威固信息

CUDA 程序编译及迁移指南

版本： V3.1.0

日期： 2023.04

目录

1 编译及迁移概述

1.1 迁移概览

RevHCU软件栈兼容CUDA通用计算模型，包括许多主要的 CUDA 等效组件、特性、API 和算法。RevHCU软件栈支持常用的CUDA API，所以大多数情况下，无需修改已有的CUDA代码即能完成迁移。

软件栈提供的编译器针对RevHCU加速卡的架构进行设计和优化。编译器基于开源 LLVM 项目开发，调用命令是 clang 或 clang++。关于 Clang 等 LLVM 工具的一般用法，请参考 LLVM 官方文档。

本指南旨在说明如何基于自研编译器，将用户原本的CUDA源码无痛迁移至RevHCU加速卡中。关于如何创建一个新的CUDA程序，不在本文档描述范围内，您可查阅其它相关资料。

1.2 与主流 GPGPU 通用计算模型的异同

RevHCU软件栈支持以下 CUDA 功能：

- Devices, context and module management (Driver API)
- Most device-side math built-in functions and intrinsic
- Memory management including shared memory manipulation
- Stream and event management
- CUDA-style topology of thread organization (grid and block), and kernel coordinate hierarchy (threadIdx, blockIdx, blockDim, gridDim)
- Kernel launching including using <<< >>> execution configuration syntax
- Barrier synchronization and cross-lane instructions
- Error reporting

除此之外，您可以通过查阅《软件栈使用指南》了解软件栈目前支持的函数库和 API，例如，暂不提供对 SIMD Intrinsics 的支持。

从其它平台迁移到RevHCU加速卡时，您需要使用RevHCU软件栈进行程序迁移，请注意以下RevHCU软件栈和CUDA Toolkit 的区别：

	RevHCU软件栈	CUDA Toolkit
线程束	64	32
线程块中包含的最大线程数量	4096	1024
预定义宏	CUDA_ARCH=ivcore1 0 __ILUVATAR__	__NVCC__ 等

2 迁移之前

2.1 准备环境

1. 检查您是否已在使用RevHCU软件栈提供的 Clang 编译器。输入如下指令：

```
$ which clang++
```

运行结果应显示为：

```
{ SOFTWARE_ROOT}/ bin/clang++
```

此处的{SOFTWARE_ROOT}指的是RevHCU软件栈的安装路径。默认安装路径是/usr/local/corex 。

2. 确保您的环境中安装了如下工具：

- GCC：建议版本是7或以上。

2.2 修改项目源文件

1. 将代码 device 部分的 double 数据类型替换成 float 数据类型。
   注：修改代码是因为 RevHCU 加速卡不支持 double 数据类型。
2. 修改项目文件夹中所有.cu 文件中对 max 和 min 的调用：
   - 用 std::max 替换 max
   - 用 std::min 替换 min

   注：修改代码是为了绕过 Clang 编译器的已知问题。
3. 修改项目文件夹中所有.cu 文件中 <<< 和 <<<，去掉多余的空格。
   注：修改代码是因为 Clang 编译器不支持 <<< 和 <<<，只支持无空格的 <<< 和 <<< 。
4. 替换或删除暂不支持的函数库和 API。您可以通过查阅《软件栈 API 参考》文档了解RevHCU加速卡软件栈目前支持的函数库和 API。例如，暂不提供对 SIMD Intrinsics 的支持。

2.3 提升性能建议

RevHCU加速卡和英伟达加速卡在芯片架构上有所差异，为了进一步优化CUDA程序性能，您还可以对项目源文件进行以下修改：

1. 修改线程束（warp size）

   RevHCU加速卡适用的线程束大小是 64，您可以根据需要，将项目源文件中定义的 warp size 修改为64。

2. 修改线程块中包含的线程数量

   RevHCU加速卡线程块中包含的线程最大值为4096，您可以根据需要，调整CUDA程序中block三元组的设置，例如：

```
dim3 blockSize(1,1,4096);
kernel<<<numBlocks, blockSize>>>(N, x, y);
```

3 CUDA 程序迁移

3.1方法一：快速迁移

以下示例是以编译RevHCU软件栈提供的 saxpy.cu 示例代码为例，构建名为 saxpy 的可执行文件。假设使用默认安装路径 /usr/local/corex 。

1. 切换到示例所在的目录。

```
$ cd /usr/local/corex/examples/cuda
```

2. 运行 **clang++** 命令及相应的必需和可选参数，生成名为 saxpy 的可执行文件。

```
$ clang++ -std=c++11 -Wall -lcudart saxpy.cu -o saxpy
```

使用clang++命令时，其命令参数选项详细说明如下:

• **-std=c++11:** 必选项。确保能编译某些 C++ std 库代码。

• **-Wall:** 可选项。启用某些用户认为有问题的构造器的所有警告。这也启用了针对特定语言的警告。

• **-lcudart:** 必选项。确保链接 CUDA 运行库；否则，会出现一些无法解析的符号。

• **- -cuda-gpu-arch=< value >:** 可选项。默认是 **- -cuda-gpu-arch=ivcore10** ，指代RevHCU加速卡。

• **-o <your output name>: 可选项。**用于设置输出的文件名。默认是 a.out 。

编译器默认开启了以下选项:

• -I/usr/local/corex/include: 用于在预处理期间设置搜索头文件的目录。

• - -cuda-path=/usr/local/corex: 指定RevHCU软件栈的路径。

• -L/usr/local/corex/lib64: 指定库文件路径。

3.2方法二：使用 Makefile 迁移

对于大型 CUDA 程序，您可以选择使用 Makefile 提高编译效率。Make 工具最主要最基本的功能就是通过创建Makefile 文件来描述源程序之间的相互关系并自动维护编译工作，您创建好 Makefile 后运行 make 命令就能为大型项目进行自动化编译。

如您想修改 Makefile 迁移大型 CUDA 程序，可参考以下步骤:

1. 修改 Makefile 中调用的编译器，需要从其它编译器（如 NVCC）指向RevHCU编译器，如您使用默认安装路径，RevHCU编译器的具体路径是 /usr/local/corex/bin/clang++ 。

由于RevHCU编译器是基于开源 LLVM 项目开发的，因此调用RevHCU编译器的命令是 clang 或 clang++。

2. 清除 Make 缓存: 在下次编译前，删除 CMakeCache 中新产生的所有文件: CMakeCache.txt，CMakeFiles，Makefile。必须执行这一步，否则会影响下次编译的正确性。

3. 重新编译: 使用 make 命令执行 Makefile 。

3.2.1 Makefile 示例

```
COREX_DIR := /usr/local/corex
CLANG_BIN_PATH := $(COREX_DIR)/bin
CXX := $(CLANG_BIN_PATH)/clang++
CXXFLAGS = -std=c++11 -g -Wall -I"$(COREX_DIR)"/include -I/usr/local/cuda/samples/common/inc --cuda-path="$(COREX_DIR)" -Wno-unknown-cuda-version
COREXLIB_DIR := $(COREX_DIR)/lib64
LDFLAGS := -L"$(COREXLIB_DIR)" -L"." -lcudart -lcurand -lclang-cpp -lcublas -lcufft -lpthread -lixml
TEST_SOURCE := main.cu test.cu
TARGETBIN := main
$(TARGETBIN):
        LD_LIBRARY_PATH="$(COREXLIB_DIR)" $(CXX) $(CXXFLAGS) $(LDFLAGS) $(TEST_SOURCE) $< -o $@
.PHONY: clean
clean:
        rm -rf $(TARGETBIN)
```

13.1 方法三：使用 CMake 进行迁移

CMake 可以让程序员通过一个与开发平台无关的 CMakeLists.txt 文件来定制整个编译流程，然后再根据目标用户的平台进一步生成所需的 Makefile 和工程文件。

如您既有的 CUDA 程序是用 CMake 和配置文件 CMakeLists.txt 编译的，您可使用RevHCU适配版 CMake 迁移这些 CUDA 程序。

RevHCU 适配版 CMake 基于官方 CMake V3.21.5，与官方 CMake 的区别是针对 find_packages(CUDA) 以及enable_language(CUDA) 两种模式做了以下RevHCU适配项：

1. 调用使用RevHCU编译器替代了官方版中的 NVCC 编译器
2. 过滤了不支持的编译选项
3. 禁用了 CUDA_SEPARABLE_COMPILATION 属性

除此之外，RevHCU适配版 CMake 和官方 CMake 功能一致。

安装完RevHCU适配版 CMake 后，您还可以通过设置环境变量 CMAKE_CUDA_COMPILER_ID 关闭以上对CMake 的RevHCU适配项：

- export CMAKE_CUDA_COMPILER_ID= “ NVIDIA ” 或 export CMAKE_CUDA_COMPILER_ID=“Clang” ：等同于官方 CMake V3.21.5
- export CMAKE_CUDA_COMPILER_ID= “ILUVATAR” ：默认值，对应RevHCU适配版 CMake

13.1.1 安装 RevHCU 适配版 CMake

1. RevHCU适配版CMake安装包如下：
   - X86 平台：cmake-3.21.5-corex.3.1.0-linux-x86_64.sh
   - ARM 平台：cmake-3.21.5-corex.3.1.0-linux-aarch64.sh
2. 执行以下命令安装RevHCU适配版 CMake（以 X86 平台为例）：

```
$ sudo bash cmake-3.21.5-corex.3.1.0-linux-x86_64.sh --skip-license --
     prefix=${COREX_CMAKE_PATH}
```

安装说明：

- --skip-license 是指跳过使用许可展示界面并接受该使用许可，--prefix 指定安装路径。
- 安装后可通过执行 ${COREX_CMAKE_PATH}/bin/cmake --version 检查安装是否成功。
- 可把 ${COREX_CMAKE_PATH}/bin 加入 PATH 环境变量，注意如果 PATH 中已经包含了其它版本的CMake 路径，需正确设置路径优先级。

13.1.2 使用 CMake

使用 CMake 的前提是您要编译的 CUDA 程序已有相应的 CMake 配置文件 CMakeLists.txt ，具体请咨询负责该程序建构的工程师。

使用RevHCU适配版 CMake 和使用官方 CMake 的方法一致，即：

1. 执行命令 `cmake ${CMakeLists_PATH}` 从 CMakeLists.txt 生成 Makefile，${CMakeLists_PATH} 是CMakeLists.txt 所在的路径。
2. 运行 make 命令进行编译。

13.1.3 CMake 示例

```
cmake_minimum_required(VERSION 3.21)

list(APPEND CMAKE_PREFIX_PATH /usr/local/corex/bin/)
list(APPEND CMAKE_BUILD_RPATH /usr/local/corex/lib64/ /usr/local/corex/lib64/stubs/)

if(NOT DEFINED CMAKE_CUDA_ARCHITECTURES)
  set(CMAKE_CUDA_ARCHITECTURES 70)
endif()

set(CMAKE_BUILD_TYPE RelWithDebInfo)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CUDA_STANDARD 11)
set(CMAKE_CUDA_COMPILER clang++)
set(CMAKE_CXX_COMPILER clang++)

# Set the project name
project(main LANGUAGES CXX CUDA)

file(GLOB MAIN_SRC *.cu *.cpp)

# Add an executable
add_executable(${PROJECT_NAME} ${MAIN_SRC})
target_include_directories(${PROJECT_NAME} PUBLIC ${CMAKE_CURRENT_BINARY_DIR} "/usr/local/cuda/include" "/usr/local/cuda/samples/common/inc")

# link the exe against the libraries
target_link_libraries(${PROJECT_NAME} PUBLIC -lcudart -lcurand -lclang-cpp -lcublas -lcufft -lpthread -lixml)
```

27 深度学习教程：迁移 mnistCUDNN 项目

以mnistCUDNN为例，本节给出如何使用 Make 工具（项目自带的 Makefile）将原生CUDA工程迁移至RevHCU加速卡的步骤。mnistCUDNN 项目使用 CUDNN/CUBLAS加速库，在 MNIST 数据集上进行深度学习训练。mnistCUDNN 项目包括两个卷积层、两个池化层、两个全连接层，以及一个线性整流层 (ReLu) 和一层 softmax。

27.1 迁移开始之前

27.1.1 准备 mnistCUDNN 项目

根据以下步骤设置 mnistCUDNN 项目：

1. mnistCUDNN 项目的下载地址为：https://github.com/haanjack/mnist-cudnn。执行以下命令获取mnistCUDNN源代码并切换到该路径：

```
$ git clone https://github.com/haanjack/mnist-cudnn.git
$ cd mnist-cudnn
```

注：项目版本是 commit 94ca9ca0b0837b89140ddebc45fd2a2cf542cb69。

2. 下载 MNIST 数据集:

```
$ ./download_mnist.sh
```

27.1.2 （可选）验证 mnistCUDNN 项目训练过程

mnistCUDNN 项目已经提供了 Makefile，您可在 NVIDIA 平台上进行编译并训练模型，验证项目设置是否正确。

1. 根据您 GPU 架构修改 mnistCUDNN 项目提供的 Makefile 中的参数 SMS = <NV_arch> 。

   注：SMS对应着各种英伟达GPU架构：Fermi 、Maxwell 、Pascal 、Volta 、Turing 。

2. 执行 Makefile:

```
$ make
```

该过程分别使用 g++ 和 nvcc 编译 mnistCUDNN 中的C++ 和 CUDA 代码，最后生成的可执行文件可以运行在 NVIDIA 平台上。

3. 编译成功后，您可以执行以下命令在 NVIDIA 平台上开始模型训练。

```
$ ./train
```

27.1.3 修改 mnistCUDNN 项目代码

参考 “2.2节 修改项目源文件”, 在mnistCUDNN中, 需要修改 “convolution.cu” 和 “src/loss.cu” 两个文件中的两处 “max” 和 “min” 函数代码，具体如下:

```
convolution.cu:
- workspace_size = max(workspace_size, temp_size);
+ workspace_size = std::max(workspace_size, temp_size);
src/loss.cu:
- int num_blocks = min(num_blocks_per_sm * num_sms, \
+ int num_blocks = std::min(num_blocks_per_sm * num_sms, \
```

27.2 使用 Makefile 进行程序迁移

1. 修改 Makefile, 尤其是 CUDA_PATH 、HOST_COMPILER 、NVCC 、NVCC_FLAGS 和 SMS 的值。您可以参考以下 git diff 的结果进行修改。修改参数说明见表1。

```
$ git diff Makefile
diff --git a/Makefile b/Makefile
index fa306ec..d6652c1 100644
--- a/Makefile
+++ b/Makefile
@@ -1,21 +1,22 @@
-CUDA_PATH=/usr/local/cuda
```

```
-HOST_COMPILER ?= g++
-NVCC=${CUDA_PATH}/bin/nvcc -ccbin ${HOST_COMPILER}
+CUDA_PATH=/usr/local/corex
+HOST_COMPILER ?= clang++
+NVCC=clang++
  TARGET=train convolution
  INCLUDES = -I${CUDA_PATH}/samples/common/inc -I$(CUDA_PATH)/include
-NVCC_FLAGS=-G --resource-usage -Xcompiler -rdynamic -Xcompiler -fopenmp -rdc=true
                          -lnvToolsExt
+NVCC_FLAGS=-G --resource-usage

IS_CUDA_11:=$(shell echo `nvcc --version | grep compilation | grep -Eo -m 1 '[0-9]
            +.[0-9]' | head -1` \>= 11.0 | bc)

# Gencode arguments
-SMS = 35 37 50 52 60 61 70 75
ifeq "$(IS_CUDA_11)" "1"
     +SMS = 52 60 61 70 75 80
endif
-$(foreach sm, ${SMS}, $(eval GENCODE_FLAGS+=-gencode arch=compute_$(sm),code=sm_$(sm)))
-LIBRARIES += -L/usr/local/cuda/lib -lcublas -lcudnn -lgomp -lcurand
+LIBRARIES += -L/usr/local/corex/lib64 -lcublas -lcudnn -lgomp -lcurand -lcudart -ldl
ALL_CCFLAGS += -m64 -g -std=c++11 $(NVCC_FLAGS) $(INCLUDES) $(LIBRARIES)
```

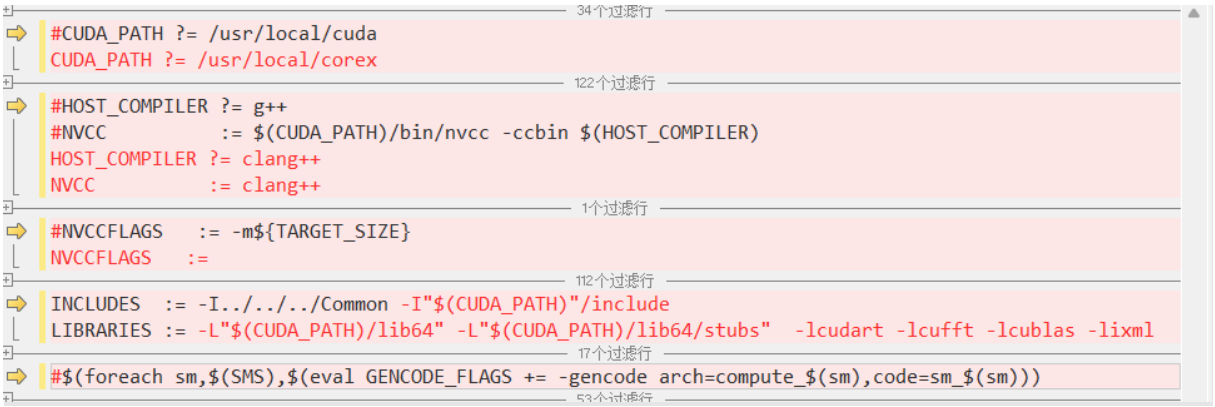

表1 修改参数说明

修改参数	参数说明	修改前	修改后
CUDA_PATH	软件栈路径	/usr/local/cuda	/usr/local/corex
HOST_COMPI	主机编译器	g++	clang++

LER			
NVCC	设备编译器	${CUDA_PATH}/bin/nvcc -ccbin ${HOST_COMPILER}	clang++
NVCC_FLAGS	NVCC编译参数	-G --resource-usage -Xcompiler -rdynamic -Xcompiler -fopenmp -rdc=true	-G --resource-usage
SMS	适配的设备架构参数	35 37 50 52 60 61 70 75	52 60 61 70 75 80
LIBRARIES	各类库	-L/usr/local/cuda/lib -lcublas -lcudnn -lgomp -lcurand	-L/usr/local/corex/lib64 -lcublas -lcudnn -lgomp -lcurand -lcudart -ldl

2. 执行 Makefile，并在RevHCU加速卡上开始模型训练:

```
$ make
$ ./train
```

注：如过程中碰到报错，请参考“常见问题”。

28 常见问题

问题1: 遇到报错 error: no matching function for call to 'max'

问题现象

```
mnist-cudnn/src/layer.cu:588:19: error: no matching function for call to 'max'
workspace_size = max(workspace_size, temp_size);
```

问题分析

这是 Clang 编译器的已知问题。

解决方法

用 std::max 替换 max。即把 workspace_size = max(workspace_size, temp_size) 修改成 workspace_size = std::max(workspace_size, temp_size)。

问题2: 遇到报错 error: __float128 is not supported on this target

问题现象

```
 In file included from /usr/lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++
    /7.5.0 /cmath:47:
 /usr/lib/gcc/x86_64-linux-gnu/7.5.0/../../../include/c++/7.5.0/bits/std_abs.h:102:7:
   error: __float128 is not supported on this target
abs(__float128 __x)
      ^
/usr/lib/gcc/x86_64-linux-gnu/7.5.0/../../../include/c++/7.5.0/bits/std_abs.h:101:3:
    error: __float128 is not supported on this target
__float128
```

问题分析

CMake 自动加上了 -std=gnu++14 编译选项。

解决方法

如果 -std=gnu++14 编译选项对本次编译并非必须，您可以在 CMake 的配置中删除该选项。

例如，mnistCUDNN 项目并不需要设置 -std=gnu+ +14 编 译 选 项， 您可以将 mnist-cudnn/ build/ CMakeFiles/train.dir/flags.make 中以下内容:

```
CUDA_FLAGS=--std=c++14 -O3 -DNDEBUG --cuda-gpu-arch=ivcore10 --cuda-
        path=/usr/local/corex
        -O3 -DNDEBUG --cuda-path=/usr/local/corex -std=gnu++14
```

改成

```
CUDA_FLAGS=--std=c++14 -O3 -DNDEBUG --cuda-gpu-arch=ivcore10 --cuda-
        path=/usr/local/corex
        -O3 -DNDEBUG --cuda-path=/usr/local/corex
```

问题3: 遇到报错 /usr/bin/ld: cannot find -lcudadevrt

问题现象

```
[16%] Linking CXX executable train
/usr/local/bin/cmake -E cmake_link_script CMakeFiles/train.dir/link.txt --verbose=1
/usr/local/bin/clang++ --std=c++14 -lcudart -lcublas -lcudnn -lgomp -lcurand -
    lnvToolsExt -L/usr/local/cuda/lib64 -O3 -DNDEBUG
    CMakeFiles/train.dir/src/layer.cu.o CMakeFiles/ train.dir/src/loss.cu.o
    CMakeFiles/train.dir/src/mnist.cpp.o CMakeFiles/train.dir/src/ network.cpp.o
    CMakeFiles/train.dir/train.cpp.o -o train -lcudadevrt -lcudart_static -lrt -
    lpthread -ldl
/usr/bin/ld: cannot find -lcudadevrt
```

问题分析

CMake 自动将 cudadevrt 赋值给了若干 flag，但是RevHCU软件栈暂不支持 CUDA Device Runtime (cudadevrt) 函数库。

解决方法

有些项目，例如 mnistCUDNN 项目，实际上并未使用到这个函数库。所以，您可以将文件中设置相关 flag 的代码注释掉。以 CMake 3.18 为例，参考下述对 /usr/local/share/cmake-3.18/Modules/Compiler/ Clang-CUDA.cmake 的修改。

```
#set(CMAKE_CUDA_RUNTIME_LIBRARY_DEFAULT "STATIC")
#set(CMAKE_CUDA_RUNTIME_LIBRARY_LINK_OPTIONS_STATIC "cudadevrt;cudart_static")
#set(CMAKE_CUDA_RUNTIME_LIBRARY_LINK_OPTIONS_SHARED "cudadevrt;cudart")
#set(CMAKE_CUDA_RUNTIME_LIBRARY_LINK_OPTIONS_NONE "")
```

问题4：目前不支持哪些 CUDA 特性?

本次发布尚不支持以下 CUDA 特性：

- Double precision on GPU device
- Textures filtering/interpolation and cuda array • Dynamic parallelism
- Unified memory
- Graphics interoperability and rendering
- Tensor Core related APIs
- Multi-Process Service (MPS)
- Cooperative Groups
- Warp level primitives
- CUDA Graphs
- cuBLASLt
- Virtual Memory Management APIs

问题 5: makefile 静态库链接后，符号定义找不到?

问题现象

```
undefined reference to `myprint'
clang-13: error: linker command failed with exit code 1 (use -v to see invocation)
```

问题分析

链接的静态库有前后的依赖顺序。

解决方法

修正静态库的链接顺序。