Caffe源码解析2：SycedMem

转载请注明出处，楼燚(yì)航的blog，**http://www.cnblogs.com/louyihang loves baiyan/**

看到SyncedMem就知道，这是在做内存同步的操作。这类个类的代码比较少，但是作用是非常明显的。文件对应着syncedmem.hpp,着syncedmem.cpp

首先是两个全局的内联函数。如果机器是支持GPU的并且安装了cuda，通过cudaMallocHost分配的host memory将会被pinned，这里我谷歌了一下，pinned的意思就是内存不会被paged out，我们知道内存里面是由页作为基本的管理单元。分配的内存可以常驻在内存空间中对效率是有帮助的，空间不会被别的进程所抢占。同样如果内存越大，能被分配的Pinned内存自然也越大。还有一点是，对于单一的GPU而言提升并不会太显著，但是对于多个GPU的并行而言可以显著提高稳定性。

这里是两个封装过的函数，内部通过cuda来分配主机和释放内存的接口

```
inline void CaffeMallocHost(void** ptr, size_t size, bool* use_cuda) {
#ifndef CPU_ONLY
  if (Caffe::mode() == Caffe::GPU) {
    CUDA_CHECK(cudaMallocHost(ptr, size));// GPU模式下cuda分配内存
    *use_cuda = true;
    return;
  }
#endif
  *ptr = malloc(size);//如果没有cuda则通过c的malloc函数分配
  *use_cuda = false;
  CHECK(*ptr) << "host allocation of size " << size << " failed";
}

inline void CaffeFreeHost(void* ptr, bool use_cuda) {
#ifndef CPU_ONLY
  if (use_cuda) {
    CUDA_CHECK(cudaFreeHost(ptr));//cuda的主机内存释放操作
    return;
  }
#endif
  free(ptr);//c的释放操作
}
```

SyncedMemory类，首先是构造函数和析构函数

```
class SyncedMemory {
 public:
  SyncedMemory() //参数构造函数，负责初始化
      : cpu_ptr_(NULL), gpu_ptr_(NULL), size_(0), head_(UNINITIALIZED),
        own_cpu_data_(false), cpu_malloc_use_cuda_(false), own_gpu_data_(false),
        gpu_device_(-1) {}
  explicit SyncedMemory(size_t size)//带explicit关键字的，单个参数构造函数，explicit禁止单参数构造函数的隐式转换
      : cpu_ptr_(NULL), gpu_ptr_(NULL), size_(size), head_(UNINITIALIZED),
        own_cpu_data_(false), cpu_malloc_use_cuda_(false), own_gpu_data_(false),
        gpu_device_(-1) {}
  ~SyncedMemory();//其在析构时调用的也是CaffeFreeHost
```

这几个函数分别是

```
  const void* cpu_data();
  void set_cpu_data(void* data);
  const void* gpu_data();
  void set_gpu_data(void* data);
```

cpu_data()主要是获得cpu上data的地址，set_cpu_data是将cpu的data指针指向一个新的区域由data指针传入，并且将原来申请的内存释放。下面两个同理，分别是获得gpu数据地址和set gpu数据地址。

```
  void* mutable_cpu_data();
```

```
  void* mutable_gpu_data();
  enum SyncedHead { UNINITIALIZED, HEAD_AT_CPU, HEAD_AT_GPU, SYNCED };
  SyncedHead head() { return head_; }
  size_t size() { return size_; }
```

前两个分别是返回cpu和gpu上的data指针，并且置状态为 `head_ = HEAD_AT_CPU` 和响应的gpu版本。SyncedHead主要是个枚举类型，用来设定head_的状态，head()函数即返回相应的数据状态，而size()函数返回数据大小

```
#ifndef CPU_ONLY
  void async_gpu_push(const cudaStream_t& stream);
#endif
```

这是一个cuda拷贝的异步传输，从数据从cpu拷贝到gpu，异步传输是已经假定caller会在使用之前做同步操作。

```
 private:
  void to_cpu();
  void to_gpu();
  void* cpu_ptr_;
  void* gpu_ptr_;
  size_t size_;
  SyncedHead head_;
  bool own_cpu_data_;
  bool cpu_malloc_use_cuda_;
  bool own_gpu_data_;
  int gpu_device_;

  DISABLE_COPY_AND_ASSIGN(SyncedMemory);//禁止该类的拷贝与赋值
};  // class SyncedMemory
```

其实这里的东西也不多了，to_cpu()，to_gpu()这个看名字就知道了，需要注意的是，如果head 是未被初始化的状态，那么首先需要先分配内存，这个根据cpu和gpu视情况而定，之后再将数据从cpu或者gpu拷贝到另一处。之后函数会重新标记Head的状态，数据是否在cpu或者在gpu中,cpu这里是简称，其实是主机。cpu_ptr和gpu_ptr分别是在cpu和gpu中的数据指针，size_这就不再说了,head_之前也液晶提到过了，后面都是几个相应的标记为，以及gpu的ID号

楼燚航的blog
关注 - 1
粉丝 - 60
+加关注

« 上一篇：Caffe源码解析1：Blob
» 下一篇：Caffe源码解析3：Layer

posted @ 2016-01-22 10:58 楼燚航的blog 阅读(370) 评论(0) 编辑 收藏

--gaosi123

2. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

BTW, 我电脑是i7 4790k + 16GB内存，所以硬件设备应该不会是限制。不知道问题出在哪里

--gaosi123

3. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

如果可以，欢迎留个email

--gaosi123

4. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

你好，我也是直接把DPM代码拷贝到工程里，但是想你这样直接拷进去不会报错吗？我直接拷贝进去按照你的来，报错信息如下：Error 4 error C2039: 'dpm' : is not a memb.....

--gaosi123

5. Re:Fast RCNN 训练自己数据集 (2修改数据读取接口)

@楼燚航的blog楼主你好！我在EdgeBoxes提取OP的时候也是直接用的默认参数，并且将坐标[x y w h]变成了左上右下的形式，但是发现检测车的时候效果并没有Selective Search好，.....

–JustJay

阅读排行榜

1. Fast RCNN 训练自己数据集 (2修改数据读取接口)(3946)
2. Fast RCNN 训练自己数据集 (1编译配置)(3166)
3. Fast RCNN 训练自己的数据集（3训练和检测）(3097)
4. RCNN (Regions with CNN) 目标物检测 Fast RCNN的基础(2096)
5. Hog SVM 车辆 行人检测(979)

评论排行榜

1. Fast RCNN 训练自己数据集 (2修改数据读取接口)(22)
2. Fast RCNN 训练自己数据集 (1编译配置)(21)
3. Fast RCNN 训练自己的数据集（3训练和检测）(5)
4. opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）(4)
5. DPM检测模型 训练自己的数据集 读取接口修改(2)

推荐排行榜

1. Fast RCNN 训练自己数据集 (2修改数据读取接口)(5)
2. 车脸检测 Adaboost 检测过程(3)
3. Caffe 抽取CNN网络特征 Python(2)
4. DPM检测模型 训练自己的数据集 读取接口修改(2)
5. RCNN (Regions with CNN) 目标物检测 Fast RCNN的基础(2)

最新知识库文章:

· 我是一个线程

· 为什么未来是全栈工程师的世界？

· 程序bug导致了天大的损失，要枪毙程序猿吗？

· 如何运维千台以上游戏云服务器

· 架构漫谈（一）：什么是架构？

» 更多知识库文章...