# [CUDA架构及对应编译参数](https://www.cnblogs.com/phillee/p/12049208.html)

NVIDIA CUDA C++ 编译器 nvcc 基于每个内核，既可以用来产生特定于体系结构的 cubin 文件，又能产生前向兼容的 PTX 版本。

每个 cubin 文件针对特定的计算能力版本，并且仅与相同主要版本号的 GPU 架构向前兼容。

例如，针对计算能力 3.0 的 cubin 文件支持所有计算能力 3.x 设备，但不支持计算能力 5.x 或 6.x 设备。

基于这个原因，为了确保与应用程序发布后引入的 GPU 架构的向前兼容性，建议所有应用程序都包含其内核的 PTX 版本。

注意：CUDA 运行时应用程序同时包含针对给定体系结构的 cubin 和 PTX 代码，默认情况下自动使用 cubin，严格保留 PTX 路径以实现前向兼容性。

对于已经包含了其内核的 PTX 版本的应用程序应在基于 Volta 的 GPU 上原样工作。而对于通过 cubin 文件仅支持特定 GPU 架构的应用程序，需要更新以提供与 Volta 兼容的 PTX 或 cubins 。

## 1. 不同NVIDIA显卡对应的SM架构（CUDA arch and CUDA gencode）

1.1 NVIDIA的nvcc sm标志是干什么用的

使用NVCC编译器编译CUDA源文件时，架构标志位 -arch 指明了CUDA文件编译产生的结果所依赖的NVIDIA GPU架构的名称，而生成码 -gencode 允许生成更多的PTX文件，并且对不同的架构可以重复许多次。

当编译CUDA代码时，只能根据一种架构进行编译，用来匹配使用最多的GPU显卡。

这使得运行时间最短，因为code generation总是发生在编译期间，如果你只指明了-gencode而忽略了-arch，GPU code generation会由CUDA驱动在JIT编译器产生。

若要加速CUDA编译，就减少不相关-gencode标志的数量，然而有时我们却希望更好的CUDA向后兼容性，只能添加更多的-gencode。

1.2 首先检查你使用的GPU型号和CUDA版本

以下是支持的 sm 变量和相对应的典型显卡型号

CUDA 7以上版本

* Fermi (CUDA 3.2 一直到 CUDA 8) (deprecated from CUDA 9):

SM20 or SM\_20, compute\_30 – 比较旧的显卡 GeForce 400, 500, 600, GT-630

* Kepler (CUDA 5及以上):

SM30 or SM\_30, compute\_30 – Kepler architecture (generic – Tesla K40/K80, GeForce 700, GT-730)

Adds support for unified memory programming

SM35 or SM\_35, compute\_35 – More specific Tesla K40

Adds support for dynamic parallelism. Shows no real benefit over SM30 in my experience.

SM37 or SM\_37, compute\_37 – More specific Tesla K80

Adds a few more registers. Shows no real benefit over SM30 in my experience

* Maxwell (CUDA 6及以上版本):

SM50 or SM\_50, compute\_50 – Tesla/Quadro M series

SM52 or SM\_52, compute\_52 – Quadro M6000 ,

GeForce 900,

GTX-970, GTX-980, GTX Titan X

SM53 or SM\_53, compute\_53 – Tegra (Jetson) TX1 / Tegra X1

* Pascal (CUDA 8及以上版本)

SM60 or SM\_60, compute\_60 – Quadro GP100,

Tesla P100,

DGX-1 (Generic Pascal)

SM61 or SM\_61, compute\_61 – GTX 1080, GTX 1070, GTX 1060, GTX 1050, GTX 1030,

Titan Xp,

Tesla P40, Tesla P4,

Discrete GPU on the NVIDIA Drive PX2

SM62 or SM\_62, compute\_62 – Integrated GPU on the NVIDIA Drive PX2, Tegra (Jetson) TX2

* Volta (CUDA 9及以上版本)

SM70 or SM\_70, compute\_70 – DGX-1 with Volta,

Tesla V100,

GTX 1180 (GV104),

Titan V, Quadro GV100

SM72 or SM\_72, compute\_72 – Jetson AGX Xavier

* Turing (CUDA 10及以上版本)

SM75 or SM\_75, compute\_75 – GTX Turing – GTX 1660 Ti,

RTX 2060, RTX 2070, RTX 2080,

Titan RTX,

Quadro RTX 4000, Quadro RTX 5000, Quadro RTX 6000, Quadro RTX 8000

1.3 根据 NVIDIA 的官方说明

nvcc的 -gencode= 命令行选项的 arch= 指定前端编译目标，并且必须始终为PTX版本。

code= 指定后端编译目标，可以是cubin或PTX或两者均可。

只有由 code= 指定的后端目标版本将保留在结果二进制文件中，至少包含一个PTX以提供Volta兼容。

1.4 参数示例

取得最大兼容性的CUDA 7标志示例

-arch=sm\_30 -gencode=arch=compute\_20,code=sm\_20 -gencode=arch=compute\_30,code=sm\_30 -gencode=arch=compute\_50,code=sm\_50 -gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_52,code=compute\_52

CUDA 8

-arch=sm\_30 -gencode=arch=compute\_20,code=sm\_20 -gencode=arch=compute\_30,code=sm\_30 -gencode=arch=compute\_50,code=sm\_50 \

-gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_60,code=sm\_60 \

-gencode=arch=compute\_61,code=sm\_61 -gencode=arch=compute\_61,code=compute\_61

CUDA 9 Volta 型号显卡

-arch=sm\_50 -gencode=arch=compute\_50,code=sm\_50 -gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_60,code=sm\_60 \

-gencode=arch=compute\_61,code=sm\_61 -gencode=arch=compute\_70,code=sm\_70 \

-gencode=arch=compute\_70,code=compute\_70

CUDA 10 Turing 型号显卡

-arch=sm\_50 -gencode=arch=compute\_50,code=sm\_50 -gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_60,code=sm\_60 \

-gencode=arch=compute\_61,code=sm\_61 -gencode=arch=compute\_70,code=sm\_70 \

-gencode=arch=compute\_75,code=sm\_75 -gencode=arch=compute\_75,code=compute\_75

# 2.基于NVIDIA Volta架构为GPU构建 CUDA 应用程序

第一步：检查 Volta 兼容的设备代码编译到了应用程序之中

2.1.1 使用 CUDA Toolkit 8.0 及之前版本的应用程序

使用CUDA Toolkit版本2.1至8.0构建的CUDA应用程序兼容Volta，只要构建时包含了其内核的PTX版本。可以通过如下步骤检测：

1. 下载安装最新版驱动 <http://www.nvidia.com/drivers>
2. 设置环境变量 CUDA\_FORCE\_PTX\_JIT=1
3. 登录应用程序

第一次登录CUDA应用程序时，CUDA驱动将会为每个CUDA内核进行JIT编译PTX，在本地cubin代码中使用。

如果按上述说明设置了环境变量登录之后正常工作，说明已经成功验证了Vlota兼容性。（注意：确保在验证之后将CUDA\_FORCE\_PTX\_JIT复位！）

2.1.2 使用CUDA Toolkit 9.0的应用程序

使用CUDA Toolkit 9.0构建的CUDA应用程序兼容Volta，只要构建时包含了Volta-native cubin格式的内核或PTX格式的内核或两者都有。

第二步：构建 Volta 支持的应用程序

当一个 CUDA 应用程序登录内核时，CUDA Runtime 会决定系统中每个 GPU 的计算能力，并利用这一信息自动寻找该内核最匹配的 cubin 或 PTX 版本。

如果 cubin 文件支持当前可用的目标 GPU 的体系架构，就是用该 cubin 文件；否则 CUDA Runtime 将加载 PTX ，并在登录之前 JIT 编译此 PTX 以得到本地 cubin 格式的 GPU cubin 文件。

如果两者都不满足，内核登录失败。

构建本地 cubin 格式或至少支持 Volta 的 PTX 的应用程序的方法取决于使用的 CUDA Toolkit 版本。

提供本地 cubin 文件的主要优势如下：

节省了终端用于 JIT 编译仅支持 PTX 的内核的时间。所有的内核编译成应用程序之后在加载时必须要有本地二进制文件，或者将即刻从 PTX 进行编译构建，包括来自所有库文件的内核，这些库文件链接到应用程序，

即使该应用程序永远都不会登录这些内核。特别是，当使用比较大的库时，JIT 编译过程将消耗相当的时间。

CUDA 驱动将缓存这些 PTX JIT 产生的 cubin 结果，这多数情况下对一个使用者只有一次时间消耗，但如果有可能我们还是希望避免。

PTX JIT 编译内核通常并不能很好地利用较新的 GPU 的架构特征，也即是说本地编译产生的代码可能运行得更快或更准确。

2.2.1 使用 CUDA Toolkit 8.0 及之前版本的应用程序

CUDA Toolkit 8.0 或更早版本中包含的编译器会生成 Maxwell 和 Pascal 等早期 NVIDIA 架构的本地 cubin 文件，但无法生成 Volta 架构的 cubin 文件。

为了在使用 8.0 或更早版本的 CUDA Toolkit 时支持 Volta 和将来的体系结构，编译器必须为每个内核生成 PTX 版本。

下面是可以用来构建 mykernel.cu 的编译器设置，mykernel.cu 可以在 Maxwell 或 Pascal 设备上本地运行，在 Volta 设备上通过 PTX JIT 运行。

注意

* compute\_XX 指的是 PTX 版本
* sm\_XX 指的是 cubin 版本
* nvcc 的 -gencode= 命令行选项的 arch= 指定前端编译目标，并且必须始终为 PTX 版本。
* code= 指定后端编译目标，可以是 cubin 或 PTX 或两者均可。
* 只有由 code= 指定的后端目标版本将保留在结果二进制文件中，至少包含一个PTX以提供Volta兼容性。

Mac/Linux

/usr/local/cuda/bin/nvcc -gencode=arch=compute\_50,code=sm\_50 -gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_60,code=sm\_60 \

-gencode=arch=compute\_61,code=sm\_61 -gencode=arch=compute\_61,code=compute\_61 \

-O2 -o mykernel.o -c mykernel.cu

另外，你可能熟悉 nvcc 命令行选项 -arch=sm\_XX，它的简写相当于上面使用的更明确的 -gencode= 命令行选项。

-arch=sm\_XX 展开成如下形式

-gencode=arch=compute\_XX,code=sm\_XX

-gencode=arch=compute\_XX,code=compute\_XX

然而，虽然 -arch=sm\_XX 命令行选项确在默认情况下导致包含PTX后端目标，它一次只能指定一个目标 cubin 体系结构，并且不能使用多个 -arch= 选项相同的 nvcc 命令行，这就是上面的示例显式使用 -gencode= 的原因。

2.2.2 使用CUDA Toolkit 9.0的应用程序

使用CUDA Toolkit 9.0版本，nvcc可以生成Volta体系结构（计算能力7.0）的本地cubin文件。

使用CUDA Toolkit 9.0时，为了确保nvcc将为所有最新的GPU架构以及PTX版本生成cubin文件，以便于将来的GPU体系架构进行前向兼容，可以像下面的示例一样在nvcc命令行指定适当的 -gencode= 参数。

Mac/Linux

/usr/local/cuda/bin/nvcc -gencode=arch=compute\_50,code=sm\_50 -gencode=arch=compute\_52,code=sm\_52 -gencode=arch=compute\_60,code=sm\_60 \

-gencode=arch=compute\_61,code=sm\_61 -gencode=arch=compute\_70,code=sm\_70 \

-gencode=arch=compute\_70,code=compute\_70 -O2 -o mykernel.o -c mykernel.cu

* compute\_XX 指PTX版本
* sm\_XX 指cubin版本
* nvcc的 -gencode= 命令行选项的 arch= 指定前端编译目标，并且必须始终为 PTX 版本。
* code= 指定后端编译目标，可以是 cubin 或 PTX 或两者均可。
* 只有由 code= 指定的后端目标版本将保留在结果二进制文件中，至少包含一个 PTX 以提供未来体系架构的兼容性。
* 同时，注意 CUDA 9.0 移除了对计算能力 2.x 设备的支持，任何 compute\_2x 及 sm\_2x 需要从编译选项中移除。

2.2.3 独立线程调度兼容性

Volta体系架构在线程束中引入了独立线程调度。

如果开发人员对扭曲同步性做出了假设，那么与以前的体系架构相比，此功能可以更改参与执行的代码的线程集合。

更多细节问题和正确操作请参考 **CUDA C++ 编程指南** 中的计算能力7.0部分。

为了帮助迁移，Volta开发人员可以通过下面编译选项的组合选择加入Pascal调度模型。

ncvv -arch=compute\_60 -code=sm\_70

参考

[1] [Matching SM architectures (CUDA arch and CUDA gencode) for various NVIDIA cards](https://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/)

[2] [Volta Compatibility Guide for CUDA Applications](https://docs.nvidia.com/cuda/volta-compatibility-guide/index.html)