(Nearly everything you need to know about) optimising convolutional neural networks on embedded platforms with OpenCL

Anton Lokhmotov [dividiti]
Grigori Fursin [dividiti / cTuning foundation]

The 4th International Workshop on OpenCL (IWOCL)
21 April 2016, Vienna
Example: trends and challenges in auto industry

Trends
- Connected cars
- Natural user interfaces
- Autonomous and semi-autonomous cars

Challenges
- Sophisticated algorithms
- Heterogeneous hardware
- Software must be both reliable and efficient
Convolutional neural networks (CNNs)

- “Deep” (multi-layered) neural networks:
  - One or more convolutional layers
  - One or more fully connected layers
  - Normalisation, pooling, dropout...

- Take advantage of the 2D structure in images, hence useful for classification, localisation, detection.
CNN example: AlexNet (A. Krizhevsky et al., 2012)
CNN training and deployment

- Training is typically done on clusters with NVIDIA GPUs.
- Deployment is spreading to mobile & embedded platforms.
  - Can we deploy a CNN to achieve the required rate and accuracy of recognition on a given platform?
  - Can we identify or build such a platform under given constraints such as those on power, memory, price?
  - If all else fails, can we design another CNN by trading off performance, accuracy and cost?
Optimising CNNs with OpenCL
OpenCL support in Caffe

- Caffe (caffe.berkeleyvision.org) is a popular deep learning framework with a DSL for describing neural networks.
- Caffe’s master branch still only supports CUDA.
- AMD’s Caffe port uses OpenCL 1.2 and C++ templates.
- Caffe’s OpenCL branch is in active development led by Fabian Tschopp.
  - ViennaCL: required.
  - clBLAS: optional.
Preliminary results for AlexNet on Chromebook 2

- **Samsung Chromebook 2:**
  - Quad-core ARM Cortex-A15 CPU @ 1900 MHz
  - Quad-core ARM Mali-T628 GPU @ 533 MHz
  - 2 GB RAM
- **AlexNet w/ batch size of 128 using:**
  - CPU w/ OpenBLAS 0.2.17
  - GPU w/ ViennaCL 1.7.1: ~10x slower than OpenBLAS
  - GPU w/ clBLAS 2.4: ~4x slower than OpenBLAS
SGEMM - FP32 matrix-matrix multiplication

- Convolution is implemented as matrix-matrix multiplication.
- ~20,000 kernel enqueues, ~95% of which do SGEMM.
- Pros:
  - Single, regular routine to optimise.
- Cons:
  - Memory expansion (size + bandwidth implications).
  - Possibly awkward dimensions.
  - Is FP32 really necessary?
Crowdtuning ARM’s GEMM implementation

- ViennaCL performs FP32 GEMM @ <0.5 GFLOPS
- ARM’s implementation performs:
  - FP32 GEMM @ ~24 GFLOPS
  - FP16 GEMM @ ~45 GFLOPS
  - FP32/FP16 GEMM @ ~27 GFLOPS

cknowledge.org/repo/web.php?wcid=graph:crowdtune-sgemm-mali
Open call for collaborative optimisation
Collaborative optimisation of CNNs

Huge design and optimisation space
● Network design (state-of-the-art is ad-hoc).
● Network “compression” (50x storage reduction; 1% accuracy loss).
● Basic building blocks (GEMM, direct convolutions, FFT?).
● Data types (FP32, FP16, INT8?) + data layout transformations.

Continuous benchmarking and optimisation (see next slide)
● For speed, accuracy, size, energy consumption, etc.
● Across representative inputs, filter sizes, hardware platforms, etc.
Collective Knowledge: our humble solution

- Open framework + methodology (github.com/ctuning/ck).
- Combines reproducible experimentation with predictive analytics to extract “valuable insights” from “raw data”.
- Stimulates collaboration, thus reduces costs and risks.
- Dramatically accelerates knowledge discovery and optimization from many months to few days.

cknowledge.org; bit.ly/ck-date16; bit.ly/ck-multiprog16; arxiv.org/abs/1506.06256; dx.doi.org/10.3233/SPR-140396
Open call for collaborative benchmarking
The need for representative workloads

Benchmark [bench-mahrk]
- *noun* an abusive term for poorly constructed software, e.g. “this piece of software is a benchmark”
- *verb* to create a meaningless set of measurements, e.g. “we benchmarked the latest device”

Workload [wurk-lohd]
- *noun* a self-contained series of machine-executable actions formed from production code that presents a use case of interest for performance analysis

-- “Benchmarks vs Zombie Apocalypse: a Comparison”
(ADAPT’16 keynote by Ed Plowman, ARM)
The need for collaborative design and optimization

- Ever increasing complexity (many things may go wrong).
- Large, diverse engineering groups (e.g. hardware designers, system programmers, performance analysts).
- Ineffective collaboration wastes precious resources and increases business risks.
- Users run tomorrow’s workloads on yesterday’s hardware.
- Too easy to ignore emerging workloads, as they simply do not have the same status as benchmarks.
Community-sourced workloads

- Incentives for academia: demonstrable impact.
- Incentives for industry:
  - Software developers: similar to open-source software.
  - Hardware vendors:
    - Focussed effort on better design and optimisation.
    - Reduced effort on benchmarks.
    - Fair competition.
Our long term mission is to enable efficient and reliable computing everywhere.
WE'RE ON A MISSION FROM GOD

"We're getting the band back together."
Thank you!

anton@dividiti.com
CK modules (wrappers) with JSON API to abstract access to changing SW and HW

Unified input (JSON)

Unified command line interface

Any tool (compiler, lib, profiler, script …)

Processing (Python)

Set environment (tool versions, system state, …)

Parse and unify output

Unified output (JSON)

Detected features

Detected choices

Monitored run-time state

Monitored behavior

Generated files

CK entries with Unique IDs

Assemble experimental workflows from CK modules as LEGO for agile prototyping, crowdsourcing and analysis

Choose exploration strategy

Generate choices (code sample, data set, compiler, flags, architecture …)

Compile source code

Run code

Analyze variation

Apply Pareto filter

Stat. analysis and predictive analytics

Apply complexity reduction

Gradually convert to Collective Knowledge /

Ad-hoc tuning scripts

Algorithm, Program

Source to source transformations, Compilation

Data set

Hardware

State

Collection of CSV, XLS, TXT and other files

Original ad-hoc input

JSON converted into CK vectors

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd

Gradually convert to Collective Knowledge /

Typical experimental workflow

$ ck pull repo:ctuning-programs
$ ck list program
$ ck list dataset
$ ck compile program:*susan -speed
$ ck run program:automotive-susan
$ ck crowdtune program:automotive-susan

$ b = B( c, f, s )

Interdisciplinary crowd
Acknowledgements

HiPEAC
CARP (FP7)
MILEPOST (FP6)
TETRACOM (FP7)