**clCaffe 安装和运行**

1. **硬件的选择**

* 支持5代(Haswell) 后的所有intel 核显.
* 必须要配置双通道 (2片内存条, 不然速度降低至少50%)
* 本文的测试是基于**（Intel）Skull CanyonNUC6i7KYK I7-6770HQ**

1. **Opencl驱动的安装**

* 安装必要的包 sudo apt-get install clinfo ocl-icd-opencl-dev opencl-headers ocl-icd-libopencl1
* 升级kernel到4.7   
  官网地址: http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.7.10/  
  下载3个包: amd64的 非lowlatency的, 下面地址可能会有更新  
  <http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.7.10/linux-headers-4.7.10-040710_4.7.10-040710.201610220847_all.deb>  
  <http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.7.10/linux-headers-4.7.10-040710-generic_4.7.10-040710.201610220847_amd64.deb>  
  <http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.7.10/linux-image-4.7.10-040710-generic_4.7.10-040710.201610220847_amd64.deb>  
  安装这3个包, sudo dpkg -i \*\*\*.deb 注意有安装顺序  
  sudo reboot
* 安装官网opencl驱动 : <https://software.intel.com/en-us/articles/opencl-drivers>  
  下载驱动<http://registrationcenter-download.intel.com/akdlm/irc_nas/11396/SRB4.1_linux64.zip>  
  然后执行  
  mkdir intel-opencl  
  tar -C intel-opencl -Jxf intel-opencl-r4.1-BUILD\_ID.x86\_64.tar.xz  
  tar -C intel-opencl -Jxf intel-opencl-devel-r4.1-BUILD\_ID.x86\_64.tar.xz  
  tar -C intel-opencl -Jxf intel-opencl-cpu-r4.1-BUILD\_ID.x86\_64.tar.xz  
  sudo cp -R intel-opencl/\* /  
  sudo ldconfig   
  注意之前一定要看看 /etc/Opencl/vendors 下面是否有其它的.icd文件, 确保要删除其它文件.   
  执行clinfo 查看opencl驱动为2个, 其中有一个type为GPU

Device Name Intel(R) HD Graphics

Device Vendor Intel(R) Corporation

Device Vendor ID 0x8086

Device Version OpenCL 2.0

Driver Version r4.1.61547

Device OpenCL C Version OpenCL C 2.0

Device Type GPU

Device Profile FULL\_PROFILE

1. **安装python环境(开发环境**conda下<https://www.continuum.io/downloads/> 本机下载linux 2.7 64bit的安装包, 安装后, 选择自动填写环境变量到bashrc  
   更新opencv到 3.2 conda install -c menpo opencv3  
   然后更新conda install libgcc  
   (可以更新到国内源   
   conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/  
   conda config --set show\_channel\_urls yes)  
   可以参考<http://blog.csdn.net/awakeljw/article/details/54021602>  
     
   如果使用Ubuntu自带的python, 请使用example的脚本安装
2. **安装依赖包 (Ubuntu16.04)**sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler

sudo apt-get install --no-install-recommends libboost-all-dev  
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev  
sudo apt-get install libstdc++6

1. **源码的编译**

* 参考<https://github.com/01org/caffe/wiki/clCaffe> 编译依赖  
    
  使用MKL, 性能对clCaffe基本无提升

mkdir -p $HOME/code

cd $HOME/code

git clone https://github.com/viennacl/viennacl-dev.git

cd viennacl-dev

mkdir build && cd build

cmake -DBUILD\_TESTING=OFF -DBUILD\_EXAMPLES=OFF -DCMAKE\_INSTALL\_PREFIX=$HOME/local - DOPENCL\_LIBRARY=/opt/intel/opencl/libOpenCL.so ..

make -j4

make install

cd $HOME/code

git clone https://github.com/intel/isaac

cd isaac

mkdir build && cd build

cmake -DCMAKE\_INSTALL\_PREFIX=$HOME/local .. && make -j4

make install

* 编译clCaffe (make –j32 2>&1 |tee log 可以看make的log)

mkdir -p $HOME/code

cd $HOME/code

git clone <https://github.com/01org/caffe.git>

cd caffe

git checkout inference-optimize

mkdir build && cd build

export ISAAC\_HOME=$HOME/local

cmake .. -DUSE\_GREENTEA=ON -DUSE\_CUDA=OFF -DUSE\_INTEL\_SPATIAL=ON -DBUILD\_docs=0 -DUSE\_ISAAC=ON -DViennaCL\_INCLUDE\_DIR=$HOME/local/include -DOPENCL\_LIBRARIES=/opt/intel/opencl/libOpenCL.so -DOPENCL\_INCLUDE\_DIRS=/opt/intel/opencl/include

make all -j4

* 测试 (skylake i7 + Intel Iris Pro Graphics 580 (Skylake GT4e)) 注意第一次会慢一点  
  ./build/tools/caffe time -model ./models/bvlc\_alexnet/deploy.prototxt -gpu 0 -phase TEST  
  I0407 15:05:43.975138 2473 caffe.cpp:483] Average Forward pass: 37.3194 ms.  
  I0407 15:05:43.975147 2473 caffe.cpp:485] Average Backward pass: 0 ms.  
  I0407 15:05:43.975152 2473 caffe.cpp:487] Average Forward-Backward: 37.6609 ms.  
  I0407 15:05:43.975172 2473 caffe.cpp:489] Total Time: 1883.04 ms.  
    
  ./build/tools/caffe time -model ./models/inference\_optimize/AlexNet-merged-1.prototxt -gpu 0 -phase TEST  
  I0407 15:54:03.365612 3155 caffe.cpp:483] Average Forward pass: 14.794 ms.  
  I0407 15:54:03.365622 3155 caffe.cpp:485] Average Backward pass: 0 ms.  
  I0407 15:54:03.365625 3155 caffe.cpp:487] Average Forward-Backward: 15.082 ms.  
  I0407 15:54:03.365630 3155 caffe.cpp:489] Total Time: 754.1 ms.  
    
  ./build/tools/caffe time -model ./models/bvlc\_googlenet/deploy.prototxt -gpu 0 -phase TEST 注意这里第一次时间会非常长  
  I0407 16:16:30.982638 3611 caffe.cpp:483] Average Forward pass: 86.3635 ms.  
  I0407 16:16:30.982668 3611 caffe.cpp:485] Average Backward pass: 0 ms.  
  I0407 16:16:30.982686 3611 caffe.cpp:487] Average Forward-Backward: 86.6913 ms.  
  I0407 16:16:30.982702 3611 caffe.cpp:489] Total Time: 4334.56 ms.
* FP16的执行方式: 使用caffe-fp16.bin而不是caffe 就是fp16的执行方式  
  ./build/tools/caffe-fp16.bin time -model ./models/bvlc\_googlenet/deploy.prototxt -gpu 0 -phase TEST  
  ---- 如果模型中含有BatchNorm, 做了归一化, 那么fp16基本没有精度损失~=0  
  ----- 否则fp16 目前可能会有比较大的精度损失, 比如SSD检测, VGG精度从0.77降低到0.68, 但是mobilenet+SSD就没有精度损失. 我们计划开发一个模型处理工具, 来给参数归一化  
  ---- 一般来说 fp16会比fp32 快接近一倍 , 但是如果模型中大量的1\*1 kernel, 目前优化没没有很好, fp16此时提升性能并不多, 正在优化中
* 如何内嵌clCaffe到自己的工程内, 请参考<https://github.com/liyuming1978/caffe_example> 里面有fp16和fp32 以及cmake脚本
* Cache机制 (新版本)  
  export VIENNACL\_CACHE\_PATH=${HOME}/.cache/viennacl 不加这个可能会crash  
  然后运行模型, 生产的时候把cache下的viennacl\*\*\*和viennacl目录拷贝到生产机器, 在bashrc 里面加入export VIENNACL\_CACHE\_PATH=${HOME}/.cache/clcaffe\_cache/viennacl/
* 模型fuse的方法: python tools/inference-optimize/model\_fuse.py --indefinition yolo416/yolo\_deploy.prototxt --outdefinition yolo416/fused\_yolo\_deploy.prototxt --inmodel yolo416/yolo.caffemodel --outmodel yolo416/fused\_yolo.caffemodel  
  模型fuse 如果报错, 请email我们  
  使用fuse后的prototxt必须对应使用fuse后的model才能正确运行,如果仅仅是caffe time 测测时间, 可以加--proto\_only 仅仅fuse prototxt  
  ------ fuse可以增加5~15%的性能提升  
  ------ fuse对计算结果和精度没有任何影响
* BKM:
  + 使用gpu的时候不能用set\_cpu\_data, 必须要memcpy(net->input\_blobs()[0]->mutable\_cpu\_data(), xxx…)
  + CHECK\_EQ(size\_ % 2, 1)<< "LRN only supports odd values for local\_size"; 新版本caffe添加的限制, 如果是ACROSS\_CHANNELS, 是可以去除这个限制的
  + 自写的cudnn转到caffe需要注意 pooling层有区别, 需要修改PoolingLayer<Dtype>::Reshape 函数内的ceil函数为floor函数
  + Mobilenet-ssd wiki: https://github.com/listenlink/caffe/wiki/MobileNet-SSD-on-clCaffe