caffe中的主存模型

为什么在caffe这边会涉及到主存模型呢？最重要的原因是由于我们的深度学习模型是可以跑在GPU上的。但是这样会就会有一个比较难受的问题，Host端的内存和Device端上的内存并不是同一块的，并不能统一寻址的。因此有些数据我们可能既要在Host端的内存上面持有一份，同时也要在Device端的内存上持有一份。如果某一端的数据发生了变化，我们也应该提供一个机制让另一端的数据更新，（不是实时更新，而是在我们需要用到的时候在进行更新）。这样我们可以确保访问的的数据在CPU和GPU上一致的。在caffe中提供了SyncedMemory类来具体来管理主存一致性的问题，我们可以具体看一下吧。

SyncedMemory中host主存和device主存同步问题

首先介绍一下两个cuda API，cudaMallocHost和cudaFreeHost，通过这两个API就可以在Host端分配锁页内存，这里简单的解释一下锁页内存，锁页内存意思是这一块内存不会与磁盘进行数据交换了。我们在操作系统课程中学习的内存一般是可分页的，就是内存会磁盘进行交互，利用锁页内存可以提高Host和Device交换数据的效率。但同时锁页内存也是一种较为稀有的资源，在使用的时候要注意这一点，如果锁住的内存块太大了会导致Host端其他程序分配不到内存，因而可能会影响其他程序的效率，那我们看一下SyncedMemory如何包装这两个函数的吧：

(include/caffe/syncedmem.hpp)

```
inline void CaffeMallocHost(void** ptr, size_t size, bool* use_cuda) {   //分配host端内存
#ifndef CPU_ONLY
    if (Caffe::mode() == Caffe::GPU) {
        CUDA_CHECK(cudaMallocHost(ptr, size));
        *use_cuda = true;
        return;
    }
#endif
    *ptr = malloc(size);
    *use_cuda = false;
    CHECK(*ptr) << "host allocation of size " << size << " failed";
}

inline void CaffeFreeHost(void* ptr, bool use_cuda) {     //回收host端内存
#ifndef CPU_ONLY
    if (use_cuda) {
        CUDA_CHECK(cudaFreeHost(ptr));
        return;
    }
#endif
free(ptr)
}
```

在上面的代码中我去除了和MKL相关的代码，因为这个不会去分析它。我们可以看到如果是GPU模式就调用cuda API来分配内存，如果是CPU模式的话则调用malloc等系统函数来分配内存。

CUDA内存优化

使用统一内存寻址指针

Blob类管理数据