精简CUDA教程-DriverAPI

咕泡出品 必属精品

cuInit - 驱动初始化

1. cuInit的意义是，初始化驱动API，如果不执行，则所有API都将返回错误，全局执行一次即可
2. 没有对应的cuDestroy，不需要释放，程序销毁自动释放

咕泡出品 必属精品

返回值检查

1. 正确友好的检查cuda函数的返回值，有利于程序的组织结构
2. 使得代码可读性更好，错误更容易发现

咕泡出品 必属精品

CUcontext

1. context是一种上下文，关联对GPU的所有操作
2. context与一块显卡关联，一个显卡可以被多个context关联
3. **每个线程都有一个栈结构储存context**，栈顶是当前使用的context，对应有push、pop函数操作context的栈，所有api都以当前context为操作目标
4. 试想一下，如果执行任何操作你都需要传递一个device决定送到哪个设备执行，得多麻烦

```
cuMalloc(device, &ptr, 100);
cuFree(device, ptr);
cuMemcpy(device, dst, src, 100);
```

没有context的代码

```
cuCreateContext(device, &context);
cuPushCurrent(context);
cuMalloc(&ptr, 100);
cuFree(ptr);
cuMemcpy(dst, src, 100);
cuPopCurrent(context);
```

有context的代码

context只是为了方便控制device的一种手段而提出来的

栈的存在是为了方便控制多个设备

必属精品 咕泡出品

CUcontext

1. 由于高频操作，是一个线程基本固定访问一个显卡不变，且只使用一个context，很少会用到多context
2. CreateContext、PushCurrent、PopCurrent这种多context管理就显得麻烦，还得再简单
3. 因此推出了cuDevicePrimaryCtxRetain，为设备关联主context，分配、释放、设置、栈都不用你管
4. primaryContext：给我设备id，给你context并设置好，此时**一个显卡对应一个primary context**
5. 不同线程，只要设备id一样，primary context就一样。context是线程安全的

```
cuCreateContext(device, &context);
cuPushCurrent(context);
cuMalloc(&ptr, 100);
cuFree(ptr);
cuMemcpy(dst, src, 100);
cuPopCurrent(context);
```

有context的代码

runtimeAPI自动使用cuDevicePrimaryCtxRetain

```
cuDevicePrimaryCtxRetain(device, &context);
cuMalloc(&ptr, 100):
cuFree(ptr);
cuMemcpy(dst, src, 100);
```

不用管理context栈的代码

谢谢!

咕泡出品 必属精品