主要评估数据

4.3.2 根据归纳总结出来的框架与原有的异构设备的demo进行对比

- 数据包括视频处理的速度，视频质量，码率，支持的格式（其中分别有解码格式和编码格式），可扩展性，多路的支持（是否支持或支持多少路），稳定性，可编程性（主要是基于此方案，程序员是否方便修改和调用）。
- 接口的对比和归纳：我们注意到的结果是视频处理应用层上的接口，如前面视频处理流程分析，接口包括：设备初始化接口，资源分配，设备配置，视频信息的组织和存储，视频头部的分析，编解码器的实现，编解码器的调用，解码过程，编码过程，封装过程，解封装等。

4.3.2 通过对三种异构设备提供的底层接口，定义应用层的接口，并实现其中的几种。

- 定义视频处理应用层的接口，定义的内容包括：接口的功能，参数，使用前提条件，调用后的结果
- 针对上面的各个功能接口，分别详细的列出对应的接口实现。

五．已有的科研基础和所需的科研条件

5.1 本人已有研究成果

5.1.1 `Intel HD Graphics` 编解码方案

这一套编解码方案是基于`Intel HD Graphics`集成显示核心。Intel HD Graphics是Intel一系列的集成显示核心。显示核心是集成于处理器上。我们使用过Intel HD Graphics 2500(6个运行单元），Intel Graphics HD 4000(16个运行单元）等型号，属于第三代集成显示核心【1】。Intel Media SDK【3】是提供给程序员开发使用的API,使用Intel Media SDK，我们完成了视频的编解码。可达到很高的处理速度。并且此编解码方案还应用到了其它需要处理视频的项目中，取得了很好的效果。

基于此项目成果，我们有一套完整且高效的视频编解码方案，这些成果可用于以后的研究和工程项目中。

5.1.2 `Godson-dm` 芯片，使用的是 `Coda980 Video codec IP`

`Coda980 Video codec IP`是`Chips&Media`【4】的视频处理方案，使用的异构设备是Godson-dm芯片。目前已基于此完成了多个相关项目，包括解码，编码，转码，多路解码等，达到了很好的效果。这套方案也是我们借鉴的重要依据。

5.1.3 `ffmpeg` 视频开源解决方案

Ffmpeg是一个开源免费跨平台的视频和音频方案【2】。目前基于此方案，我们添加了硬件编码器，可实现视频转码加速。Ffmpeg提供了一套完整的音视频解决方案，可用CPU编解码，也可调用特定的异构设备来完成视频加速，但需要底层的支持并编写对应的编解码器。Ffmpeg也有一套成熟的视频处理流程，由于开源的，业内已广泛使用，好多播放器，视频转码等都是基于此方案开发的。在项目中，我们也借用过ffmpeg的解决方案，取得了很好效果。这一项目成果，也可用于研究当中，值得今后借鉴。

图二　ffmpeg解码流程

5.2 现有成果整理

以上三种解决方案的相关数据，包括硬件要求，平台，环境，操作系统，编解码速度,
格式支持，视频质量相关等。
对应层次的接口规范和SDK调用情况，包括，相同功能的API接口对比，数据的组织结构特点，各编程框架自己特有的功能，视频编解码的处理流程等．

1. 视频编解码必要步骤和需要处理的过程

5.3总结归纳

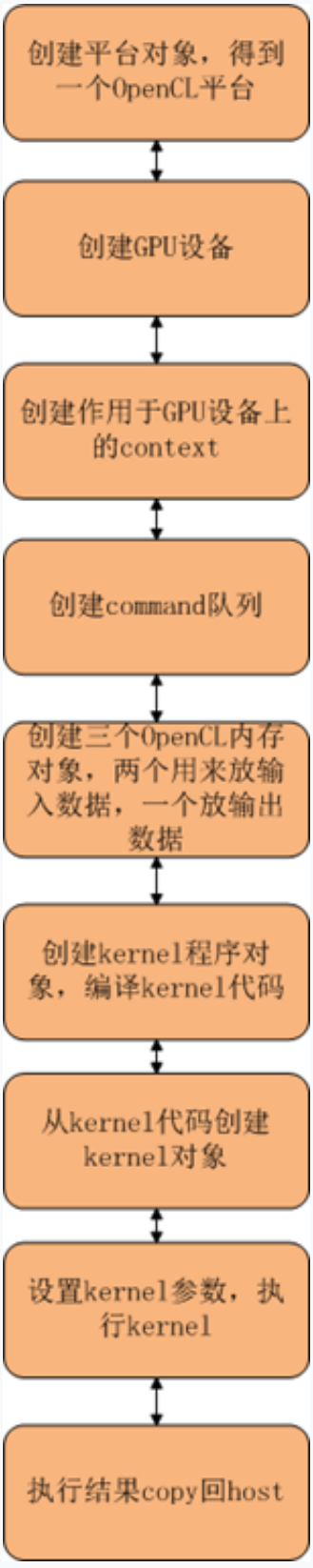
1 针对以上大约四种异构转码解决方案，分别总结出各种解决方案应用层的实现方法

1. 定义相关条件和接口要素，并在四种方案中找到对应的接口实现

5.4 后续研究

- 使用GPU来进行视频编解码，或基于CPU的GPU异构平台核并行编程模型的研究【7】
  GPU
- 使用`OpenCL`编程框架
  `OpenCL`是第一个面向异构系统通用目的并行编程的开放式、免费标准，也是一个统一的编程环境，其编程框架如图三，便于软件开发人员为高性能计算服务器、桌面计算系统、手持设备编写高效轻便的代码，而且广泛适用于多核心处理器(CPU)、图形处理器(GPU)、Cell类型架构以及数字信号处理器(DSP)等其他并行处理器，在游戏、娱乐、科研、医疗等各种领域都有广阔的发展前景。

- 调用`Nvidia`的`Cuda`接口来实现转码
  `CUDA`是一种由`NVIDIA`推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题。 它包含了CUDA指令集架构（ISA）以及GPU内部的并行计算引擎。 开发人员现在可以使用C语言来为CUDA™架构编写程序，C语言是应用最广泛的一种高级编程语言。所编写出的程序于是就可以在支持CUDA的处理器上以超高性能运行。
  其它异构视频解决方案，并比较异同。
  研究其它异构视频处理设备
  丰富和完善编程接口



图三　OpenCL编程框架

六．工作计划和进度安排

6.1 时间节点

- 1．2014.10 开题
- 2．2014.12 中期答辩
- 3．2015.05 答辩
- 4．2015.06 正常毕业

6.2 计划安排

- 从开题到中期阶段
    - 总结归纳已有成果，并收集得到相关数据。
    - 调研最新技术，大体有论文雏形。
    - 完成之前项目中涉及到的研究点，丰富研究的内容，并完成相关实验，得到数据。
    - 实现前面提到的三种视频处理方案
- 中期答辩
    - 准备中期答辩，内容有：
    - 前期项目中的成果，总结和提出来的编程框架
    - 得到的数据分析和对比，此编程框架的基本模式。
- 从中期到答辩
    - 根据已有的结论成果，应用于其它异构平台。
    - 主要是GPU+OpenCl编程框架中
    - 并研究细节问题，丰富论文内容。
- 毕业答辩
    - 整理研究成果
    - 制作图表和相关材料
    - 撰写论文，查阅文献，修改论文。

七．参考文献

- 【1】视频编解码技术的应用和发展 庞涛 《科技成果纵横》 2008年04期
- 【2】`http://www.intel.cn/content/www/cn/zh/homepage.html`
- 【3】`http://www.ffmpeg.org/`
- 【4】`https://software.intel.com/en-us/vcsource/tools/media-sdk`
- 【5】`http://www.chipsnmedia.com/`
- 【6】基于GPU的高性能并行算法研究 吉林大学博士学位论文 白洪涛 2010.06
- 【7】基于CPU_GPU异构平台的性能优化及多核并行编程模型的研究 中国科学技术大学硕士学位论文 陈波 2011.04.18
- 【8】基于通用可编程GPU的视频编解码器*架构*算法与实现 浙江大学硕士学位论文 2005.11

八．需要制作的图表

（论文中需要用到的并制作的）

8.1 图:

- 图1： 视频处理层次图（从底层硬件一直到上层应用）
- 图2： 应用层视频处理逻辑层
- 图3-5: 针对具体的某一个实现，比如 `coda980,Media SDK,ffmpeg` 的实现，这三个图，需要根据图2的框架，具体到接口和定义上（重点）
- 图6: `GPU+OpenCL` 编程框架，需要按照本论文要点作图

8.2 表

- 表1：主流异构视频转码方案和应用场景，平台等
- 表2：三种已有实现方案的对比，主要是接口的对比，以图3-5为依据
- 表3：三种已有实现方案数据的对比，主要是针对影响视频的关键要素

8.3 文献中引用的图表