clCaffe

Zhigang Gong edited this page 3 days ago · 33 revisions

Caffe's OpenCL backend for Intel

This repo contains an OpenCL backend for Caffe from Intel. Besides seamless integration of OpenCL support to original Caffe framework, this backend also provides the following major features:

- Implementation of two types of convolution engines in the forward pass: GEMM based and the Intel spatial convolution. The best option for Intel platform is the intel spatial convolution engine which is much faster than the default one.
- Runtime auto-tuner in spatial domain convolution to generate best kernels according to HW config.
- Switch among OpenCL, CPU and CUDA implementations at runtime.
- It is tested on Intel, AMD and NVidia, with support to OpenCL 1.1 and up.

The above features are already in opencl branch which is the staging branch for the upstream BVLC caffe's opencl branch. The most active branch is inference-optimzie branch which has the following additional features:

- FP16 support. Now FP16 is the first class data type in inference-optimization branch.
- Layer fusion support for inference only, currently the following combinations are supported. Please refer the following section for detail usage of the layer fusion feature.
- SSD support.
- Yolo2 support.

Dependencies

- OpenCL runtime and driver, please see *Usage* section for details
- ISAAC. The upstream version is at https://github.com/ptillet/isaac. Please use the git master version.
- Viennacl latest git master version which is at: https://github.com/viennacl/viennacl-dev. Please make sure to use the latest git master version, otherwise, the kernel cache mechanism will be broken.

Compile & install prerequisites

Make sure you have installed all the regular Caffe dependencies as described here. clCaffe depends on ViennaCL and ISAAC for optimal performance.

```
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 -
make -j4
make install
cd $HOME/code
git clone https://github.com/intel/isaac
cd isaac
```

Clone this wiki locally

https://github.com/01org/

```
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX=$HOME/local .. && make -j4
make install
```

Optional, but recommended, is to install MKL. Otherwise you'll have to install Atlas or OpenBLAS to install Caffe. Install MKL for Ubuntu from http://softwareproducts.intel.com/ILC/. Before compiling clCaffe, export LD_LIBRARY_PATH to indicate MKL directory export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/mkl/lib/intel64_lin/

```
cd $HOME/code
git clone https://github.com/01org/caffe clcaffe
cd clcaffe
git checkout inference-optimize
mkdir build && cd build
export ISAAC_HOME=$HOME/local
cmake .. -DUSE_GREENTEA=ON -DUSE_CUDA=OFF -DUSE_INTEL_SPATIAL=ON -DBUILD_docs=0 -DUS
make -j4
export CAFFE_ROOT=$HOME/code/clcaffe
```

Make sure that Caffe is finding your Gen device:

```
#:~/code/clcaffe/build (opencl)$ ./tools/caffe device_query -gpu all
I0914 10:48:49.130982   890 common.cpp:373] Total devices: 2
I0914 10:48:49.131080   890 common.cpp:374] CUDA devices: 0
I0914 10:48:49.131099   890 common.cpp:375] OpenCL devices: 2
I0914 10:48:49.131101   890 common.cpp:399] Device id:                   0
I0914 10:48:49.131103   890 common.cpp:401] Device backend:              OpenCL
I0914 10:48:49.131130   890 common.cpp:403] Backend details:             Intel(R)
I0914 10:48:49.131134   890 common.cpp:405] Device vendor:               Intel(R)
I0914 10:48:49.131155   890 common.cpp:407] Name:                        Intel(R)
I0914 10:48:49.131157   890 common.cpp:409] Total global memory:         268789514
I0914 10:48:49.131160   890 common.cpp:399] Device id:                   1
I0914 10:48:49.131178   890 common.cpp:401] Device backend:              OpenCL
I0914 10:48:49.131182   890 common.cpp:403] Backend details:             Intel(R)
I0914 10:48:49.131188   890 common.cpp:405] Device vendor:               Intel(R)
I0914 10:48:49.131192   890 common.cpp:407] Name:                        Intel(R)
I0914 10:48:49.131283   890 common.cpp:409] Total global memory:         336091750
```

Test your install by benchmarking the Alexnet example:

```
./tools/caffe time -model ../models/bvlc_alexnet/deploy.prototxt -gpu 0
```

Thanks for @ngaloppo to prepare the detail setp by step instructions to prepare and compile clcaffe.

OpenCL drivers for Intel platforms

For 4th,5th,6th or 7th generation Intel Cores and Intel® Xeon® v3, or Intel® Xeon® v4 processor. We recommend the driver at the following link: https://software.intel.com/en-us/articles/opencl-drivers#latest_linux_driver. For 3th generation cores and atom, we recommend Beignet: https://www.freedesktop.org/wiki/Software/Beignet/.

Auto-tuning and kernel cache mechanism

The convolution kernel applies auto-tuner mechanism to tune a best kernel for current parameters then store the result to a specified cache directory. Thus at the first run, it will take relatively long time to perform the auto-tuning process. At the second run, it will get the result from the cache subdirectory directly. By default it will use $HOME/.cache/clcaffe as the cache directory for the auto-tuned configurations. If you specified the VIENNACL_CACHE_PATH as below:

```
# export VIENNACL_CACHE_PATH=${HOME}/.cache/viennacl/
```

Then all the program binary will be cached as well, and the auto-tuned configurations will be stored at ${HOME}/.cache/viennacl/clcaffe/spatialkernels If the VIENNACL_CACHE_PATH is empty, the binary kernel/program cache mechanism will be disabled. It is recommended to set VIENNACL_CACHE_PATH as it will reduce most of the overhead of the kernel building which may take severa seconds at the begging.

Inference optimization

Please be noted that all the following features are in inference-optimization branch only.

Layer fusion.

ClCaffe is to use a python script to pre-process a topology file and the model parameter file to generate a fused topology file and if needed a fused model parameter. For example, we use the following command to fuse a yolo2 net model:

```
# sudo pip install scikit-image
# sudo pip install protobuf
# export PYTHONPATH=${CAFFE_ROOT}/python/
# python tools/inference-optimize/model_fuse.py --indefinition models/yolo/yolo416/y
```

After that, you can use the fused_yolo_deploy.protoxt and fused_yolo.caffemodel to do inference which is much faster than the original version.

FP16 support.

The FP16(half) data type is the first class data type now. You can refer the tools/caffe-fp16.cpp for how to use FP16. In one word, you just need to specifiy half rather than float when you create the net model.

SSD model support.

The SSD implementation is from https://github.com/weiliu89/caffe/tree/ssd. Please refer the corresponding wiki page to generate SSD model and get the pretrained parameters. You can also use fusion to accelerate the inference speed.

Yolo2 model support.

Yolo2 models are in models/yolo directly. You need to download https://pjreddie.com/media/files/yolo-voc.weights firstly. And then use the conversion script to convert it to a caffe topology.

```
wget https://pjreddie.com/media/files/yolo-voc.weights -O models/yolo/yolo416/yolo.w
python models/yolo/convert_yolo_to_caffemodel.py
```

Then you will get a caffemodel models/yolo/yolo416/yolo.caffemodel. You can use the fusion script to process it further as below:

```
python tools/inference-optimize/model_fuse.py --indefinition yolo416/yolo_deploy.pro
```

Then you can use this fused model to detect an image. Use the FP16 version:

```
build/examples/yolo/yolo_detect-fp16.bin models/yolo/yolo416/fused_yolo_deploy.proto
```

Use the FP32 version:

```
build/examples/yolo/yolo_detect models/yolo/yolo416/fused_yolo_deploy.prototxt model
```

You can also use the following command to process a video:

```
build/tools/caffe-fp16.bin  test -model models/yolo/yolo416/yolo_fused_test.prototxt
```