



# 登临 GPU 适配 CUDA 11 开源代码 操作指导

DL-DG/SW-050A-02

2024-01-04

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

## 商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

## 通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼栋5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

## 更新历史

版本	更新描述
02	适配CUDA 11。
01	第一次发布。

# 内容目录

更新历史

内容目录

## 1 登临 Hamming V2 SDK CUDA 编程

### 1.1 登临 Hamming V2 SDK 文档

### 1.2 登临 SDK 与 NVIDIA CUDA 11.7 的差异

## 2 CUDA 开源代码 登临GPU适配 工作流程

总体流程概述

### 2.1 A - Requirement

#### 2.1.1 定义

#### 2.1.2 步骤

### 2.2 B - Dependency Preparation

#### 2.2.1 定义

#### 2.2.2 步骤

#### 2.2.3 Checklist

### 2.3 C - Source Build & Sample Running

#### 2.3.1 定义

#### 2.3.2 步骤

#### 2.3.3 Checklist

### 2.4 D - Algorithm Application Unit Test & Debug

#### 2.4.1 定义

#### 2.4.2 步骤

#### 2.4.3 CheckList

## 3 示例：cuOSD 登临卡适配

### 3.1 A - Requirement

#### A1 - Product Background

#### A2 - Algorithm Application Requirement

### 3.2 B - Dependency Preparation

#### B1 - Hamming V2 SDK & NVIDIA CUDA 11.7 Setup

#### B2 - Third Party Setup

#### B3 - Hamming V2 SDK Document Study

### 3.3 C - Source Compilation & Sample Running

#### C1 - Download Source & Study

#### C2 - NVCC Build & Sample Running

#### C3 - DLCC Build & Sample Running

### 3.4 D - Algorithm Application Unit Test & Debug

#### D1 - Unit Test Design for Algorithm Application

#### D2 - Unit Test & Debugging

#### D3 - Test Report

FAQS

1 DL卡挂起 ( Hang ) 了，怎么处理？

2 如何对 CUDA C 源码编译输出汇编文件？

# 1 登临 Hamming V2 SDK CUDA 编程

## 1.1 登临 Hamming V2 SDK 文档

文档名称	文档描述	NVIDIA CUDA 11.7 对应文档
《Denglin Hamming V2 dICU Programming Guide》	登临第二代芯片 CUDA 编程指南	<a href="#">《NVIDIA CUDA C Programming Guide》</a>
《Denglin Hamming V2 dICU API》	登临第二代芯片 CUDA Runtime & Driver API 使用说明	<a href="#">《NVIDIA CUDA Runtime API》</a> <a href="#">《NVIDIA CUDA Driver API》</a>
《Denglin Hamming V2 Texture Feature Guide》	登临第二代芯片 CUDA Texture 使用说明	<a href="#">《NVIDIA CUDA C Programming Guide》</a> <a href="#">3.2.12. Texture and Surface Memory</a>
《Denglin Compute Compiler V2 User Guide》	登临第二代芯片 CUDA 编译器使用说明	<a href="#">《NVIDIA CUDA Compiler Driver NVCC》</a>
《登临GPU适配 CUDA 11 开源代码操作指导》	CUDA 开源代码 登临GPU 适配 参考文档	-
《Hamming V2 SDK 与 CUDA 11.7 差异说明》	Hamming V2 SDK 与 CUDA 11.7 差异项的说明	-

## 1.2 登临 SDK 与 NVIDIA CUDA 11.7 的差异

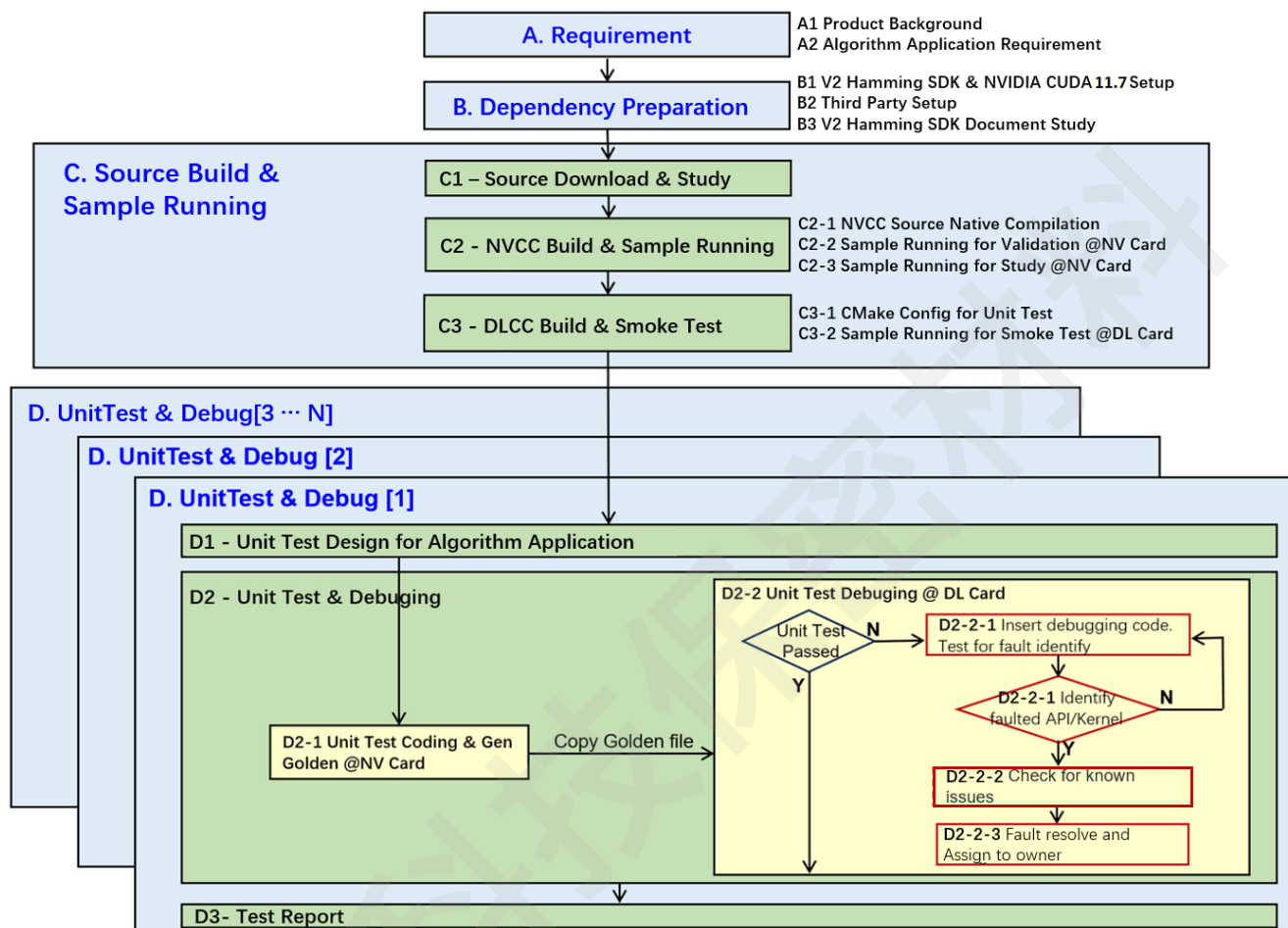
具体差异请参考《Hamming V2 SDK 与 CUDA11.7 差异说明》，该文档对以下差异项进行了说明：

- CUDA Driver API
- CUDA Runtime API
- Compiler 编译参数
- CUDA Intrinsics
- Math 函数精度
- 其他差异项

## 2 CUDA 开源代码 登临GPU适配 工作流程

### 总体流程概述

总体流程如下图所示：



### 2.1 A - Requirement

#### 2.1.1 定义

根据产品应用，明确算法开源代码的应用需求。

#### 2.1.2 步骤

##### A1 - Product Background

- 了解产品应用背景包括：产品功能、软件Pipeline、登临 GPU 型号等。

##### A2 - Algorithm Application Requirement

- 明确算法应用需求包括：算法功能、规格、算法调用方式的要求等，并与算法产品应用团队对齐。
- 算法应用需求作为单元测试（Unit Test）设计的输入。

## 2.2 B - Dependency Preparation

### 2.2.1 定义

前置依赖项的准备，包括：

- 软件依赖环境准备。
- 学习登临 Hamming SDK 文档，了解其与 NVIDIA CUDA 的差异。

### 2.2.2 步骤

#### B1 - Hamming V2 SDK & NVIDIA CUDA 11.7 Setup

- 在开发主机上安装 NVIDIA 卡，然后安装 CUDA 11.7。
- 在开发主机上安装 Denglin GPU 卡，然后安装登临 Hamming SDK，安装方法请参考《登临 Hamming V2 SDK 安装指南》。

#### B2 - Third Party Setup

- 安装第三方依赖库，如 OpenCV、GoogleTest 等。
- 检查软件环境是否满足要求。

#### B3 - Hamming V2 SDK Document Study

- 阅读 章节 1.1 登临编程文档 中列出的文档。
- 阅读差异说明文档《Hamming V2 SDK 与 CUDA 11.7 差异说明》。

### 2.2.3 Checklist

编号	检查项	说明
BC-1	Linux 系统、CUDA、GCC、Hamming SDK、第三方环境/软件等都满足版本要求。	可使用 Shell 脚本来打印和检查。
BC-2	开发工程师已知晓 Hamming SDK 与 CUDA 11.7 的差异。	-

## 2.3 C - Source Build & Sample Running

### 2.3.1 定义

开源代码构建和测试环境准备，为下一步进入单元测试做好准备，具体包括：

1. 源码下载 & 构建
2. 源码 Sample 运行 & 学习
3. 单元测试环境搭建 & 冒烟测试 (Smoke Test)

## 2.3.2 步骤

### C1 - Source Download & Study

- 下载开源代码，了解整个源码框架和关键数据结构。

### C2 - NVCC Build & Sample Running

- C2-1 NVCC Source Native Compilation
  - 使用源码自带构建文件（makefile/cmake），进行编译和构建。
- C2-2 Sample Running for Validation @NV Card
  - 在NVIDIA卡上运行与 A2 Algorithm Application Requirement 相关的 Sample，验证算法效果。
- C2-3 Sample Running for Study @NV Card
  - 通过样例运行（Sample Running）、断点调试、关键参数调整等，学习算法实现流程。

### C3 - DLCC Build & Sample Running

- C3-1 Build Cmake Config for Unit Test
  - 参考《Denglin Compute Compiler V2 User Guide》文档内容，配置 Cmake 的编译相关参数。
  - 为单元测试配置 Cmake。
- C3-2 Sample Running for Smoke Test @DL Card
  - DLCC Build & 样例冒烟测试（Sample Smoke Test），确认是否可以进入单元测试。

## 2.3.3 Checklist

编号	检查项	说明
CC-1	DLCC Cmake 配置完成后，根据《V2 Hamming SDK 与 CUDA 11.7 差异说明》来检查确认编译选项是否正确。	-

## 2.4 D - Algorithm Application Unit Test & Debug

### 2.4.1 定义

根据算法应用需求，进行单元测试与调试，包括：

1. 暴露适配问题（Fault）
2. Bug 解决和分发（Bug Resolve and Assign）
3. 生成测试报告（Test Report）

### 2.4.2 步骤

#### D1 - Unit Test Design for Algorithm Application

- 根据 A2 Algorithm Application Requirement，设计单元测试用例。
- 与算法产品应用团队进行评审确认。

#### D2 - Unit Test & Debugging

- D2-1 Unit Test Coding & Gen Golden @ NV Card



- 单元测试用例编码实现，对于源码中已实现的单元测试，可直接复用。
- 在NVIDIA的卡上运行，生成登临卡单元测试所需的 Golden 与 Perf 参考数据。
- **D2-2 Unit Test Debugging @ DL Card**
  - D2-2-1 Unit Test & Identify Faulted API/Kernel
    - 在登临卡上运行单元测试。若运行失败，通过调试找到 Fault API/Kernel。
    - 先定位和解决一致性问题（Comformance Fault），再解决性能问题（Performance Fault）。
  - D2-2-2 Check for Known Issues
    - 对照《Hamming V2 SDK 与 CUDA 11.7 差异说明》文档内容，确认该问题是否为已知差异。
    - 查看已知 Bug 清单，确认该问题是否已经在 Bug 列表中。
  - D2-2-3 Fault Resolve and Assign to Owner
    - 对源码进行调整，通过绕开的方式来解决。
    - 问题转为 Bug，并按照 Bug 提交流程，提交到 Bug 列表。

### D3 - Test Report

- 生成单元测试报告，并提交评审以确认是否满足应用要求。

#### 2.4.3 CheckList

编号	检查项	说明
DC-1	DLCC编译器更新后，使用 <code>dlcc --version</code> 命令检查是否升级成功。	-

## 3 示例：cuOSD 登临卡适配

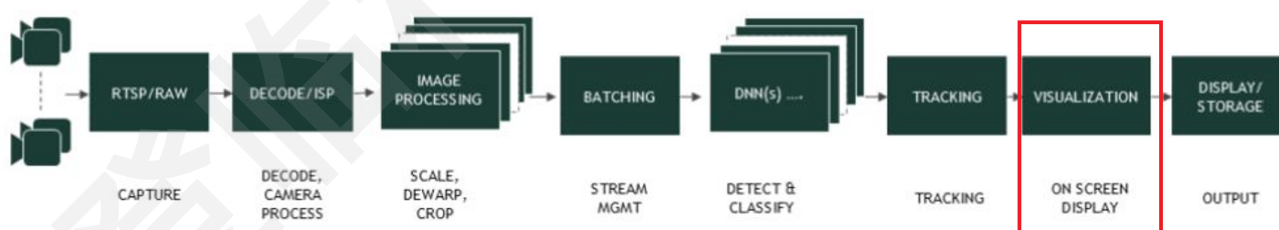
### 3.1 A - Requirement

#### A1 - Product Background

OSD 即 On-Screen Display，cuOSD 功能如下图所示：



OSD 在推理产品应用 Pipeline 中的位置如下图所示：



目前 OSD 在 CPU 上运行高帧率大图像应用场景，会导致大量数据在 CPU 与 GPU 之间来回拷贝，影响 Pipeline 整体效率。

因此，将 OSD 算法不放在 CPU 而是放在 GPU 上运行，可消除 CPU 与 GPU 之间冗余数据来回拷贝，提升 Pipeline 整体效率。

## A2 - Algorithm Application Requirement

- OSD 算法放在 GPU 上运行，提升 Pipeline 整体效率。
- cuOSD 是 NVIDIA 开源的 OSD 算法的 CUDA 实现，开发需求是把 cuOSD 适配到登临 GPU。

## 3.2 B - Dependency Preparation

### B1 - Hamming V2 SDK & NVIDIA CUDA 11.7 Setup

- 在开发主机上安装 NVIDIA 卡与 CUDA 11.7 SDK，使用 `nvcc --version check` 命令检查是否安装成功。
- 在开发主机上安装登临 GPU 卡，请参考《登临 Hamming V2 SDK安装指南》提供的方法安装 GPU 驱动与 Denglin Hamming V2 SDK。

### B2 - Third Party Setup

- 安装 cuOSD 适配开发所需要的第三方依赖库，包括：
  - Google Test 1.7.0
  - OpenCV 4.2.0
- 检查软件环境是否满足要求：
  - `source sdk_dir/env.sh`，配置SDK环境变量。
  - 执行 `print_env.sh` 脚本来检查，Shell 脚本参考代码如下：

```
# bash
# Usage:
# "./print_env.sh" - prints to stdout
# "./print_env.sh > env.txt" - prints to file "env.txt"
print_env() {
  echo "***OS Information***"
  cat /etc/*-release
  uname -a
  echo

  echo "***nVidia GPU Information***"
  nvidia-smi
  echo

  echo "***Denglin GPU Information***"
  sudo cat /proc/driver/denglin0/chip_info
  echo

  echo "***CPU***"
  lscpu
  echo

  echo "***CMake***"
  which cmake && cmake --version
  echo

  echo "***g++***"
  which g++ && g++ --version
```

```

echo

echo "***nvcc***"
which nvcc && nvcc --version
echo

echo "***dlcc***"
which dlcc && dlcc --version
echo

echo "***python***"
which python && python -c "import sys; print('Python {0}.{1}.
{2}'.format(sys.version_info[0], sys.version_info[1], sys.version_info[2]))"
echo

echo "***Environment Variables***"
printf '%-32s: %s\n' PATH $PATH
printf '%-32s: %s\n' LD_LIBRARY_PATH $LD_LIBRARY_PATH
}

echo "      ----- Start print_env -----"
echo "      "
print_env | while read -r line; do
    echo "      $line"
done
echo "      ----- End of print_env -----"

```

- 进入 Hamming SDK 的 sample/cuda 目录，编译并运行如下2个 Sample，并检查运行结果。
  - simpleStreams
  - vectorAdd

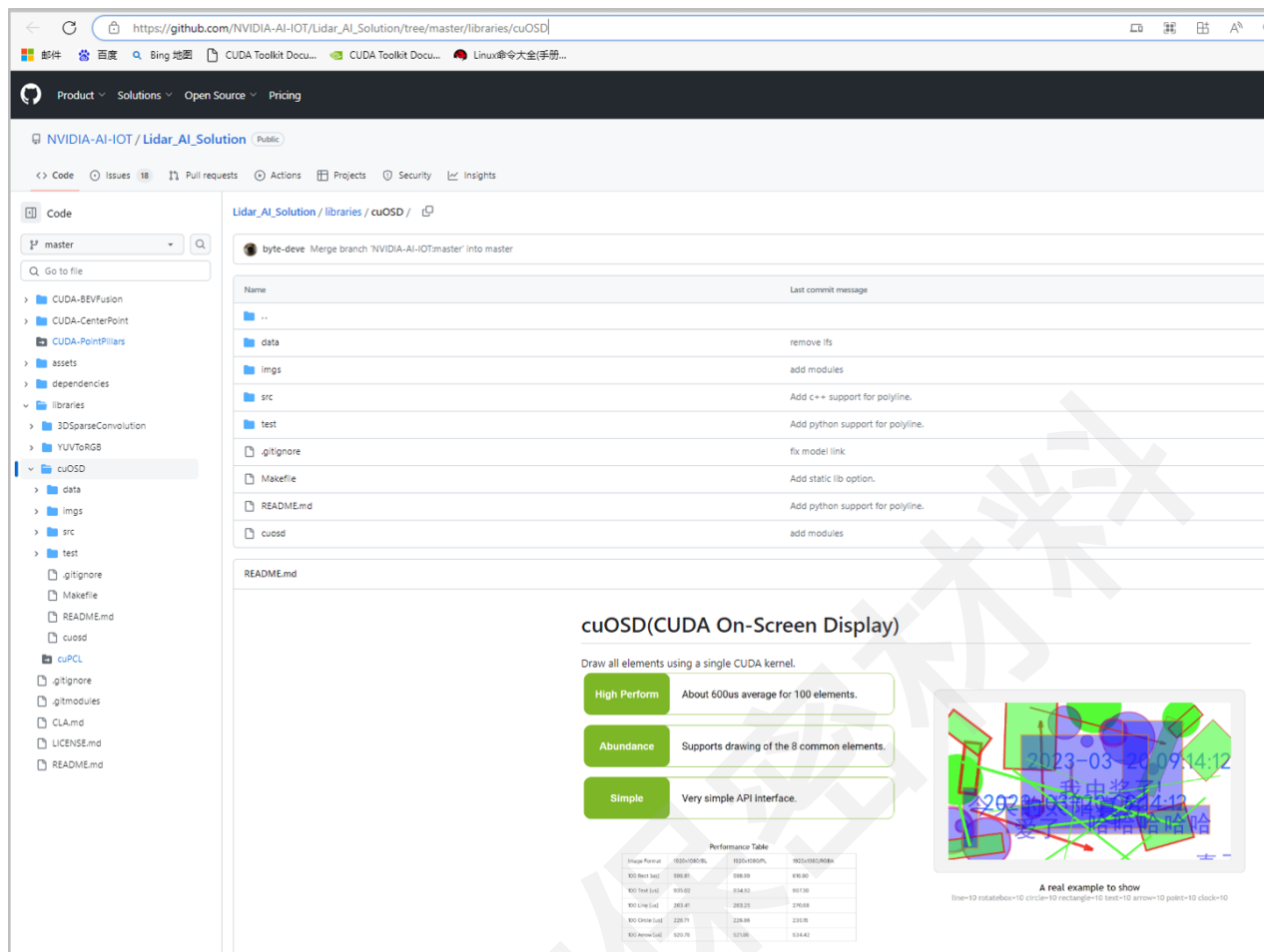
### B3 - Hamming V2 SDK Document Study

- 阅读 **章节1.1 登临 Hamming V2 SDK文档** 中列出的参考文档。
- 打印《**V2 Hamming SDK 与 CUDA 11.7 差异说明**》文档，方便查看和翻阅。
- 阅读《**V2 Hamming SDK 与 CUDA 11.7 差异说明**》文档，知晓 V2 Hamming SDK 与 NVIDIA CUDA 11.7 的差异项。

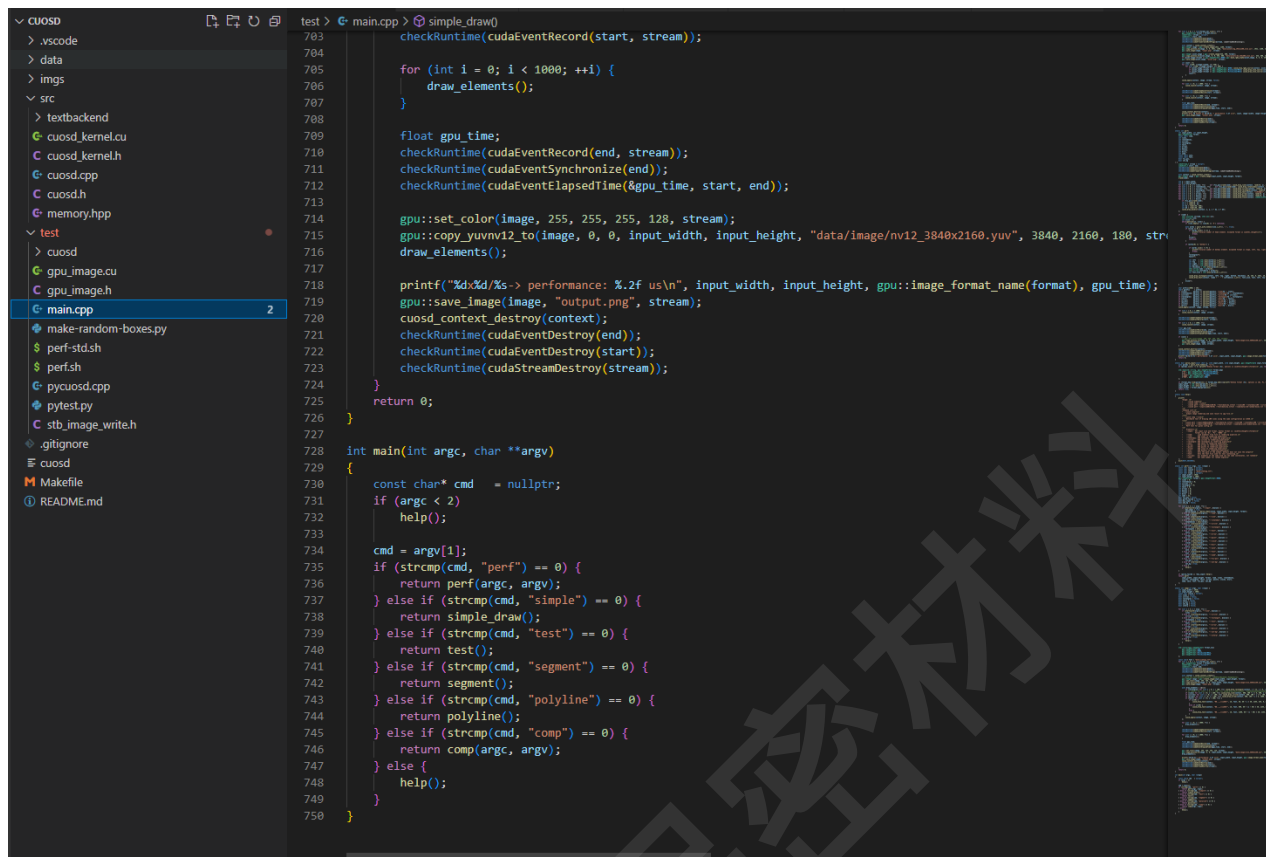
## 3.3 C - Source Compilation & Sample Running

### C1 - Download Source & Study

- 下载 cuOSD 开源代码，阅读源码框架与关键数据结构。
- [点击下载开源代码](#)，打开后下载页面如下图所示：



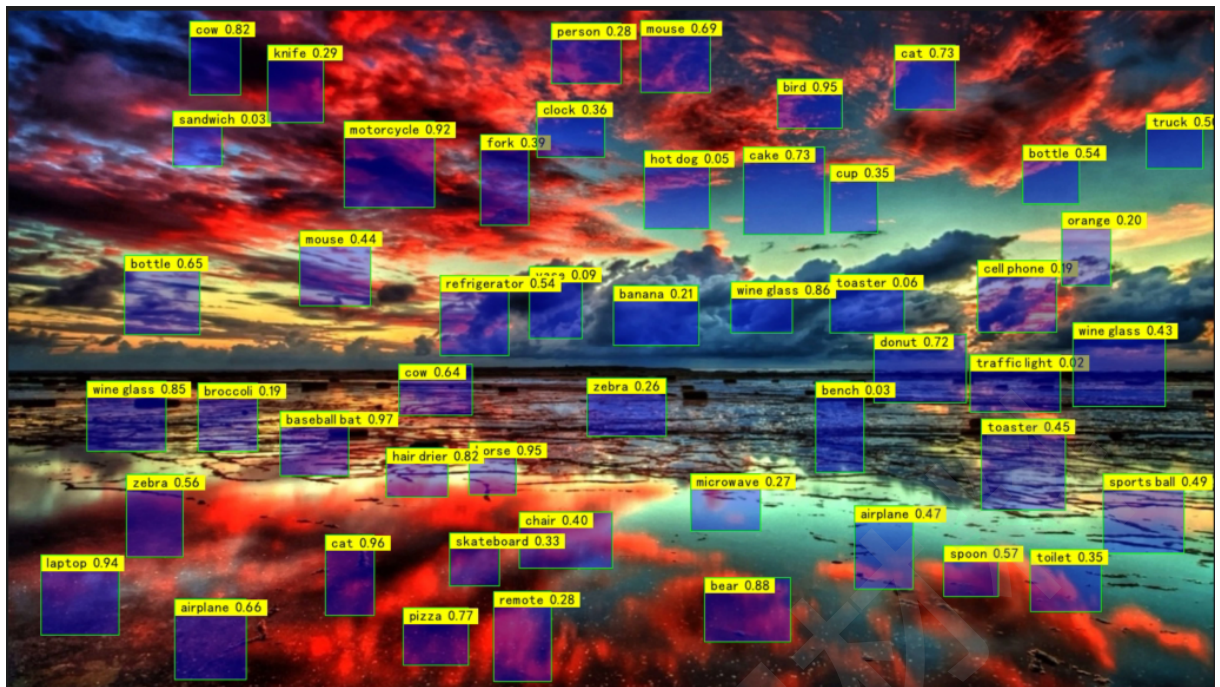
- 下载后代码内部如下图所示：



## C2 - NVCC Build & Sample Running

- C2-1 NVCC Source Native Compilation
  - cuOSD 使用 makefile 来构建，调整 makefile 的编译选项与 NVCC 11.7 对齐。
    - stdcpp := c++17 调整为 stdcpp := c++14
    - NVCC 10.0 不支持 -MT与-MF选项，删除 objs/%.cpp.mk : %.cpp 与 objs/%.cu.mk : %.cu 构建项
  - 进入 makefile 目录，使用 make 命令构建。
- C2-2 Sample Running for Validation @ NV Card
  - 在 NVIDIA 卡上运行 Sample（./cuosd simple; ./cuosd comp），验证算法效果。





- C2-3 Sample Running for Study @ NV Card
  - 在核心 Kernel 调用处 ( cuosd\_launch\_kernel\_impl函数 ) 设置断点, 查看整个函数栈。
  - 对源码中一些参数进行调整, 如绘制元素 size, 背景色 config 等, 通过程序运行结果理解关键数据结构。

### C3 - DLCC Build & Sample Running

- C3-1 Build Cmake Config for Unit Test
  - 参考《Denglin Compute Compiler V2 User Guide》、《V2 Hamming SDK 与 CUDA 11.7 差异说明》编译差异章节，配置 Cmake。
  - 把 NVCC 与 DLCC 的构建都配置在同一个 CMakeLists.txt 文件，方便使用和维护。如下列代码所示：

```
cmake_minimum_required(VERSION 3.15)
STRING(REGEX REPLACE ".*/(.*)" "\\1" CURRENT_FOLDER ${CMAKE_CURRENT_SOURCE_DIR})
set(PROJECT_NAME cuosd)
if(DLI_CUDA)
    project(${PROJECT_NAME} LANGUAGES CXX)
else()
    project(${PROJECT_NAME} LANGUAGES CUDA CXX)
endif(DLI_CUDA)

include_directories(${CMAKE_CURRENT_SOURCE_DIR}/../..../third_party/googletest/include)
add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/../..../third_party/googletest
googletest)

include_directories(
${CMAKE_CURRENT_SOURCE_DIR}/../..../third_party/opencv/opencv_4.2.0/include/opencv4)
link_directories(
${CMAKE_CURRENT_SOURCE_DIR}/../..../third_party/opencv/opencv_4.2.0/lib)
```

```

include_directories(/mercury/tools/python/penv3.6.8/include/python3.6m)
include_directories(..../src)

file(GLOB src_files ./*.cpp)
aux_source_directory(..../src src_files)
aux_source_directory(..../src/textbackend src_files)
file(GLOB cu_files ../src/*.cu ../*.cu )
list(APPEND src_files ${cu_files})
list(REMOVE_ITEM src_files ${CMAKE_CURRENT_SOURCE_DIR}/../pycuosd.cpp) #not using
for python

add_executable(${PROJECT_NAME} ${src_files})
set_target_properties(${PROJECT_NAME} PROPERTIES
    CXX_STANDARD 14
    CXX_EXTENSIONS NO
    CXX_STANDARD_REQUIRED YES)

if(DLI_CUDA)
    set_source_files_properties(${src_files} PROPERTIES LANGUAGE CXX)
    target_include_directories(${PROJECT_NAME} PUBLIC ${LLVM_CUDA_INCLUDE_DIR})
    link_directories(${LLVM_CUDA_LIB_DIR})
    set(CMAKE_CXX_COMPILER "${LLVM_BIN_DIR}/clang++")

    target_compile_options (${PROJECT_NAME} PRIVATE -fPIC --cuda-gpu-arch=dlgput64
-g -x cuda -O0 -DENABLE_TEXT_BACKEND_STB)
    target_link_libraries(${PROJECT_NAME} PRIVATE curt)

else()
    set_source_files_properties(${src_files} PROPERTIES LANGUAGE CUDA)

    target_compile_options (${PROJECT_NAME} PRIVATE -g -DNV_CARD_RUN -O0 -
DENABLE_TEXT_BACKEND_STB)
    target_include_directories(${PROJECT_NAME} PUBLIC /usr/local/cuda/include)
    target_link_libraries(${PROJECT_NAME} PRIVATE cudart cuda)

endif(DLI_CUDA)

target_link_libraries(${PROJECT_NAME} PRIVATE gtest)
target_link_libraries(${PROJECT_NAME} PRIVATE opencv_imgproc
                                                opencv_core
                                                opencv_highgui
                                                opencv_imgcodecs )

execute_process(COMMAND cp -rf ${CMAKE_CURRENT_SOURCE_DIR}/../data
${CMAKE_CURRENT_BINARY_DIR})
execute_process(COMMAND cp -rf ${CMAKE_CURRENT_SOURCE_DIR}/../imgs
${CMAKE_CURRENT_BINARY_DIR})

```

- C3-2 Sample Running for Smoke Test @ DL Card

- 使用 `./cuosd simple` 与 `./cuosd comp` Sample在登临卡上运行，进行冒烟测试。
- Sample 运行结果中如有下列两个问题，不影响单元测试 & 调试，可进入下一步。



- NV12\_BlockLinear 格式下，图像输出为 null。
- 其他格式下，输出图像有异常像素点。

## 3.4 D - Algorithm Application Unit Test & Debug

### D1 - Unit Test Design for Algorithm Application

根据 A2 Algorithm Application Requirement，设计的单元测试用例如下表。

编号	测试类型	应用场景	输入	预期输出
CT-01	Comformance Test	典型应用 1 stream 80 stream	图像： 1 ) nv12 2 ) 1920x1080 OSD 元素： 1 ) 矩形与文字数量：32 2 ) ROI模糊区域数量：32 3 ) Segment mask数量：32 OSD 位置与 Size：rand	与 NV GPU Golden 视觉效果一致
CT-02	Comformance Test	极限应用 1 stream 80 stream	图像： 1 ) nv12 2 ) 1920x1080 OSD元素： 1 ) 矩形与文字数量：100 2 ) ROI模糊区域数量：100 3 ) Segment mask数量：100 OSD 位置与 Size：rand	与 NV GPU Golden 视觉效果一致
PT-01	Performance Test	典型应用 1 stream 80 stream	图像： 1 ) nv12 2 ) 1920x1080 OSD 元素： 1 ) 矩形与文字数量：32 2 ) ROI模糊区域数量：32 3 ) Segment Mask数量：32 OSD 位置与 Size：rand	Performance 满足算法应用要求
PT-02	Performance Test	极限应用 1 stream 80 stream	图像： 1 ) nv12 2 ) 1920x1080 OSD 元素： 1 ) 矩形与文字数量：100 2 ) ROI模糊区域数量：100 3 ) Segment Mask数量：100 OSD 位置与 Size：rand	Performance 满足算法应用要求
CT-03	Comformance Test	Sample	复用 cuOSD Sample	与 NV GPU Golden 视觉效果一致

## D2 - Unit Test & Debuging

### • D2-1 Unit Test Coding & Gen Golden @ NV Card

- 根据单元测试测试用例，进行编码实现。
  - 尽量复用源码已有test代码。
  - 在源码关键位置插入 Golden 生成模块（使用 `utestSaveGoldenAndComp` 来实现，参考D2-2-1）。
- 在 NVIDIA 卡上运行。
  - 通过直接看算法效率来查看是否 Pass。
  - 生成 Golden 文件与 perf 数据。
- Build & Test Running、Golden Copy 采用了 Shell 脚本的方式，加快测试效率。代码示例如下：

```
# bash
device="dl"
is_buildall="no"
is_runing="no"
is_cmodel="no"
verbose_open="no"

while getopts ":narcv1" opt
do
    case $opt in
        n)
            device="nv"
            ;;
        a) is_buildall="yes"
            ;;
        r) is_runing="yes"
            ;;
        c) is_cmodel="yes"
            ;;
        v) verbose_open="yes"
            ;;
        *)
            echo "error argument ..."
            ;;
    esac
done

current_dir=$(basename "$(pwd)")
exe_file="cuosd"
exe_dir=""

if [ "$device" = "nv" ]; then
    echo ">>>>>>>> NVIDIA CARD env"
    cd ../../b1d_nvcc/
else
    echo ">>>>>>>> DENGLIN CARD env"
    cd ../../b1d_dlcc/
```

```

echo "    >>>>>>>> SW env CONFIG begin"
if [ ! $SDK_DIR ] && [ ! $SOFTWARE_BUILD_DIR ]; then
    echo "    >>>>>>>> source sw env"
    source ~/Share/sdk/env.sh
fi

if [[ ${SDK_DIR} != "" ]];then
    echo ${SDK_DIR}
else
    echo ${SOFTWARE_BUILD_DIR}
fi
echo "    >>>>>>>> SW env CONFIG end"
fi

exe_dir=$(pwd)
echo "    >>>>>>>> BIN path is:$exe_dir"

rm -f "$exe_file"

if [ "$is_buildall" = "yes" ]; then
    make clean
    echo "    >>>>>>>> REBUILD ALL BEGIN-----"
else
    echo "    >>>>>>>> INCREMENTAL BUILD BEGIN "
fi

if [ "$verbose_open" = "yes" ]; then
    make VERBOSE=1 -j20
else
    make -j20
fi

echo "    >>>>>>>> BUILD done!-----"

if [ "$is_running" = "yes" ]; then
    echo "    ****>>>>>>>>> BIN file is:$exe_file"
    ./"$exe_file" comp --line
    echo "    ****>>>>>>>>> RUNNING done!-----"
    if [ "$device" = "nv" ]; then
        echo "    *****>>>>>>>>> COPY Golden file to DLCC PATH -----"
        _"
        cp -f cusod_golden.dat ../bld_dlcc/cusod_golden.dat
        echo "    *****>>>>>>>>> COPY Golden done! -----"
    fi
fi

```

脚本命令使用说明如下表所示：

序号	命令	说明
----	----	----

序号	命令	说明
1	./bld -n	使用NVCC进行 Incremental编译
2	./bld -n -r	使用NVCC进行 编译，并且Running，结束后copy glolden file
3	./bld -n -a	使用NVCC进行 Rebuild all
4	./bld -n -r -a	使用NVCC进行 Rebuild all，并且Running，结束后copy glolden file
5	./bld	使用DLCC进行 Incremental编译
6	./bld -a	使用DLCC进行 Rebuild all
7	./bld -a -r	使用DLCC进行 Rebuild all，并且Running

## • D2-2 Unit Test Debugging @ DL Card

### • D2-2-1 Unit Test & Identify Faulted API/Kernel

- 在源码中添加 `utestSaveGoldenAndComp_proc` 函数，调试输出 OSD 算法处理中间过程，逐步找到 Fault API/Kernel。
- `utestSaveGoldenAndComp` 模块伪代码如下：

```
FILE *g_golden_gencmp_file; //global golden genearte&compare data file

//创建utestSaveGoldenAndComp
bool utestGlodenGenCmp_create(const char *file_name){

#ifdef NV_CARD_RUN //跑在NV卡上的宏定义
    g_golden_gencmp_file = fopen(file_name, "wb");
#else
    g_golden_gencmp_file = fopen(file_name, "rb");
#endif
    return true;
}

//utestSaveGoldenAndComp处理模块，NV卡上保存Golden，DL卡上与 Glolden Compare and Show
bool utestGoldenGenCmp_proc(void *ch1_ptr, void *ch2_ptr, int width, int height,
GOLDEN_GENCMP_TYPE img_type, const char *msg){
    int size      = width * height;
    int img_size  = size * 3 / 2;

    std::shared_ptr<uint8_t> imgbuf_ptr(new uint8_t[img_size]);
    uint8_t *imgbuf = imgbuf_ptr.get();

    if(img_type == GOLDEN_NV12_DEV){
        cudaMemcpy(imgbuf, ch1_ptr, size, cudaMemcpyDeviceToHost);
        cudaMemcpy(imgbuf + size, ch2_ptr, size / 2, cudaMemcpyDeviceToHost);
    }
    else if(img_type == GOLDEN_NV12_HOST) {
        memcpy(imgbuf, ch1_ptr, size);
    }
}
```

```

        memcpy(imgbuf + size, ch2_ptr, size / 2);
    }

#ifdef NV_CARD_RUN//跑在NV卡上的宏定义，NV卡上只做Golden保存
    fwrite(imgbuf, img_size, sizeof(uint8_t), g_golden_gencmp_file);
#else
    std::shared_ptr<uint8_t> goldenbuf_ptr(new uint8_t[img_size]);
    uint8_t *goldenbuf = goldenbuf_ptr.get();
    //读取Golden data
    fread(goldenbuf, img_size, sizeof(uint8_t), g_golden_gencmp_file);

    //DL卡上，进行NV Golden 与 Denglin 处理结果 对比与显示
    utestCmpAndShow(imgbuf, goldenbuf, width, height, img_type, msg);

#endif
    return true;
}

//utestSaveGoldenAndComp模块 Release
bool utestGoldenGenCmp_release(){
    if(g_golden_gencmp_file != nullptr){
        fclose(g_golden_gencmp_file);
    }
    return true;
}

```

#### • D2-2-2 Check for Known Issues

找到 Fault API/Kernel 后，检查是否为已知问题。

1. 通过《V2 Hamming SDK 与 CUDA 11.7 差异说明》查看该问题是否为已知差异。
2. 通过已知 Bug 清单，查看该问题是否已在 Bug 列表中。

#### • D2-2-3 Fault Resolve and Assign to Owner

- 对源码进行修改，通过绕开问题的方式来解决。
- 问题转为 Bug，按照 Bug 提交流程，提交给对接人处理。

### 问题定位示例1 - CUDA Runtime API Fault

现象：

BlockLinearNV12 图像格式下 DL Card 输出图像为无效图像（全绿图像）。

#### D2-2-1 Unit Test & Identify faulted API/Kernel：

分别在读取 YUV 之后、OSD Kernel 处理完（图像从 GPU 拷贝到 CPU 后）这2个节点，调用 `utestGoldenGenCmp_proc`。

```

dIOSD > test > gpu_image.cu > {} gpu > mask_rgba_alpha(Image *, unsigned char, unsigned char, unsigned char, unsigned char, void *)
437
438
439 void copy_yuvnv12_to(Image* image, int dst_x, int dst_y, int dst_w, int dst_h, const char* yuvnv12file, int yuvwidth, int yuvheight,
440
441     cudaStream_t stream = (cudaStream_t)_stream;
442     Image* yuv = load_yuvnv12(yuvnv12file, yuvwidth, yuvheight, stream);
443     if (yuv == nullptr) return;
444
445     utestGoldenGenCmp_proc(yuv->data0, yuv->data1, yuvwidth, yuvheight, GOLDEN_NV12_DEV, "load_yuvnv12@copy_yuvnv12_to");
446
447     if (image->format == ImageFormat::RGB) {

```

```

dIOSD > test > gpu_image.cu > {} gpu > save_image(Image *, const char *, void *)
505 // Save image to file, file format is png if rgba, otherwise jpg
506 bool save_image(Image* image, const char* file, void* _stream) {
507
508     cudaStream_t stream = (cudaStream_t)_stream;
509     if (image->format == ImageFormat::RGB) {
510         unsigned char* pdata = nullptr;
511         checkRuntime(cudaMallocHost(&pdata, image->stride * image->height));
512         checkRuntime(cudaMemcpyAsync(pdata, image->data0, image->stride * image->height, cudaMemcpyDeviceToHost, stream));
513         checkRuntime(cudaStreamSynchronize(stream));
514         stbi_write_jpg(file, image->width, image->height, 3, pdata, 100);
515         checkRuntime(cudaFreeHost(pdata));
516         return true;
517     } else if (image->format == ImageFormat::RGBA) {
518         unsigned char* pdata = nullptr;
519         checkRuntime(cudaMallocHost(&pdata, image->stride * image->height));
520         checkRuntime(cudaMemcpyAsync(pdata, image->data0, image->stride * image->height, cudaMemcpyDeviceToHost, stream));
521         checkRuntime(cudaStreamSynchronize(stream));
522         stbi_write_png(file, image->width, image->height, 4, pdata, image->stride);
523         checkRuntime(cudaFreeHost(pdata));
524         return true;
525     } else if (image->format == ImageFormat::PitchLinearNV12) {
526         unsigned char* pdata = nullptr;
527         unsigned char* rgbdata = nullptr;
528         checkRuntime(cudaMallocHost(&pdata, image->width * image->height * 3 / 2));
529         checkRuntime(cudaMallocHost(&rgbdata, image->width * image->height * 3));
530         checkRuntime(cudaMemcpyAsync(pdata, image->data0, image->width * image->height, cudaMemcpyDeviceToHost, stream));
531         checkRuntime(cudaMemcpyAsync(pdata + image->width * image->height, image->data1, image->width * image->height / 2, cudaMemcpyDeviceToHost, stream));
532         checkRuntime(cudaStreamSynchronize(stream));
533         convert_nv12_to_rgb(pdata, rgbdata, image->width, image->height);
534         stbi_write_jpg(file, image->width, image->height, 3, rgbdata, 100);
535         checkRuntime(cudaFreeHost(pdata));
536         checkRuntime(cudaFreeHost(rgbdata));
537         return true;
538     } else if (image->format == ImageFormat::BlockLinearNV12) {
539         unsigned char* pdata = nullptr;
540         unsigned char* rgbdata = nullptr;
541         checkRuntime(cudaMallocHost(&pdata, image->width * image->height * 3 / 2));
542         checkRuntime(cudaMallocHost(&rgbdata, image->width * image->height * 3));
543         checkRuntime(cudaMemcpy2DFromArrayAsync(
544             pdata, image->width,
545             (cudaArray_t)image->reserve0, 0, 0, image->width, image->height, cudaMemcpyDeviceToHost, stream
546         ));
547         checkRuntime(cudaMemcpy2DFromArrayAsync(
548             pdata + image->width * image->height, image->width,
549             (cudaArray_t)image->reserve1, 0, 0, image->width, image->height / 2, cudaMemcpyDeviceToHost, stream
550         ));
551         checkRuntime(cudaStreamSynchronize(stream));
552
553         utestGoldenGenCmp_proc(pdata, pdata + image->width * image->height, image->width, image->height, GOLDEN_NV12_HOST, "cudaMemcpyAsync@save_image");
554
555         convert_nv12_to_rgb(pdata, rgbdata, image->width, image->height);

```

1. 使用 "bld.sh -n -r" 先在 NV 卡上运行一遍，自动将 Golden 文件拷贝到调用 DLCC build 目录。
2. 再使用 "bld.sh -r" 在 DL 卡上运行一遍，第1个 `utestGoldenGenCmp_proc` 的 Golden Compare 结果为 Pass，第2个 `utestGoldenGenCmp_proc` 的 Golden Compare 结果为 Failed。

## D2-2-2 Check for Known Issues :

先确认 `cudaMemcpy2DFromArrayAsync` API 是否为已知问题。

- 查阅《V2 Hamming SDK 与 CUDA 11.7 差异说明》发现该 API 不被支持。

## D2-2-3 Fault Resolve and Assign to Owner :

- 使用 `cudaMemcpy2DFromArray(no fault API)` 代替 `cudaMemcpy2DFromArrayAsync` 的方式解决。

## 问题定位示例2 - CUDA Kernel Fault

**现象：**

PitchLinearNV12 图像格式下，从 YUV 文件读到的数据，DL Card 输出图像部分像素有明显异常，与 NV Golden 结果不一致。

**D2-2-1 Unit Test & Identify Faulted API/Kernel：**

分别在读取 YUV 之后、PitchLinearNV12 处理 Kenel 后，调用 `utestGoldenGenCmp_proc`。

```

dlOSD > test > gpu_image.cu
437
438
439 void copy_yuvnv12_to(Image* image, int dst_x, int dst_y, int dst_w, int dst_h, const char* yuvnv12file, int yuvwidth, int yuvheight, unsigned char yuvalpha)
440
441     cudaStream_t stream = (cudaStream_t)stream;
442     Image* yuv = load_yuvnv12(yuvnv12file, yuvwidth, yuvheight, stream);
443     if (yuv == nullptr) return;
444
445     utestGoldenGenCmp_proc(yuv->data0, yuv->data1, yuvwidth, yuvheight, GOLDEN_NV12_DEV, "load_yuvnv12@copy_yuvnv12_to");
446
447     if (image->format == ImageFormat::RGB) {
448
449         dim3 block(32, 32);
450         dim3 grid((dst_w + block.x - 1) / block.x, (dst_h + block.y - 1) / block.y);
451         copy_nv12_to_rgb<<grid, block, 0, stream>>>{
452             (unsigned char*)image->data0, dst_x, dst_y, dst_w, dst_h, image->width, image->height,
453             (unsigned char*)yuv->data0, (unsigned char*)yuv->data1, yuv->width, yuv->height
454         };
455     } else if (image->format == ImageFormat::RGBA) {
456
457         dim3 block(32, 32);
458         dim3 grid((dst_w + block.x - 1) / block.x, (dst_h + block.y - 1) / block.y);
459         copy_nv12_to_rgba<<grid, block, 0, stream>>>{
460             (unsigned char*)image->data0, dst_x, dst_y, dst_w, dst_h, image->width, image->height,
461             (unsigned char*)yuv->data0, (unsigned char*)yuv->data1, yuv->width, yuv->height, yuvalpha
462         };
463     } else if (image->format == ImageFormat::PitchLinearNV12) {
464
465         dim3 block(32, 32);
466         dim3 grid((dst_w + block.x - 1) / block.x, (dst_h + block.y - 1) / block.y);
467         copy_nv12_to_pl<<grid, block, 0, stream>>>{
468             (unsigned char*)image->data0, (unsigned char*)image->data1, dst_x, dst_y, dst_w, dst_h, image->width, image->height,
469             (unsigned char*)yuv->data0, (unsigned char*)yuv->data1, yuv->width, yuv->height
470         };
471
472         utestGoldenGenCmp_proc(image->data0, image->data1, image->width, image->height, GOLDEN_NV12_DEV, "copy_nv12_to_pl@copy_yuvnv12_to");
473
474     } else if (image->format == ImageFormat::BlockLinearNV12) {
475
476         dim3 block(32, 32);
477         dim3 grid((dst_w + block.x - 1) / block.x, (dst_h + block.y - 1) / block.y);
478         copy_nv12_to_bl<<grid, block, 0, stream>>>{
479             (cudaSurfaceObject_t)image->data0, (cudaSurfaceObject_t)image->data1, dst_x, dst_y, dst_w, dst_h, image->width, image->height,
480             (unsigned char*)yuv->data0, (unsigned char*)yuv->data1, yuv->width, yuv->height
481         };
482     }
483     checkRuntime(cudaStreamSynchronize(stream));
484     free_image(yuv);
485 }
486

```

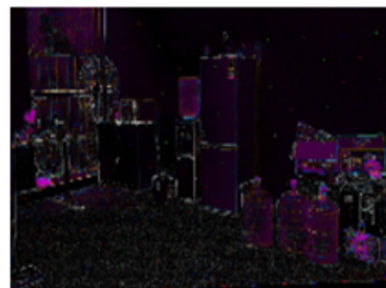
1. 使用 `bld.sh -n -r` 先在NV卡上运行一遍，自动将 Gloden文件拷贝到调用 DLCC Build 目录。
2. 再使用 `bld.sh -r` 在DL卡上运行一遍，第1个 `utestGoldenGenCmp_proc` 的 Golden Compare 结果为 Pass，第2个 `utestGoldenGenCmp_proc` 的 Golden Compare 结果为 Failed，部分像素位置显示异常（如下图所示）。



NV Golden



DL Output



Comparing Image



确定是下图出现的 Kernel 有问题。

```
static __global__ void copy_nv12_to_pl(
    unsigned char* ydata, unsigned char* uvdata, int dst_x, int dst_y, int dst_w, int dst_h, int dst_width, int dst_height,
    unsigned char* nv12_y, unsigned char* nv12_uv, int nv12_w, int nv12_h
) {
    int ix = (blockDim.x * blockIdx.x + threadIdx.x) + dst_x;
    int iy = (blockDim.y * blockIdx.y + threadIdx.y) + dst_y;
    if (ix >= dst_w || iy >= dst_h) return;

    int nx = ix * nv12_w / (float)dst_w;
    int ny = iy * nv12_h / (float)dst_h;
    unsigned char value_y = nv12_y[(ny * nv12_w + nx)];
    unsigned char value_u = nv12_uv[(ny / 2) * nv12_w + round_down2(num: nx) + 0];
    unsigned char value_v = nv12_uv[(ny / 2) * nv12_w + round_down2(num: nx) + 1];

    ix += dst_x;
    iy += dst_y;
    ydata[(iy * dst_width + ix)] = value_y;

    uvdata[(iy / 2) * dst_width + round_down2(num: ix) + 0] = value_u;
    uvdata[(iy / 2) * dst_width + round_down2(num: ix) + 1] = value_v;
}
```

单独写一个单元测试来测试该 Kernel，输入图像值初始化为 const 120，并调用该 Kernel。



```

denglin > osd > unit_test_osd > osdut_test.cc > FourThreadsWriteOneAddr
109
110
111
112 bool FourThreadsWriteOneAddr() {
113     uint width = 320, height = 320;
114     uint size = width * height * sizeof(uint8_t);
115     uint uv_size = size / 2;
116     uint8_t *h_data_src = reinterpret_cast<uint8_t *>(malloc(size));
117     uint8_t *d_data_src;
118     uint8_t *d_data_dsty, *d_data_dstuv;
119     // You, 4 days ago * [test]update 4ThreadWrite1Addr failedcase ...
120     // init input data
121     for (uint i = 0; i < width * height; i++) {
122         h_data_src[i] = 120;
123     }
124
125     EXPECT_EQ(cudaMalloc(devPtr: &d_data_src, size), cudaSuccess);
126     EXPECT_EQ(cudaMemcpy(dst: d_data_src, src: h_data_src, count: size, kind: cudaMemcpyHostToDevice), cudaSuccess);
127
128     EXPECT_EQ(cudaMalloc(&d_data_dsty, size), cudaSuccess);
129     EXPECT_EQ(cudaMalloc(&d_data_dstuv, uv_size), cudaSuccess);
130
131     // output uv data
132     uint8_t *h_uv_output = reinterpret_cast<uint8_t *>(malloc(size: uv_size));
133
134
135     // Invoke kernel
136     launch_initimg_kernel(src_img: d_data_src, sufobj_y: d_data_dsty, sufobj_uv: d_data_dstuv, width, height);
137
138     // Copy data from device back to host
139     EXPECT_EQ(cudaMemcpy(dst: h_uv_output, src: d_data_dstuv, count: uv_size, kind: cudaMemcpyDeviceToHost),
140         cudaSuccess);
141
142     EXPECT_EQ(cudaStreamSynchronize(stream: nullptr), cudaSuccess);
143
144     bool result = true;
145     for (uint i = 0; i < uv_size; i++) {
146         uint32_t temp = std::abs(x: h_data_src[i] - h_uv_output[i]);
147
148         if ((temp == 5) || (temp == 3)) {
149
150         }
151         else{
152             printf(format: "[%d] --- out=%d\n", i, h_uv_output[i]);
153             result = false;
154         }
155     }
156
157     EXPECT_EQ(result, true);
158
159     // Free device memory
160     EXPECT_EQ(cudaFree(devPtr: d_data_src), cudaSuccess);
161     EXPECT_EQ(cudaFree(devPtr: d_data_dsty), cudaSuccess);
162     EXPECT_EQ(cudaFree(devPtr: d_data_dstuv), cudaSuccess);
163
164     // Free host memory
165     free(ptr: h_data_src);
166     free(ptr: h_uv_output);
167
168     return true;
169 }
170
171 TEST_F(OsdUt, WriteImg4th1addr) {
172     EXPECT_EQ(FourThreadsWriteOneAddr(), true);
173 }

```

该单元测试在 NV 卡上 Pass，但在 DL 卡上 Failed。

阅读代码里面有4个线程写同1个地址的行为，把代码修改为1个线程写1个地址后，DL卡上也 Pass。那么基本可以确定在 DL 卡上，多个线程写同1个地址时会出现异常值的问题。

```

335 static __global__ void copy_nv12_to_pl(
336     unsigned char* ydata, unsigned char* uvdata, int dst_x, int dst_y, int dst_w, int dst_h, int dst_width, int dst_height,
337     unsigned char* nv12_y, unsigned char* nv12_uv, int nv12_w, int nv12_h
338 ) {
339     int ix = (blockDim.x * blockIdx.x + threadIdx.x) + dst_x;
340     int iy = (blockDim.y * blockIdx.y + threadIdx.y) + dst_y;
341     if (ix >= dst_w || iy >= dst_h) return;
342
343     int nx = ix * nv12_w / (float)dst_w;
344     int ny = iy * nv12_h / (float)dst_h;
345     unsigned char value_y = nv12_y[(ny * nv12_w + nx)];
346     unsigned char value_u = nv12_uv[(ny * nv12_w + round_down2(nx) + 0)];
347     unsigned char value_v = nv12_uv[(ny * nv12_w + round_down2(nx) + 1)];
348
349     ix += dst_x;
350     iy += dst_y;
351     ydata[(iy * dst_width + ix + 0)] = value_y;
352
353     if((ix & 1) && (iy && 1)){
354         uvdata[(iy / 2) * dst_width + round_down2(ix) + 0] = value_u;
355         uvdata[(iy / 2) * dst_width + round_down2(ix) + 1] = value_v;
356     }
357 }

```

## D2-2-2 Check for Known Issues :

先确认是否为已知问题：

- 查阅《V2 Hamming SDK 与 CUDA 11.7 差异说明》，发现没有该差异项的说明。
- 查阅已有 Bug 清单，也没有该问题记录。

## D2-2-3 Fault Resolve and Assign to Owner :

- 该问题作为一个 Bug，按照 Bug 提交流程，提交到 Bug 列表（ARCH 团队已确认该问题，并会在下一代架构进行改进）。

## D3 - Test Report

- 测试报告（Test Report）采用自动生成 CSV 模块输出测试结果。
- 对 CSV 表格进行二次处理后，就可以生成单元测试用例对应的 Excel 表格。对表格进行评审。

## FAQS

### 1 DL卡挂起（Hang）了，怎么处理？

使用如下命令对卡进行重启：

```
sudo bash -c "echo 1 > /proc/driver/denglin0/soft_reset"
```

### 2 如何对 CUDA C 源码编译输出汇编文件？

以 test.cu 文件为例，使用如下编译选项：

```
dlcc -x cuda test.cu --cuda-gpu-arch=dlgput64 --cuda-device-only -emit-asm -save-temps
```