Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

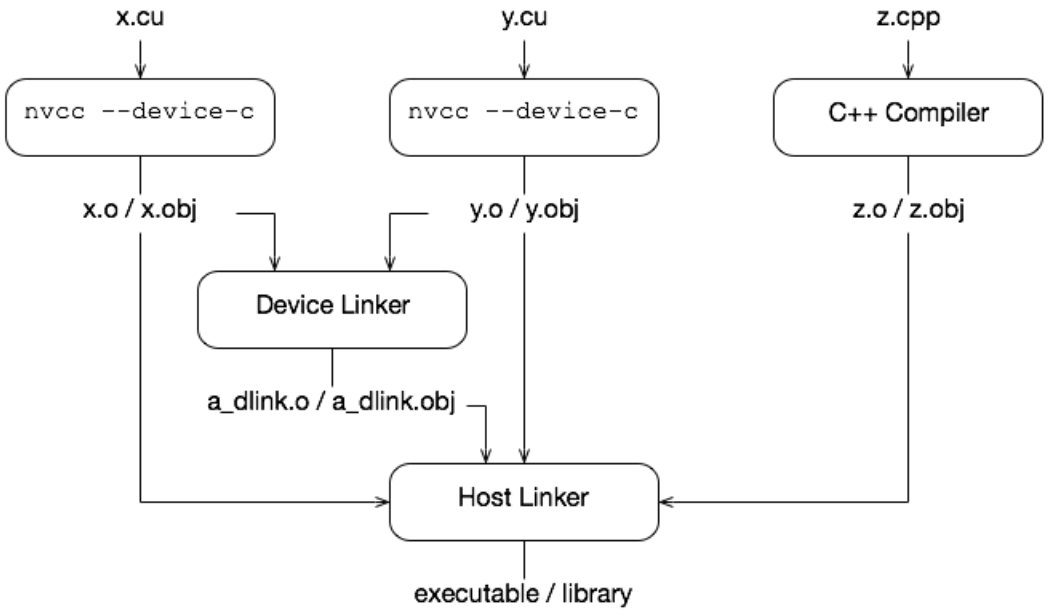
About

SiteXC

分离编译并打包 CUDA 函数库

最近遇到一个小问题：我写了一个小的库，这个库需要同时提供 CUDA device API 和 CPU host C API。本文记录一下编译、打包和链接到这个库的方法。

CUDA 分离编译 (separate compilation) 允许跨文件访问 device functions and variables. CUDA 文档给出了这么一个流程图：



主要有两步：

1. 对于需要将本文件的 device functions and variables 暴露给其他文件访问的 CUDA source code， `nvcc` 需要加上 `-rdc=true/--device-c` 参数来使用分离编译；
2. 分离编译得到的 obj files 需要再用 device linker 链接一个新的 obj file，然后把所有 obj files 一起用 host linker 处理得到 executable / library。

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

Tricky 的地方出在图上语焉不详处，即最后一步 host linker 到 library。我摸索了一下以后发现 device linker 这一步不应该放在打包 library 中，而应该放在最后生成可执行文件的过程中。同时，需要打开 `-fPIC` 编译选项。下面给出两个我测试过的 makefiles，分别用来编译并打包库，以及编译应用程序和链接到我们打包的库。

假设我们的库和应用程序：

- 只有 CUDA code 和 C code，C code 使用 C99 标准
- CUDA code 使用 CUDA 10.0 来编译，CUDA 10.0 安装在 `/usr/local/cuda-10.0`
- CUDA code 只为 Pascal 和 Volta 架构生成代码
- C code 包含 MPI 和 OpenMP 函数，需要使用 `mpicc` （或者其他 MPI compiler wrapper）来编译
- 我们的库打包好以后安装在 `$LIBDIR/lib/libmylib.a`，头文件安装在 `$LIBDIR/include/` ，应用程序可执行文件名为 `myapp.exe`

下面这个 makefile 适用于打包 library:

```
1  LIBA    = libmylib.a
2
3  CC      = mpicc
4  CFLAGS  = -O3 -Wall -g -std=c99 -fPIC
5
6  GENCODE_SM60  = -gencode arch=compute_60,code=sm_60
7  GENCODE_SM70  = -gencode arch=compute_70,code=sm_70
8  GENCODE_FLAGS = $(GENCODE_SM60) $(GENCODE_SM70)
9
10 CUDA_PATH   ?= /usr/local/cuda-10.0
11 NVCC        = $(CUDA_PATH)/bin/nvcc
12 NVCCFLAGS   = -O3 -g --compiler-options -fPIC $(GENCODE_FLAGS)
13
14 ifeq ($(shell $(CC) --version 2>&1 | grep -c "icc"), 1)
15 AR      = xiar rcs
16 CFLAGS += -fopenmp -xHost
17 endif
18
19 ifeq ($(shell $(CC) --version 2>&1 | grep -c "gcc"), 1)
20 AR      = ar rcs
21 CFLAGS += -fopenmp -lm -march=native -Wno-unused-result -Wno-unused-function
22 endif
23
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
24 C_SRCS  = $(wildcard *.c)
25 C_OBJS  = $(C_SRCS:.c=.c.o)
26 CU_SRCS = $(wildcard *.cu)
27 CU_OBJS = $(CU_SRCS:.cu=.cu.o)
28 OBJS    = $(C_OBJS) $(CU_OBJS)
29
30 # Delete the default old-fashion double-suffix rules
31 .SUFFIXES:
32
33 all: $(LIBA)
34
35 $(LIBA): $(OBJS)
36         $(NVCC) $(NVCCFLAGS) -lib -o $@ $^
37
38 %.c.o: %.c
39         $(CC) $(CFLAGS) -o $@ -c $^
40
41 %.cu.o: %.cu
42         $(NVCC) $(NVCCFLAGS) -rdc=true -o $@ -c $^
43
44 clean:
45         rm $(OBJS) $(LIBA)
```

下面这个 makefile 适用于编译应用程序并链接到我们打包好的库：

```
1  EXE     = myapp.exe
2
3  CC      = mpicc
4  CFLAGS  = -O3 -Wall -g -std=c99 -fPIC -I$(LIBDIR)/include
5  LDFLAGS = -fopenmp -L$(CUDA_PATH)/lib64 -L$(LIBDIR)/lib -L./ -lcuda -lcudart -lmylib
6
7  GENCODE_SM60  = -gencode arch=compute_60,code=sm_60
8  GENCODE_SM70  = -gencode arch=compute_70,code=sm_70
9  GENCODE_FLAGS = $(GENCODE_SM60) $(GENCODE_SM70)
10
11 CUDA_PATH   ?= /usr/local/cuda-10.0
12 NVCC        = $(CUDA_PATH)/bin/nvcc
13 NVCCFLAGS   = -O3 -g --compiler-options '-fPIC' -I$(LIBDIR)/include $(GENCODE_FLAGS)
14
15 ifeq ($(shell $(CC) --version 2>&1 | grep -c "icc"), 1)
16 CFLAGS += -fopenmp -xHost
17 endif
18
19 ifeq ($(shell $(CC) --version 2>&1 | grep -c "gcc"), 1)
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
20 CFLAGS += -fopenmp -lm -march=native -Wno-unused-result -Wno-unused-function
21 endif
22
23 C_SRCS  = $(wildcard *.c)
24 C_OBJS  = $(C_SRCS:.c=.c.o)
25 CU_SRCS = $(wildcard *.cu)
26 CU_OBJS = $(CU_SRCS:.cu=.cu.o)
27 OBJS    = $(C_OBJS) $(CU_OBJS) dlink.o
28
29 # Delete the default old-fashion double-suffix rules
30 .SUFFIXES:
31
32 all: $(EXE)
33
34 <your_application>.exe: dlink.o $(OBJS) $(LIB_DIR)/lib/libmylib.a
35         $(CC) $^ -o $@ $(LDFLAGS)
36
37 %.c.o: %.c
38         $(CC) $(CFLAGS) -o $@ -c $^
39
40 %.cu.o: %.cu
41         $(NVCC) $(NVCCFLAGS) -rdc=true -o $@ -c $^
42
43 dlink.o: $(CU_OBJS)
44         $(NVCC) $(NVCCFLAGS) -dlink -o $@ $(CU_OBJS) $(LIB_DIR)/lib/libmylib.a
45
46 clean:
47         rm $(OBJS) $(EXE)
```

注意：生成 `myapp.exe` 的命令里，这些 obj files 的顺序非常重要！

如果 `myapp.exe` 只需要一个 `myapp.cu`，MPI 安装在 `MPI_PATH` 目录下，那么可以用一个简化版的 makefile 来生成 `myapp.exe`，其中直接使用 `nvcc` 来链接生成最终的可执行文件，跳过 device linker:

```
1 EXE = myapp.exe
2 CXX = mpicxx
3
4 GENCODE_SM60  = -gencode arch=compute_60,code=sm_60
5 GENCODE_SM70  = -gencode arch=compute_70,code=sm_70
6 GENCODE_FLAGS = $(GENCODE_SM60) $(GENCODE_SM70)
7
8 CUDA_PATH   ?= /usr/local/cuda-10.0
9 NVCC        = $(CUDA_PATH)/bin/nvcc
```

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home
Archives
About
SiteXC

```
10
11 NVCC_FLAGS    = -rdc=true -Xcompiler -fopenmp -lineinfo $(GENCODE_FLAGS) -std=c++11 -g
12 NVCC_FLAGS   += -I$(LIBDIR)/include -I$(MPI_PATH)/include
13 NVCC_LDFLAGS  = -ccbin=$(CXX) --compiler-options -fopenmp -L$(CUDA_PATH)/lib64 -lcuda -lcuda
14 NVCC_LDFLAGS += -L$(MPI_PATH)/lib -lmpi -L$(LIBDIR)/lib -lmylib
15
16 %.exe: %.cu
17         $(NVCC) $(NVCC_FLAGS) -c $^ -o $@.o
18         $(NVCC) $(GENCODE_FLAGS) $@.o -o $@ $(NVCC_LDFLAGS)
19
20 clean:
21         rm *.o *.exe
```

Reference

1. CUDA Toolkit Documentation - Using Separate Compilation in CUDA
2. Separate Compilation and Linking of CUDA C++ Device Code

发布于 2020-10-31

tags: { MPI } { CUDA }