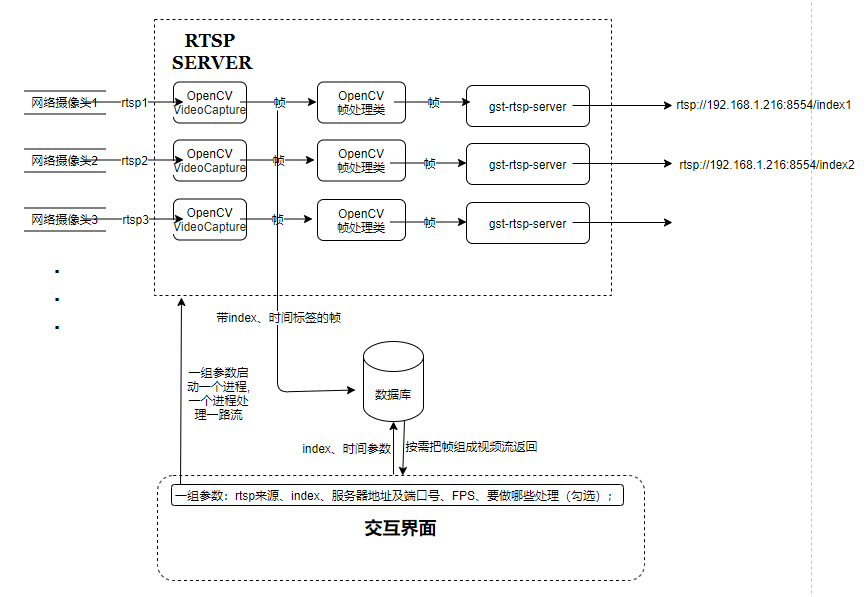
# RTSP流媒体服务器

## 实验简介

本实验拟通过利用cuda + opencv + gstreamer + gst-rtsp-server等工具交叉编译，搭建GPU驱动的多路RTSP流媒体服务器。通过管线实现对来自海康威视网络IP摄像头的RTSP视频流进行解码、处理、编码、组流、推流等；

整个RTSP流媒体服务器的系统数据流图如下图所示：



系统数据流图

系统的输入是多路来自网络摄像头的rtsp流：rtsp1、rtsp2、rtsp3…，对每一路流进行解码、处理、存储，再利用gst-rtsp-server将处理后的帧数据推送至服务器指定的端口和index（与输入rtsp ID相对应）。最后，每一路rtsp流发送到唯一的URL地址，客户端可以在rtsp://127.0.0.1:8554/index/0等与输入流对应的URL地址获取rtsp流。

其中，每一路视频流的管道及参数输入如下图所示：

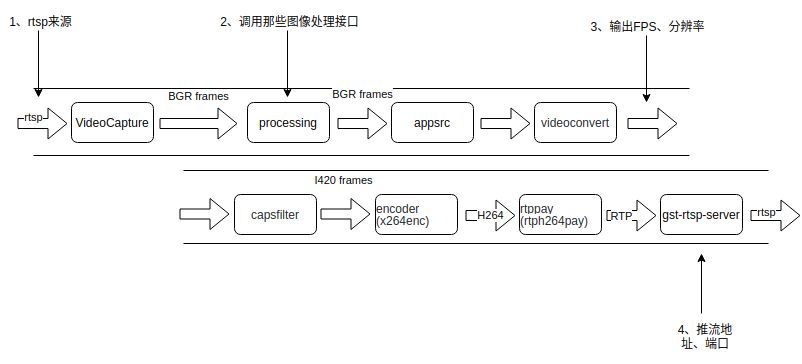


图 系统pipeline

每一路流都需要通过一组参数来配置，通过定义一个结构体来存储所有输入参数：

typedef struct

{

string INDEX; //rtsp标号

string in\_rtsp; //来源uri

int out\_width; //组流分辨率

int out\_height;

int out\_fps; //组流帧率

string out\_port; //推流端口

} Params;

通过参数设置，在需要帧数据的时候从互斥线程锁中获取拉流线程当前帧；同时在组流+推流线程中做图像处理，对组流需要的帧作处理。每一个管道（即每一路rtsp流）需要启动一个进程来管理，一个进程包括两个线程；两个线程之间通过一个全局的静态的线程互斥锁来传递帧数据。

1.拉流线程：利用opencv videocapture 源源不断地接受每一帧数据；

2.处理+组流+推流线程；

## 实验内容

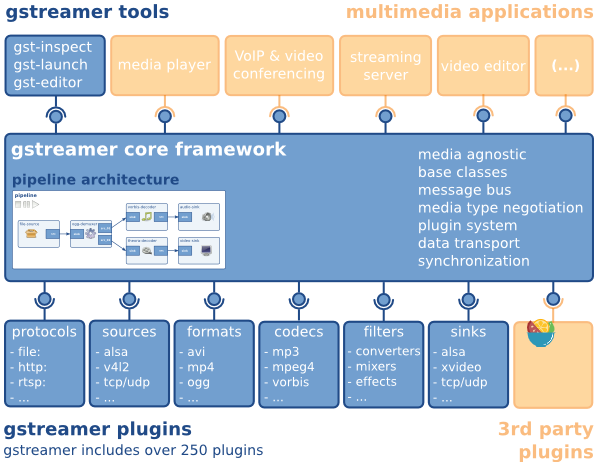
1. 利用opencv videoCapture实现RTSP解码；
2. 实现对解码出的cv::Mat帧进行处理，如区域选择；
3. 实现对解码出的cv::Mat帧进行本地存储；
4. 基于Gstreamer的gst-rtsp-server套件实现对Mat帧进行H.264编码、组流、推流至服务器指指定url（端口、index）；
5. 利用互斥线程锁将解码、处理、编码、组流等步骤封装成管线，每一路管线对应一路RTSP流；
6. 利用多进程管理实现同时处理多路流；
7. 交叉编译opencv和cuda，使用nvidia video codec sdk加速RTSP解码，解放CPU资源、降低延迟；
8. 通过init配置文件启动系统，其中包括rtsp路数以及每一路流各自的参数；

## 实验原理

### Gstreamer

Gstreamer是一个支持Windows，Linux，Android， iOS的跨平台的多媒体框架，应用程序可以通过管道（Pipeline）的方式，将多媒体处理的各个步骤串联起来，达到预期的效果。每个步骤通过元素（Element）基于GObject对象系统通过插件（plugins）的方式实现，方便了各项功能的扩展。

下图是对基于Gstreamer框架的应用的简单分层：



**Media Applications**

最上面一层为应用，比如gstreamer自带的一些工具（gst-launch，gst-inspect等），以及基于gstreamer封装的库（gst-player，gst-rtsp-server，gst-editing-services等)根据不同场景实现的应用。

**Core Framework**

中间一层为Core Framework，主要提供：

* 上层应用所需接口
* Plugin的框架
* Pipline的框架
* 数据在各个Element间的传输及处理机制
* 多个媒体流（Streaming）间的同步（比如音视频同步）
* 其他各种所需的工具库

**Plugins**

最下层为各种插件，实现具体的数据处理及音视频输出，应用不需要关注插件的细节，会由Core Framework层负责插件的加载及管理。主要分类为：

* Protocols：负责各种协议的处理，file，http，rtsp等。
* Sources：负责数据源的处理，alsa，v4l2，tcp/udp等。
* Formats：负责媒体容器的处理，avi，mp4，ogg等。
* Codecs：负责媒体的编解码，mp3，vorbis等。
* Filters：负责媒体流的处理，converters，mixers，effects等。
* Sinks：负责媒体流输出到指定设备或目的地，alsa，xvideo，tcp/udp等。
* Gstreamer框架根据各个模块的成熟度以及所使用的开源协议，将core及plugins置于不同的源码包中：
* gstreamer: 包含core framework及core elements。
* gst-plugins-base: gstreamer应用所需的必要插件。
* gst-plugins-good: 高质量的采用LGPL授权的插件。
* gst-plugins-ugly: 高质量，但使用了GPL等其他授权方式的库的插件，比如使用GPL的x264，x265。
* gst-plugins-bad: 质量有待提高的插件，成熟后可以移到good插件列表中。
* gst-libav: 对libav封装，使其能在gstreamer框架中使用。

### Gst-rtsp-server

Gst-rtsp-server是基于GStreamer之上用于构建rtsp服务器的库，由C/C++语言编写。可以简便实现编码、推流。

### NVIDIA Video Codec SDK

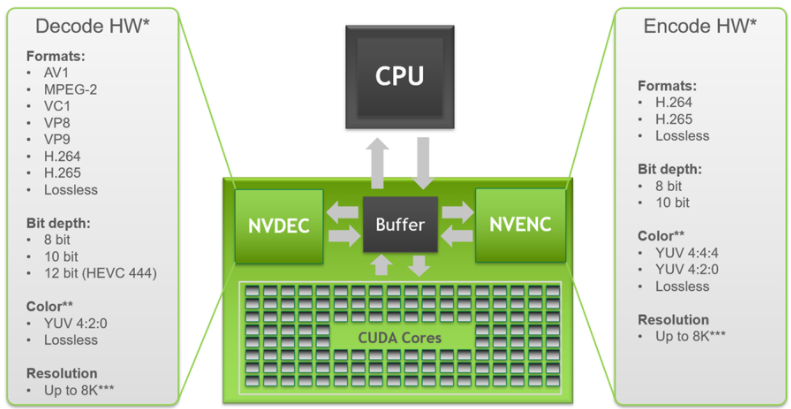
NVIDIA Video Codec SDK一套全面的API，包括用于Windows和Linux上硬件加速视频编码和解码的高性能工具，示例和文档。

SDK包含两个硬件加速接口：

* NVENCODE API用于视频编码加速
* NVDECODE API用于视频解码加速（以前称为NVCUVID API）

NVIDIA GPU包含一个或多个基于硬件的解码器和编码器（与CUDA内核分开），可为多种流行的编解码器提供完全加速的基于硬件的视频解码和编码。卸载解码/编码后，图形引擎和CPU可以自由进行其他操作。

GPU硬件加速器引擎用于视频解码的（被称为NVDEC）和视频编码（称为NVENC）支持比实时视频处理，这使得它们适合于被用于代码转换应用程序，除了视频播放速度更快。下图表示对NVIDIA Ampere GPU系列的支持.硬件本身不支持4：2：2.支持取决于编解码器。



## 实验环境

操作系统Ubuntu18.04

显卡：1080Ti

显卡驱动：NVIDIA-DRIVER-450.80.02

编程语言：C/C++

CUDA10.1、cuDNN7.6.3、nvidia video codec sdk 9.1.23、nv-codec-headers9.1、Opencv4.2、Gstreamer1.0、gst-rtsp-server等

## 实验方法

### 环境配置、编译

### 实验程序

代码已上传至：<https://github.com/chicleee/rtsp-server>

代码中附有详细注释。

git clone <https://github.com/chicleee/rtsp-server>

cd rtsp-server

cd src

其中，在源码src文件夹下有以下重要文件：

* Test\_gpu文件夹是测试交叉编译环境是否配置正确的代码，直接

# 编译

Make

# 使用rtsp流进行测试

./main.out rtsp://admin:kuangping108/192.168.1.64…

# 或者使用本地视频文件进行测试

./main.out test.h264

* Start.cpp是只使用CPU的版本

编译命令：

g++ -std=c++11 start.cpp -o start -lboost\_system -lboost\_filesystem -lpthread -lgstapp-1.0 `pkg-config --libs --cflags opencv gstreamer-1.0 gstreamer-rtsp-server-1.0`

运行命令：

./start

* start\_gpu.cpp是GPU版本

编译命令：

g++ -std=c++11 start\_gpu.cpp -g -o start\_gpu -lboost\_system -lboost\_filesystem -lpthread -lgstapp-1.0 `pkg-config --libs --cflags opencv4 gstreamer-1.0 gstreamer-rtsp-server-1.0` \

-I/usr/local/opencv4/include/opencv4/opencv2 \

-I/usr/local/cuda/include \

-L/usr/local/cuda/lib64 \

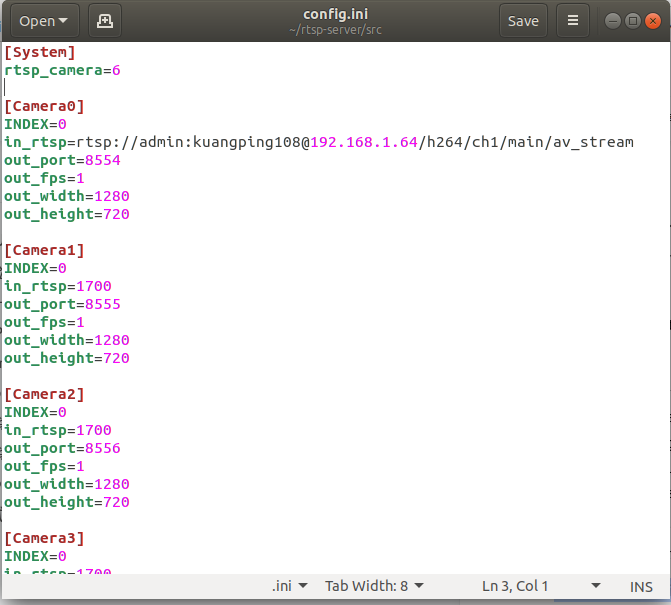
-I/usr/include/eigen3 \

-L/usr/lib/x86\_64-linux-gnu -lcuda -ldl -lnvcuvid

运行命令：

./start\_gpu

* Config.ini是系统的配置文件，包含输入参数如下图所示：



## 实验结果

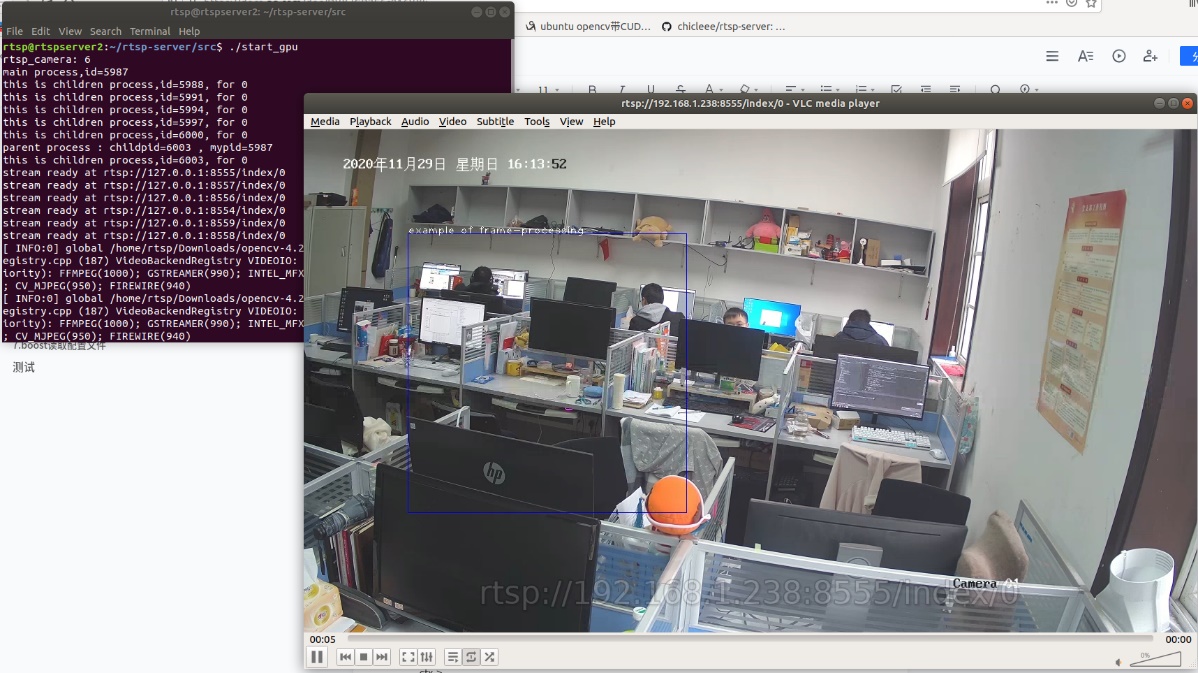
测试运行结果如下。在服务器src文件夹下编辑好config.ini配置文件后，打开终端通过输入命令如下所示：

./start\_gpu

启动程序，客户端在指定端口号和INDEX的URL地址：

<http://192.168.1.238:8554/index/0>

就可以获取我们推送的处理后的流，如下图所示，接入6路25FPS的4K RTSP流，我们的流媒体服务器将他们推送至指定的端口和地址，用VLC播放器演示客户端接取的RTSP流（1FPS、1080P、带检测实例框的视频流）：

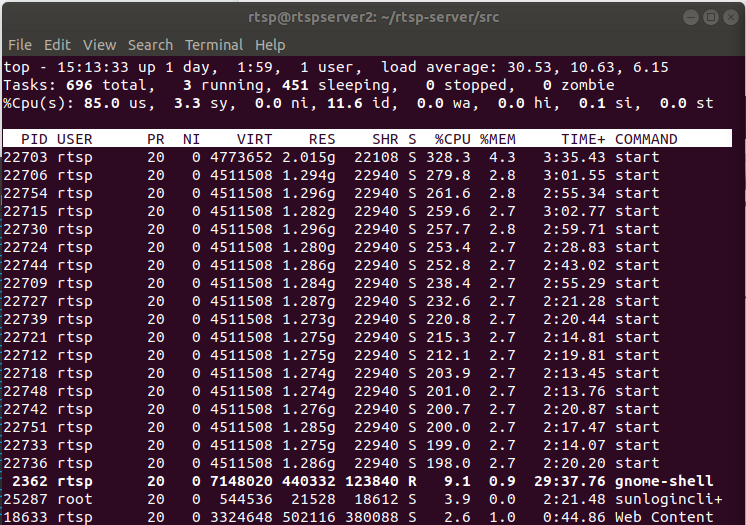


此外，对RTSP流媒体服务器的性能经行评估和测试如下：

下面是通过CPU版本的流媒体服务器推流时的使用top命令查看CPU负载情况的测试：

./start

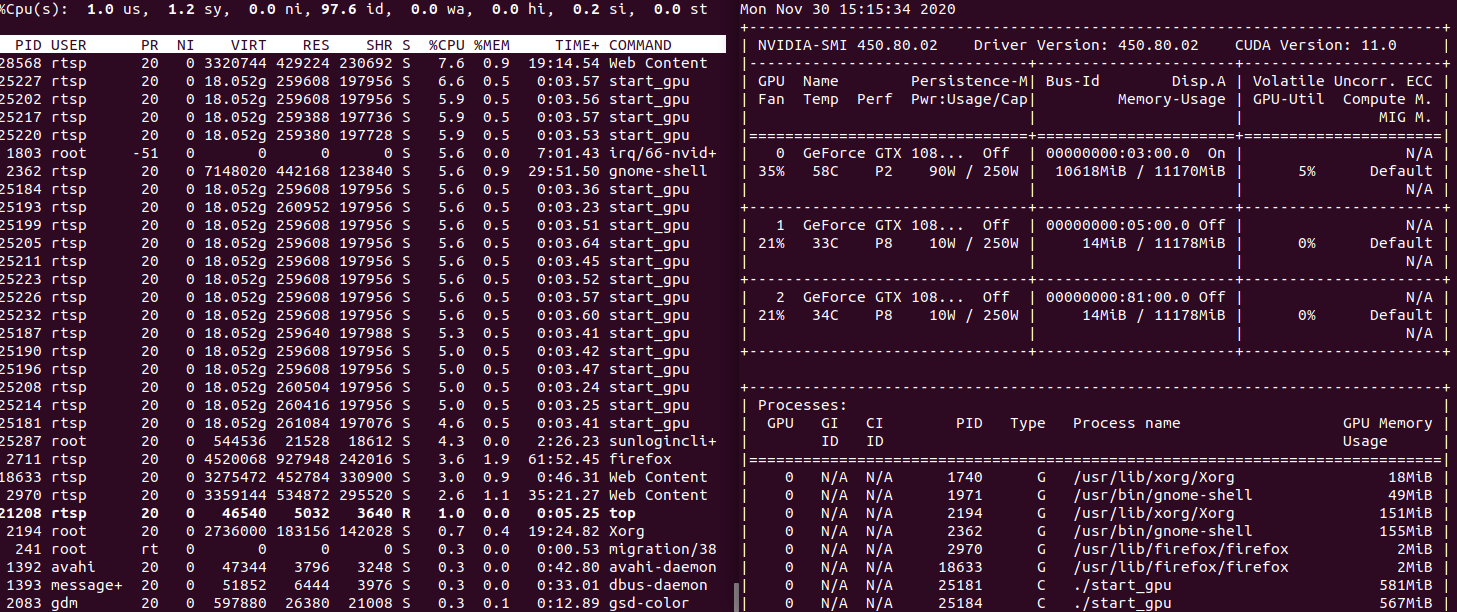
虽然在238服务器上可以使用CPU同时运行32路管线，但会严重超频。启动CPU版本的RTSP流媒体服务器同时处理18路流时，CPU的负载情况，CPU严重超频：



下面是运行命令

./start\_gpu

用GPU解码的RTSP流媒体服务器同时处理18路流时CPU的负载情况以及GPU显存消耗情况：



可以看到，在使用GPU解码的系统极大地解放了CPU资源，从每一路RTSP流在每一个进程里需要300%左右降到了6%左右。

而于此同时，释放CPU的代价就是，在声明nvidia提供的读取流接口时就会占用来580M显存，即使GPU使用率不高（每路4K RTSP流）。经过测试，该解码接口对显存的占用只受输入流的分辨率大小影响。

## 结论

本RTSP流媒体服务器实现了前期预期的各项功能，但由于基于NVIDIA视频解码接口调用GPU解码每一路4K的RTSP视频流都需要占用约580M的显存；解码一路1080P的视频流需要约200M的显存。故在单张11GB显存的1080Ti显卡上，最多只能同时处理18路4K流、或者50路1080P流。

由于目前的应用需求对路数要求高以及预期只使用单张11GB显存的显卡，此RTSP流媒体服务器暂搁置。