显卡，显卡驱动,nvcc, cuda driver,cudatoolkit,cudnn到底是什么？

Posted by marsggbo on 2019-11-11 22:01

CUDA 并行计算 机器学习 NVCC CUDNN CUDATOOLKIT CUDA DRIVER 显卡驱动 显卡

在使用深度学习框架的过程中一定会经常碰到这些东西，虽然anaconda有时会帮助我们自动地解决这些设置，但是有些特殊的库却还是需要我们手动配置环境，但是我对标题上的这些名词其实并不十分清楚，所以老是被网上的教程绕得云里雾里，所以觉得有必要写下一篇文章当做笔记供之后参考。

GPU型号含义

> 参考【GPU编程系列之一】从深度学习选择什么样的gpu来谈谈gpu的硬件架构

- **显卡**： 简单理解这个就是我们前面说的**GPU**，尤其指NVIDIA公司生产的GPU系列，因为后面介绍的cuda,cudnn都是NVIDIA公司针对自身的GPU独家设计的。
- **显卡驱动**：很明显就是字面意思，通常指**NVIDIA Driver**，其实它就是一个驱动软件，而前面的**显卡**就是硬件。
- gpu架构：Tesla、Fermi、Kepler、Maxwell、Pascal

- 芯片型号：GT200、GK210、GM104、GF104等
- 显卡系列：GeForce、Quadro、Tesla
- GeForce显卡型号：G/GS、GT、GTS、GTX

gpu架构指的是硬件的设计方式，例如流处理器簇中有多少个core、是否有L1 or L2缓存、是否有双精度计算单元等等。每一代的架构是一种思想，如何去更好完成并行的思想

芯片就是对上述gpu架构思想的实现，例如芯片型号GT200中第二个字母代表是哪一代架构，有时会有100和200代的芯片，它们基本设计思路是跟这一代的架构一致，只是在细节上做了一些改变，例如GK210比GK110的寄存器就多一倍。有时候一张显卡里面可能有两张芯片，Tesla k80用了两块GK210芯片。这里第一代的gpu架构的命名也是Tesla，但现在基本已经没有这种设计的卡了，下文如果提到了会用Tesla架构和Tesla系列来进行区分。

而显卡系列在本质上并没有什么区别，只是NVIDIA希望区分成三种选择，GeFore用于家庭娱乐，Quadro用于工作站，而Tesla系列用于服务器。Tesla的k型号卡为了高性能科学计算而设计，比较突出的优点是双精度浮点运算能力高并且支持ECC内存，但是双精度能力好在深度学习训练上并没有什么卵用，所以Tesla系列又推出了M型号来做专门的训练深度学习网络的显卡。需要注意的是Tesla系列没有显示输出接口，它专注于数据计算而不是图形显示。

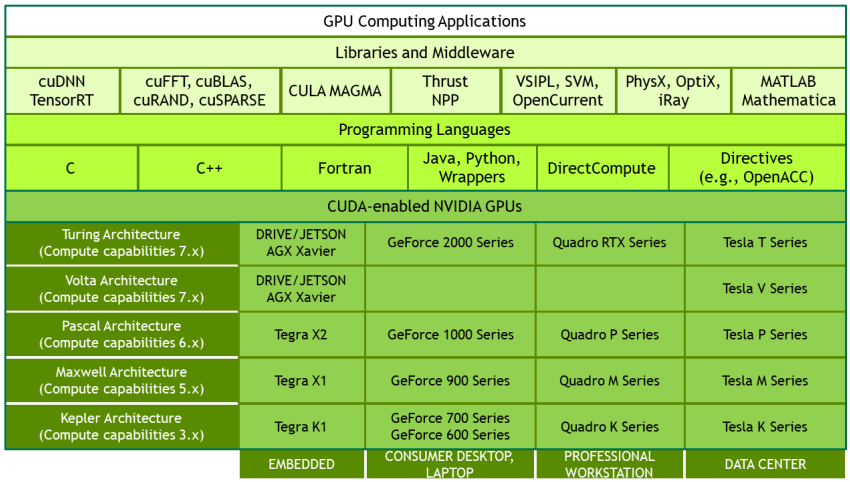
最后一个GeForce的显卡型号是不同的硬件定制，越往后性能越好，时钟频率越高显存越大，即G/GS<GT<GTS<GTX。

20 CUDA名称含义

21 CUDA

看了很多答案，有人说CUDA就是一门编程语言，像C,C++,python 一样，也有人说CUDA是API。CUDA英文全称是Compute Unified Device Architecture，是显卡厂商NVIDIA推出的运算平台。 CUDA™是一种由NVIDIA推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题。按照官方的说法是，**CUDA是一个并行计算平台和编程模型，能够使得使用GPU进行通用计算变得简单和优雅**。

MENU



2 cudnn

这个其实就是一个专门为深度学习计算设计的软件库，里面提供了很多专门的计算函数，如卷积等。从上图也可以看到，还有很多其他的软件库和中间件，包括实现c++ STL的thrust、实现gpu版本blas的cublas、实现快速傅里叶变换的cuFFT、实现稀疏矩阵运算操作的cuSparse以及实现深度学习网络加速的cuDNN等等，具体细节可参阅GPU-Accelerated Libraries

3 CUDA Toolkit

> 参考CUDA Toolkit

CUDA Toolkit由以下组件组成：

- **Compiler**: CUDA-C和CUDA-C++编译器 `NVCC` 位于 `bin/` 目录中。它建立在 `NVVM` 优化器之上，而 `NVVM` 优化器本身构建在 `LLVM` 编译器基础结构之上。希望开发人员可以使用 `nvm/` 目录下的Compiler SDK来直接针对NVVM进行开发。
- **Tools**: 提供一些像 `profiler` , `debuggers` 等工具，这些工具可以从 `bin/` 目录中获取
- **Libraries**: 下面列出的部分科学库和实用程序库可以在 `lib/` 目录中使用(Windows上的DLL位于 `bin/` 中)，它们的接口在 `include/` 目录中可获取。
    - **cudart**: CUDA Runtime
    - **cudadevrt**: CUDA device runtime
    - **cupti**: CUDA profiling tools interface
    - **nvml**: NVIDIA management library
    - **nvrtc**: CUDA runtime compilation

    - **cublas**: BLAS (Basic Linear Algebra Subprograms，基础线性代数程序集)
    - **cublas_device**: BLAS kernel interface
    - ...

- **CUDA Samples**: 演示如何使用各种CUDA和library API的代码示例。可在Linux和Mac上的 `samples/` 目录中获得，Windows上的路径是 `C:\ProgramData\NVIDIA Corporation\CUDA Samples` 中。在Linux和Mac上， `samples/` 目录是只读的，如果要对它们进行修改，则必须将这些示例复制到另一个位置。
- **CUDA Driver**: 运行CUDA应用程序需要系统至少有一个**具有CUDA功能的GPU**和**与CUDA工具包兼容的驱动程序**。每个版本的CUDA工具包都对应一个最低版本的CUDA Driver，也就是说如果你安装的CUDA Driver版本比官方推荐的还低，那么很可能会无法正常运行。CUDA Driver是向后兼容的，这意味着根据CUDA的特定版本编译的应用程序将继续在后续发布的Driver上也能继续工作。通常为了方便，在安装CUDA Toolkit的时候会默认安装CUDA Driver。在开发阶段可以选择默认安装Driver，但是对于像Tesla GPU这样的商用情况时，建议在官方安装最新版本的Driver。
  目前(2019年10月)的CUDA Toolkit和CUDA Driver版本的对应情况如下：

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.1 (10.1.105 general release, and updates)	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30

MENU

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

30 nvcc&nvidia-smi

31 nvcc

这个在前面已经介绍了， `nvcc` 其实就是CUDA的编译器,可以从CUDA Toolkit的 `/bin` 目录中获取,类似于 `gcc` 就是c语言的编译器。由于程序是要经过编译器编程成可执行的二进制文件，而cuda程序有两种代码，一种是运行在cpu上的host代码，一种是运行在gpu上的device代码，所以 `nvcc` 编译器要保证两部分代码能够编译成二进制文件在不同的机器上执行。nvcc涉及到的文件后缀及相关意义如下表

文件后缀	意义
.cu	cuda源文件，包括host和device代码
.cup	经过预处理的cuda源文件，编译选项--preprocess/-E
.c	c源文件
.cc/.cxx/.cpp	c++源文件
.gpu	gpu中间文件，编译选项--gpu
.ptx	类似汇编代码，编译选项--ptx
.o/.obj	目标文件，编译选项--compile/-c
.a/.lib	库文件，编译选项--lib/-lib
.res	资源文件
.so	共享目标文件，编译选项--shared/-shared
.cubin	cuda的二进制文件，编译选项-cubin

32 nvidia-smi

MENU

`nvidia-smi` 全程是NVIDIA System Management Interface，它是一个基于前面介绍过的 `NVIDIA Management Library(NVML)` 构建的命令行实用工具，旨在帮助管理和监控NVIDIA GPU设备。

3 nvcc和nvidia-smi显示的CUDA版本不同?

在我们实验室的服务器上 `nvcc --version` 显示的结果如下：

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Tue_Jun_12_23:07:04_CDT_2018
Cuda compilation tools, release 9.2, V9.2.148
```

而 `nvidia-smi` 显示结果如下：

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.104      Driver Version: 410.104      CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  On   | 00000000:01:00.0 Off |                  Off |
| N/A   28C    P0    26W / 250W |      0MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE...  On   | 00000000:02:00.0 Off |                  Off |
```

可以看到 `nvcc` 的CUDA 版本是9.2，而 `nvidia-smi` 的CUDA版本是10.0。很奇怪的是有时候绝大多数情况代码也能整成跑起来，stackoverflow上的一个解释如下：

CUDA有两个主要的API：**runtime(运行时) API**和**driver API**。这两个API都有对应的CUDA版本（如9.2和10.0等）。

- 用于支持**driver API**的必要文件(如 `libcuda.so`)是由**GPU driver installer**安装的。 `nvidia-smi` 就属于这一类API。
- 用于支持**runtime API**的必要文件(如 `libcudart.so` 以及 `nvcc`)是由**CUDA Toolkit installer**安装的。（CUDA Toolkit Installer有时可能会集成了GPU driver Installer）。 `nvcc` 是与CUDA Toolkit一起安装的CUDA compiler-driver tool，它只知道

它自身构建时的CUDA runtime版本。它不知道安装了什么版本的GPU driver，甚至不知道是否安装了GPU driver。

综上，如果driver API和runtime API的CUDA版本不一致可能是因为你使用的是单独的GPU driver installer，而不是CUDA Toolkit installer里的GPU driver installer。

runtime和driver API区别

下图很清楚的展示前面提到的各种概念之间的关系，其中runtime和driver API在很多情况非常相似，也就是说用起来的效果是等价的，但是你不能混合使用这两个API，因为二者是互斥的。也就是说在开发过程中，你只能选择其中一种API。简单理解二者的区别就是：runtime是更高级的封装，开发人员用起来更方便，而driver API更接近底层，速度可能会更快。

两种API详细的区别如下：

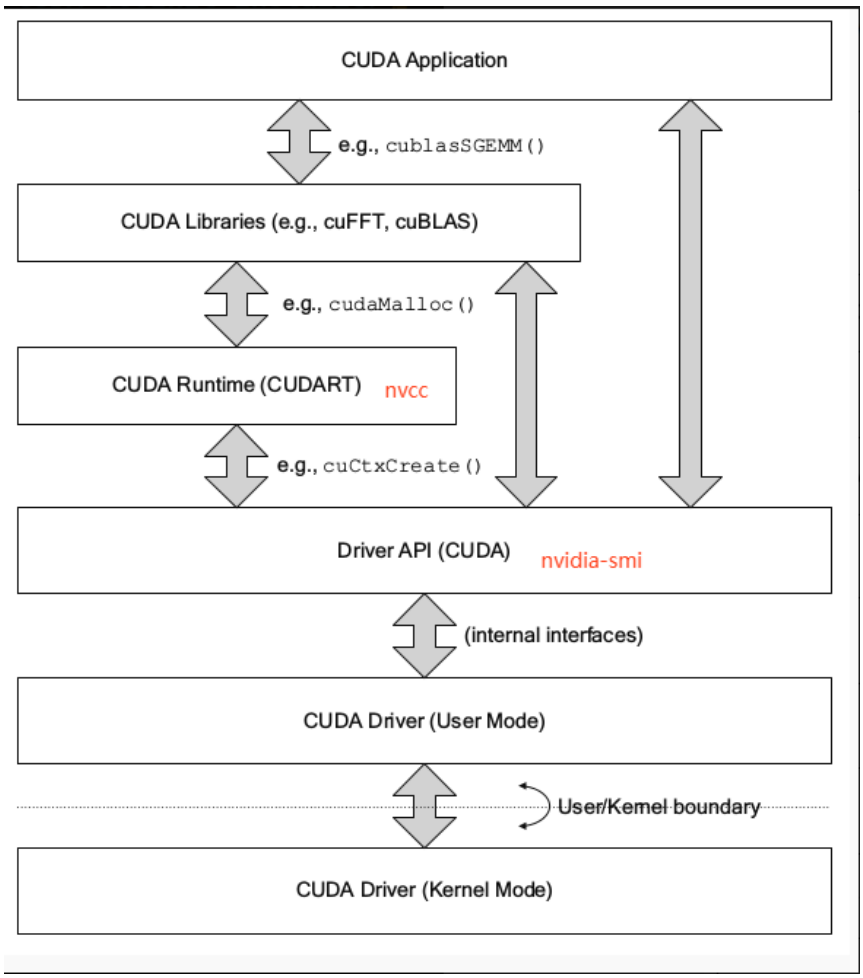
- **复杂性**
    - runtime API通过提供隐式初始化、上下文管理和模块管理来简化设备代码管理。这使得代码更简单，但也缺乏驱动程序API所具有的控制级别。
    - 相比之下，driver API提供了更细粒度的控制，特别是在上下文和模块加载方面。实现内核启动要复杂得多，因为执行配置和内核参数必须用显式函数调用指定。
- **控制**
    - 对于runtime API，其在运行时，所有内核都在初始化期间自动加载，并在程序运行期间保持加载状态。
    - 而使用driver API，可以只加载当前需要的模块，甚至动态地重新加载模块。driver API也是语言独立的，因为它只处理 `cubin` 对象。
- **上下文管理**
  上下文管理可以通过driver API完成，但是在runtime API中不公开。相反，runtime API自己决定为线程使用哪个上下文:
    - 如果一个上下文通过driver API成为调用线程的当前上下文，runtime将使用它，
    - 如果没有这样的上下文，它将使用“主上下文(primary context)”。

主上下文会根据需要创建，每个设备每个进程一个上下文，并进行引用计数，然后在没有更多的引用时销毁它们。在一个进程中，所有runtime API的用户都将共享主上下文，除非上下文已成为每个线程的当前上下文。runtime使用的上下文，即当前上下文

MENU

或主上下文，可以用 `cudaDeviceSynchronize()` 同步，也可以用 `cudaDeviceReset()` 销毁。

但是，将runtime API与主上下文一起使用会有tradeoff。例如，对于那些需要给较大的软件包写插件的开发者来说者会带来不少麻烦，因为如果所有的插件都在同一个进程中运行，它们将共享一个上下文，但可能无法相互通信。也就是说，如果其中一个在完成所有CUDA工作后调用 `cudaDeviceReset()` ，其他插件将失败，因为它们使用的上下文在它们不知情的情况下被破坏。为了避免这个问题，CUDA clients可以使用driver API来创建和设置当前上下文，然后使用runtime API来处理它。但是，上下文可能会消耗大量的资源，比如设备内存、额外的主机线程和设备上上下文切换的性能成本。当将driver API与基于runtime API(如cuBLAS或cuFFT)构建的库一起使用时，这种runtime-driver上下文共享非常重要。



5 Linux中PATH、 LIBRARY_PATH

> 参考Linux中PATH、 LIBRARY_PATH、 LD_LIBRARY_PATH的区别

5.1 PATH

MENU

PATH是可执行文件路径，是三个中我们最常接触到的，因为我们命令行中的每句能运行的命令，如ls、top、ps等，都是系统通过PATH找到了这个命令执行文件的所在位置，再run这个命令（可执行文件）。 比如说，在用户的目录 `~/mycode/` 下有一个bin文件夹，里面放了有可执行的二进制文件、shell脚本等。如果想要在任意目录下都能运行上述bin文件夹的可执行文件，那么只需要把这个bin的路径添加到PATH即可，方法如下：

```
# vim ~/.bashrc
PATH=$PATH:~/mycode/bin
```

5.2 LIBRARY_PATH和LD_LIBRARY_PATH

这两个路径可以放在一起讨论，

- `LIBRARY_PATH` 是**程序编译期间**查找动态链接库时指定查找共享库的路径
- `LD_LIBRARY_PATH` 是**程序加载运行期间**查找动态链接库时指定除了系统默认路径之外的其他路径

两者的共同点是库，库是这两个路径和PATH路径的区别，PATH是可执行文件。

两者的差异点是使用时间不一样。一个是编译期，对应的是开发阶段，如gcc编译；一个是加载运行期，对应的是程序已交付的使用阶段。

配置方法也是类似：

```
export  LD_LIBRARY_PATH=LD_LIBRARY_PATH:XXXX
```

6.0 多版本CUDA切换

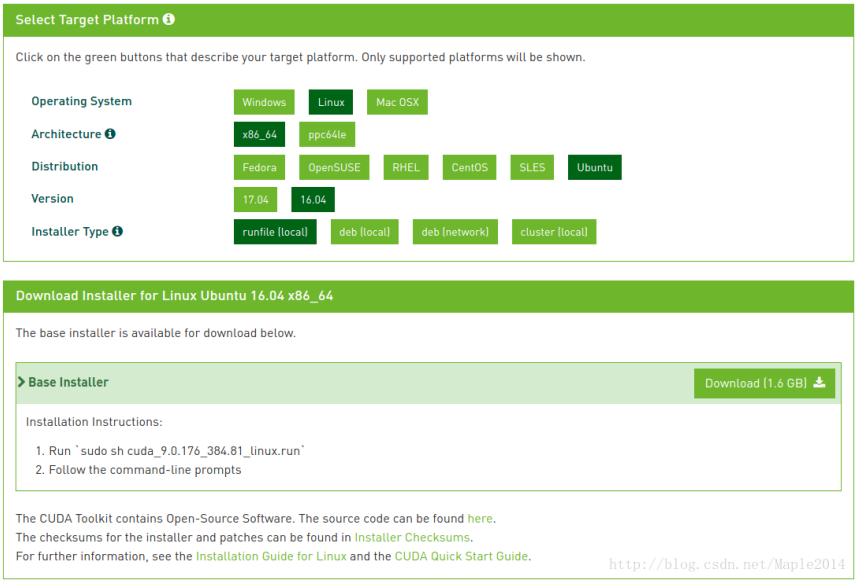
> 参考安装多版本cuda，多版本之间切换

基于前面几个小节的介绍，现在再来介绍如何管理多版本CUDA就会好懂很多了。

6.1 cuda 的下载与安装方法选择

到 CUDA Toolkit Download 下载所需版本，以cuda_9.0.176_384.81_linux.run为例：

MENU



建议选择使用 `.run` 文件安装，因为使用 `.deb` 可能会将已经安装的较新的显卡驱动替换。

2 cuda 安装

进入到放置 `cuda_9.0.176_384.81_linux.run` 的目录：

```
sudo chmod +x cuda_9.0.176_384.81_linux.run # 为 cuda_9.0.176_384.81_linux.run 添加可执行权限
./cuda_9.0.176_384.81_linux.run # 安装 cuda_9.0.176_384.81_linu
```

在安装过程中截取其中比较重要的几个选择：

```
Do you accept the previously read EULA?
accept/decline/quit: accept

Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 384.81?
(y)es/(n)o/(q)uit: n # 如果在这之前已经安装好更高版本的显卡驱动就不需要再重复安装，如果需要重复安装就选择 yes,此外还需要关闭图形界面。

Install the CUDA 9.0 Toolkit?
(y)es/(n)o/(q)uit: y

Enter Toolkit Location
 [ default is /usr/local/cuda-9.0 ]: # 一般选择默认即可，也可以选择安装在其他目录，在需要用的时候指向该目录或者使用软连接 link 到 /usr/local/cuda。

/usr/local/cuda-9.0 is not writable.
Do you wish to run the installation with 'sudo'?
(y)es/(n)o: y

Please enter your password:
Do you want to install a symbolic link at /usr/local/cuda? # 是否将安装目录通过软连接的方式 link 到 /usr/local/cuda，yes or no 都可以，取决于你是否使用 /usr/local/cuda 为默认的 cuda 目录
```

前面选择的一些汇总：

MENU

```
Driver:   Not Selected
Toolkit:  Installed in /usr/local/cuda-9.0
Samples:  Not Selected

Please make sure that
 -   PATH includes /usr/local/cuda-9.0/bin
 -   LD_LIBRARY_PATH includes /usr/local/cuda-9.0/lib64, or, add /usr/local/cuda-9.0/lib64 to /etc/ld.so.conf and run ldconfig as root

To uninstall the CUDA Toolkit, run the uninstall script in /usr/local/cuda-9.0/bin

Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-9.0/doc/pdf for detailed information on setting up CUDA.

***WARNING: Incomplete installation! This installation did not
```

安装完成后可以在 /usr/local 目录下看到：

```
cuda-8.0 # 笔者之前安装的cuda-8.0
cuda-9.0 # 刚刚安装的cuda-9.0
cuda # cuda-8.0 的软连接
```

多个 cuda 版本之间进行切换

将 `~/.bashrc` 或 `~/.zshrc` 下与cuda相关的路径都改为 `/usr/local/cuda/` 而不使用 `/usr/local/cuda-8.0/` 或 `/usr/local/cuda-9.0/` 。

```
#在切换cuda版本时
rm -rf /usr/local/cuda#删除之前创建的软链接
sudo ln -s /usr/local/cuda-8.0/ /usr/local/cuda/
nvcc --version #查看当前 cuda 版本

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Mon_Jan_23_12:24:11_CST_2017
Cuda compilation tools, release 8.0, V8.0.62

#cuda8.0 切换到 cuda9.0
rm -rf /usr/local/cuda
sudo ln -s /usr/local/cuda-9.0/ /usr/local/cuda/
nvcc --version
```

这篇文章花了我一天的时间来整理，所以如果觉得有帮助给个赞可好。