CUDA Learning (I)-NVCC Compilation Process

原创 Scott F Posted on 2019-05-07 16:43:29 Read 7.2k Collection 33 Likes 5

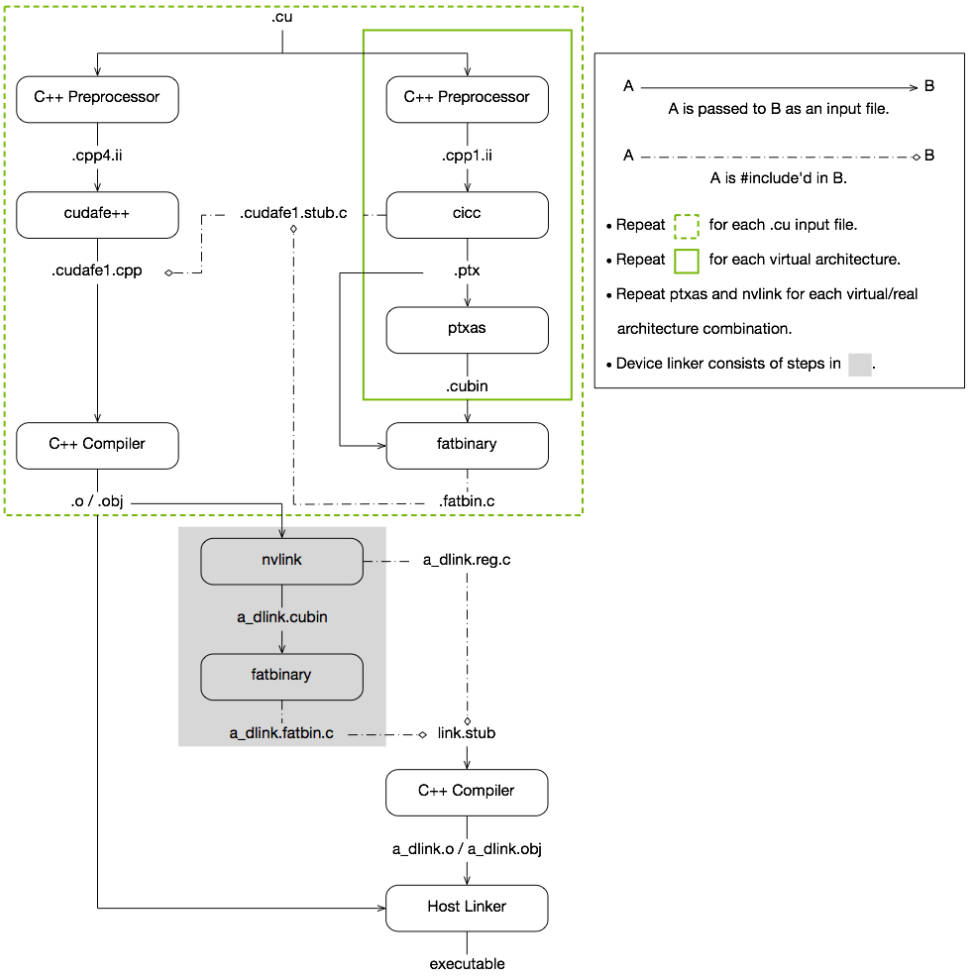
Category Column: CUDA Article Tags: CUDA GPU parallel computing

摘要 This article introduces the compilation process of NVCC in detail, including offline compilation and just-in-time compilation. Offline compilation divides the CUDA source program into host-side and device-side code processing; just-in-time compilation performs different operations based on the matching of cubin files and GPUs. It also mentions that NVC C considers backward compatibility, and virtual and real architectures can be set through commands, and the -arch configuration must be lower than -code.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

text:

The overall compilation process of NVCC can be described by a diagram:



The compilation process of NVCC is divided into two parts: offline compilation and just-in-time compilation:

Offline compilation (in the green dashed box):

The CUDA source program (xxx.cu file) is preprocessed before compilation and is divided into host code and device code (i.e. the left and right branches in the figure):

1. As shown in the right branch of the figure: On the device side, the code will be compiled into a ptx file (which can be regarded as an assembly file for the device side) or a directly executable binary file xxx.cubin, and then the .ptx/.cubin file will be placed in the fatbinary file.

2. As shown in the left branch of the figure: the host code is preprocessed and embedded into the fatbinary file, and the CUDA-specific C++ extensions are converted into standard C++ structures (synthesized into cudafe1.cpp through cudafe++ and cudafe1.stub.c), and then the C++ host compiler compiles the synthesized host code and the embedded fatbinary into host .o/.obj files.

Just-in-time compilation (below the green dashed box in the picture):

When the host device starts the code, the CUDA run-time system detects the fatbinary file to obtain an image suitable for the GPU device (mentioned later):

1. If the previous cubin file matches the current GPU, the run-time system links multiple device-side files through nvlink, and finally links the host-side files into executable files through the host linker.

2. If the virtual device represented by the cubin file does not match the current GPU or the fatbinary file only contains ptx files, the ptx files in the fatbinary file will be compiled in real time and linked (nvlink) to the host file and finally linked to the executable file through the host linker.

Replenish:

NVCC takes the backward compatibility of applications into great consideration. (As shown in the green solid line in the top figure), the input device code is compiled into ptx according to the virtual GPU structure (virtual architecture), and it is compiled into cubin file according to the current real GPU structure, which can be directly executed at that time. (As shown in Figure 1: virtual architecture)

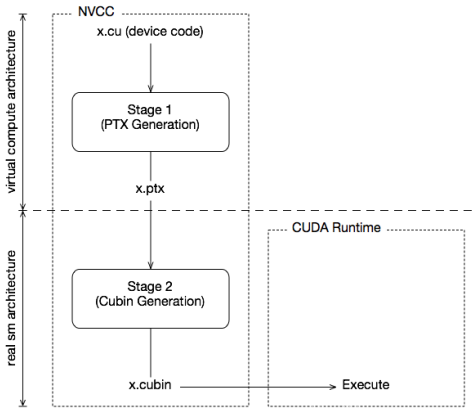


Figure 1: Virtual architecture

Due to the existence of ptx, its compatibility can be improved by just-in-time compiling ptx into cubin files and executing them at runtime. (Figure 2: Just-in-Time Compilation)

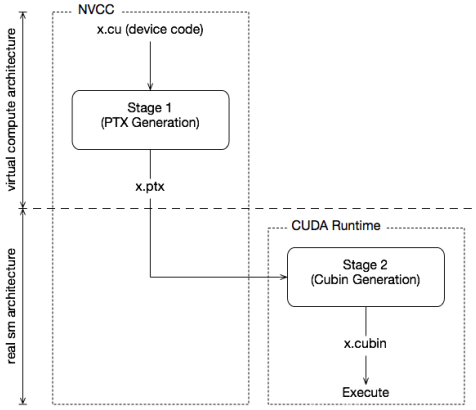


Figure 2: Just-in-Time Compilation

nvcc can set the corresponding virtual compute architecture and real sm architecture through commands


```
nvcc xxx.cu -arch=compute_30 -code=sm_30
```

Among them, -arch=compute_30 means configuring the virtual GPU architecture to compute capability 3.0 and generating the corresponding ptx; -code=sm_30 compiles the ptx file into a cubin binary file with sm 3.0.

Note: -arch configuration must be lower than -code, otherwise compilation will fail.

Since nvcc compilation supports backward compilation, ptx is compiled by the virtual architecture, and the actual version is lower than the virtual gpu architecture, which will cause it to fail to run.

The second article on nvcc compilation process: https://blog.csdn.net/shungry/article/details/89788342

For more information, you can directly refer to the official nvcc documentation: https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html