Caffe on ARM
v1.2

Ed Jaras
edvin.jaras@gmail.com

26/07/2017

Abstract

A comparison of the performance gains provided by the ARM Compute Library via ASIMD and Mali GPU acceleration.

Introduction

ACL

In March 2017 ARM released their Compute Library publicly, which is meant to replace the old ARM CV interface, and allow abstracted access to features like ASIMD or NEON, and GPGPU on Mali via OpenCL.[1]

In terms of CPU optimizations, it offers implementations of many basic algebraic functions, and matrix multiplications in a SIMD manner. For example it supports executing the common BLAS functions of HGEMM and SGEMM on NEON or ASIMD.

caffeOnACL

OpenAI Labs have worked on an implementation of Caffe on top of the ARM Compute Library, which replaces some of the Caffe layers with ACL versions. It is worth noting that not all mathematical operations are supported to be run via ACL, and some end up falling back on the BLAS implementations. The framework also provides the means to bypass some ACL layers and fallback onto the regular Caffe implementations if desired, and the user guide

example even shows how the best performance can be achieved via mixing ACL and base Caffe layers.[2]

This implementation does not include backwards propagation on ACL - all of the layers just raise a "Not Implemented" exception if you try. I expect the workflow is meant to be to train on a desktop-grade GPU, or possibly a cluster, and then run classification on Mali on the go.

Set-up

Hardware

All tests were run on a HiKey960 board. It has a Cortex-A73 and Cortex-A53 in a big.LITTLE configuration, along with a Mali-G71 MP8 GPU and 3GB of RAM. The SoC is similar to the Exynos 9 used by the Samsung Galaxy S8, except with a slightly different big core, and a MP8 instead of a MP20 GPU, which is why I believe it is a good reference of what high-end phones can achieve right now.[3]

Software

The driver version was Bifrost r6p0-01rel0 (UK version 10.6).
The OpenCL library version was 1.2.
The OS was Debian 9.
The ACL checkout was `db0e610fa90a4c7c534049792ee7fe143d46f72f`.
The caffeOnAcl checkout was `a9d213eb6936d92a493a45a0f9640da723996b03`.
The caffeOnACL build was straightforwardly adapted from the user guide, however one failure was observed when running the unit tests - it looks like the TanH layer was not built properly, so we cannot use it during benchmarking. The BLAS backend was set to OpenBLAS, which will be the fallback when we are not using ACL layers. The changes to the configuration that have been done were:

CPU_ONLY := 0
USE_PROFILING := 1

For choosing whether we use ACL layers or not, we use the `BYPASSACL` environment variable.

We control whether the GPU is used by the `-gpu all` flag.[2]

Tests

The tests consist of running the BVLC reference ImageNet model, referred to as CaffeNet. We download the reference trained model and run its testing phase as defined in the Caffe documentations, which should run validation on the ILSVRC2012 data set. In each case we run 1000 iterations and then average over them.[5] We start in the root directory of caffeOnACL.

```
CAFFE=build/tools/caffe
REF=models/bvls_reference_caffenet
MODEL=$REF/deploy.prototxt
WEIGHTS=$REF/bvlc_reference_caffenet.caffemodel
ITERS=1000
```

CPU Caffe

This uses the default Caffe CPU backend, which in our case should rely on OpenBLAS for optimizations.

```
export BYPASSACL=0xffffffff
$CAFFE test -model $MODEL -weights $WEIGHTS \
            -iterations $ITERS
```

CPU ACL

This uses the ACL backend for CPU computation, which optimizes via ASIMD for ARMv8.[4]

```
export BYPASSACL=0x0
$CAFFE test -model $MODEL -weights $WEIGHTS \
            -iterations $ITERS
```

GPU ACL

This relies on the ACL library to communicate with OpenCL to run on Mali.[4]

```
export BYPASSACL=0x0
$CAFFE test -model $MODEL -weights $WEIGHTS \
            -gpu all -iterations $ITERS
```

Results

Legend

These names are taken from Caffe's documentation, and further details can be found there.[5]

- POOLING - A pooling layer.
- FC - Fully Connected, inner product layer.
- RELU - Rectified Linear Unit.
- LRN - Local Response Normalization.
- CONV - A convolution layer.
- SOFTMAX - A softmax layer.

Values

The value are in seconds and have been rounded to 7 decimal places.

Layer	CPU Caffe	CPU ACL	GPU ACL
POOLING	0.0438035	0.1741971	0.1683502
FC	0.1208655	0.0604789	0.0597605
RELU	0.0084044	0.0040321	0.0038901
LRN	0.1069563	0.01983366	0.0196404
CONV	0.7755730	0.7150786	0.7573765
SOFTMAX	0.0009729	0.0002014	0.0002009

Discussion

For all layers except convolution, it looks like there is a small advantage in using the GPU over the CPU ACL which runs ASIMD, however for some layers the benefit looks very minute. I expect the main issue with running on the GPU is that even though it accelerates the computation greatly due to its parallelisation, however there is an overhead present due to the memory mapping. Another possible issue, especially for convolution, could be the size of the kernel, since the GPU is best suited for running a lot of small kernels in parallel, however a large convolution kernel where there could be a high amount of memory accesses can slow things down. Note that this is just conjecture for now, and more investigation is needed.
The native Caffe CPU implementation seems to pull ahead in the pooling

layer quite significantly. When ACL is not used Caffe falls back on OpenBLAS usually, so either OpenBLAS or Caffe's usage of it seems to still be more optimized than ARM's when it comes to pooling. It is surprising, since from my investigations it looks like OpenBLAS does not use ASIMD for ARMv8, which we are using, so it would be interesting to compare this to a ARMv7 benchmark.

Bibliography

[1] https://community.arm.com/graphics/b/blog/posts/arm-compute-library-for-computer-vision-and-machine-learning-now-publicly-available

[2] https://github.com/OAID/caffeOnACL

[3] https://en.wikipedia.org/wiki/Samsung_Galaxy_S8

[4] https://github.com/ARM-software/ComputeLibrary

[5] http://caffe.berkeleyvision.org/