Performance Analysis of CUDA Deep Learning Networks using TAU

Allen D. Malony, Robert Lim, Sameer Shende, Boyana Norris

Department of Computer and Information Science
Oregon Advanced Computing Institute for Science and Society

University of Oregon
Motivation

- Tremendous growth in the application of deep neural network (DNN) to deep learning (DL) applications
  - Image recognition:
    - Facebook photo tagging
  - Text recognition:
    - Google Translate
  - Speech recognition:
    - Baidu Deep Voice, ...

- DNN libraries and DL frameworks for programming
  - CUDA Deep Neural Network (cuDNN) library
  - Frameworks built around cuDNN for DL application creation
    - Torch, TensorFlow, Caffe/Caffe2, Neon, MXnet, theano, DL4j, …
Motivation (2)

- DNN/DL applications are complex (function, execution)
- Build DNN/DL development and evaluation ecosystem
- Provide support for performance measurement/analysis
  - Address concerns for users wanting to tune performance
  - Optimize sub-par configurations or target execution bugs
  - Understand performance relative to user’s high-level problem
    - execution time spent per layer, accuracy of trained results, …
- Target rich platforms for integration (libraries, frameworks)
- Integrate tools in CUDA development environment
  - Lack ability to report intricacies of cuDNN activity
  - Report behavior of DNN features at routine level
  - Confirm and optimize model/network configuration performance
Outline of Talk

- DNN libraries and DL frameworks
- Challenges in DNN/DL performance and analysis
  - Performance problems of concern
  - Application of CUDA performance tools to DNN/DL
- TAU Performance System
  - Overview
  - Support for CUDA performance analysis
- TAU prototype for DNN/DL performance analysis
  - Test know DNN benchmarks with different DL frameworks
  - Run on a variety of GPUs
  - See what DNN characteristics can be revealed
- Uses brain-inspired computing to process fine-grained tasks
- Decomposes problem into domain specific tasks
  - Tasks solve specific problem
  - Results collectively provide high level answers
Convolutional Neural Networks (CNN) Basics

- Convolution layer
  - Feature detector that learns to filter out unneeded information
    - apply a convolutional kernel to the input

- Pooling layers
  - Compute metrics on particular feature over region of input data
  - Also detects objects in unusual places, reduces memory size
Deep Neural Network (DNN) Workflow

- DNN frameworks implement a workflow made of a sequence of standard, common stages
  - Programming can be imperative or declarative

1) Target a backend (CPU or GPU, or both)

2) Load data

3) Specify model architecture
   - Create model by providing list of layers
   - Layers with weights:
     - provide function to initialize weights prior to training
     - layers (linear, convolution, pooling)
     - activations (RELU, softmax tanh)
     - initializers (constant, uniform, gaussian)
Deep Neural Network (DNN) Workflow (2)

4) Train model
   ○ Provide training data (as an iterator), cost function and optimization algorithm for updating model’s weights
   ○ Learning schedule:
     ♦ modify learning rate over training time
     ♦ datasets, costs and metrics
     ♦ optimizers (SGD, adagrad, adam)
     ♦ learning schedules

5) Evaluate
   ○ Evaluate trained model on validation dataset and metrics
   ○ Models, costs (cross entropy, SSE), metrics (LogLoss, PrecisionRecall)
NVIDIA CUDA DNN Library (cuDNN)

- GPU-accelerated library for DNN
- Provides highly-tuned implementations
  - Standard routines: forward/backward convolution, pooling, normalization, activation layers
  - Part of NVIDIA deep learning SDK
- Deep learning researchers and framework developers worldwide rely on cuDNN for acceleration
  - Focus on training neural networks and developing software applications rather than tuning low-level GPU performance
- cuDNN accelerates widely-used DL frameworks
Key Features of cuDNN

- Forward/backward paths for many common layer types
  - Pooling, LRN, LCN, and batch normalization
  - ReLU, Sigmoid, softmax, and Tanh
- Forward and backward convolution routines
  - Cross-correlation
  - Designed for convolutional neural networks (CNN)
- Recurrent Neural Networks (RNN) and Persistent RNN
  - LSTM (long short-term memory)
  - GRU (gated recurrent unit)
- 4d tensors
  - Arbitrary dimension ordering, striding, sub-regions
- Tensor transformation functions
- Context-based API for easy multithreading
Deep Learning (DL) Frameworks

- DL frameworks effectively implement DL workflows
- Large variety of DL frameworks

Torch

- Torch
  - Popular scientific framework
    - LuaJIT and Python (PyTorch) flavors
    - Has its roots with Facebook AI Research (circa 2000)
  - Best performer of DNN libraries on convnet benchmarks

- Neural network (nn) package
  - Module: abstract class inherited by Module
  - Containers: Sequential, Parallel, Concat
  - Transfer functions: Tanh, Sigmoid
  - Simple layer: Linear, Mean, Max, Reshape
  - Table layers: SplitTable, ConcatTable, JoinTable
  - Convolution layers: Temporal, Spatial, Volumetric
  - Criterion
TensorFlow

- TensorFlow
  - Deep learning library from Google
  - Open source (acquired under DeepMind project)
  - Primitives for defining functions on tensors and automatically computing derivatives

- Numerical computation using data flow graphs
  - Nodes represent mathematical operations
  - Graph edges represent the multidimensional data arrays (tensors) communicated between them
  - Flexible architecture supports CPU and GPU execution
  - Executes deep learning routines on GPU (cuDNN)
TensorFlow Computation Graph

- Placeholders - input nodes in TF computational graph
- Feed dictionaries - how users set values for placeholder (or other) variables when running a computation

**TF graph for linear regression**

```
x = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))

W = tf.get_variable("weights", (1, 1),
                   initializer=tf.random_normal_initializer())

b = tf.get_variable("bias", (1,),
                   initializer=tf.constant_initializer(0.0))

y_pred = tf.matmul(x, W) + b

J(W, b) = \frac{1}{N} \sum_{i=1}^{N} (y_i - (Wx_i + b))^2

loss = tf.reduce_sum((y - y_pred)**2 / n_samples)

opt_operation = tf.train.AdamOptimizer().minimize(loss)
```
Neon

- Neon is an open source Python-based language and a set of libraries for developing deep learning models
  - Acquired by Intel
  - Claims to be very fast
- Python-like syntax
- Object-oriented implementations of all the deep learning components, including layers, learning rules, activations, optimizers, initializers, and costs functions
- All common deep learning models, including convnets, MLPs, RNNs, LSTMs and autoencoders
- Create novel algorithms using linear algebra, auto-differentiation, and other advanced capabilities with a numpy-like syntax
Profiling DNN / DL Applications

- Performance of DNN/DL applications is important
  - Considerable time is spent in training and inference
  - How can these be sped up?
  - Understand performance inefficiencies
  - Understand opportunities for improvement
- To date, metrics reported for DNN applications include:
  - Classification accuracy
  - Time spent per layer (forward, backward)
- Currently lacking the ability to report operations executed, memory footprint, system aspects, ...
- Performance measurements of interest for developers:
  - Input resolution, ops per layer or module, memory access patterns, # hidden neurons, ...
  - Whether DNN training is making significant progress
Heterogeneous HPC and Performance Tools

- Heterogeneous systems drive HPC performance
  - Multi-CPU, multicore shared memory nodes
  - Manycore accelerators with high-BW I/O

- Heterogeneous software development technology important to deliver on performance potential
  - More sophisticated parallel programming environments
  - Integrated development and performance tools
    - support heterogeneous performance model and perspectives
  - HPC tools research on heterogeneous GPU computing
Heterogeneous + GPU Performance Analysis

- Heterogeneous HPC concerns node-level and cluster-wide performance issues
  - Maintain high concurrency and keep accelerators busy
  - Balancing of load across nodes
  - Reducing overheads and increasing efficiency

- Need to achieve high GPU performance as well
  - Multiple parameters are involved
    - threads, blocks, registers, shared memory, …
  - Data parallel programming concerns
    - work size, locality, branch divergence, ...
  - Need better support for analysis and optimization
TAU Performance System® (http://tau.uoregon.edu)

- Performance problem solving framework
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (open source)
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Used in HPC software, systems, applications
Integrated Heterogeneous Support in TAU

- State-of-the-art comprehensive HPC performance analysis
  - Multicore, node-level, communication, manycore accelerator

- GPU performance analysis
  - GPU measurement (CUPTI enabled)
    - CUDA library wrapping/callback
    - timing, counters, sampling, transfer
  - Languages
    - CUDA, OpenCL, OpenACC
    - OpenMP 4.x with offloading
  - Integrated profiling and tracing
  - Ports to Linux x86_64, CrayCNL, ARM64, and Power 8 Linux

- Static analysis of GPU kernel
  - Instruction mix, control flow, memory, occupancy, time, ...

- Autotuning with static+dynamic analysis and modeling
Configuring TAU with DNN/DL Libraries

- TAU currently supports the following libraries
  - TensorFlow, Nervana Neon, Torch (PyTorch), Theano
  - Near alpha: cuDNN, Torch (Lua)
  - Beta: DL4J

- Example config command:
  ```
  ./configure -boost=/home/roblim1/apps/boost_1_61_0 -pythoninc=/home/roblim1/anaconda2/include/python2.7 -pythonlib=/home/roblim1/anaconda2/lib -cc=gcc -cuda=/cm/extra/apps/cuda80/toolkit/8.0.27 -pdt=/home/roblim1/repos/pdtoolkit-3.20 -bfd=download && make install
  ```

- TAU makefiles allow multiple installations to coexist based on config settings (MPI, OpenMP, CUDA, etc.)
  ```
  export TAU_MAKEFILE=’/home/roblim1/repos/tau2/x86_64/lib/Makefile.tau-papi-python-cupti-pdt’
  ```
PyTau and tau_python

- TAU provides several ways of measuring performance for Python-based applications
  - tau_python source code wrapping
  - pytau instrumentation

- Python source code wrapping
  - Parses the Python source code
  - Wraps each routine with TAU Python instrumentation

- tau_python example:
  ```
tau_python -T cupti,serial -cupti mnist_mlp.py
  ```
PyTau and tau_python (2)

- TAU Python measurement API (**pytau**)
  - Calls the underlying TAU performance measurement

- **pytau** example:
  ```python
  from neon.models import Model
  from neon.optimizers import GradientDescentMomentum
  from neon.transforms import Rectlin, Logistic, CrossEntropyBinary, Misclassification
  import pytau
  ...
  x = pytau.profileTimer("MLP Fit")
  pytau.start(x)
  mlp.fit(train_set, optimizer=optimizer, num_epochs=args.epochs, cost=cost,
           callbacks=callbacks)
  error_rate = mlp.eval(valid_set, metric=Misclassification())
  pytau.stop(x)
  pytau.dumpDb()
  ...
  ```
GPU Hardware Counters

- Configure TAU to access hardware counters
  - PAPI for CPU and CUPTI for GPU
- `tau_cupti_avail`: lists available GPU counters

```
CUDA.Tesla_P100-PCIE-16GB.domain_d.active_warps        active_warps
CUDA.Tesla_P100-PCIE-16GB.domain_d.atom_count          atom count
CUDA.Tesla_P100-PCIE-16GB.domain_d.branch              branch
CUDA.Tesla_P100-PCIE-16GB.domain_d.divergent_branch    divergent branch
```

- Calculate derived metrics, such as:
  - Instructions per cycle, global store per instruction
- Enable counters “instructions executed” + “active cycles”

```
export TAU_METRICS='CUDA.Tesla_P100-PCIE-16GB.domain_d.inst_executed:
CUDA.Tesla_P100-PCIE-16GB.domain_d.active_cycles'
```
Profile Example with Neon and Alexnet

ParaProf

pprof
Profile Example with Neon and Alexnet (2)
Hardware used in Experiments

- Intel Ivy Bridge and Haswell processors
- Three GPUs from 3 NVIDIA architectures

<table>
<thead>
<tr>
<th>NVIDIA GPU</th>
<th>K80</th>
<th>M40</th>
<th>P100</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA capability</td>
<td>3.5</td>
<td>5.2</td>
<td>6.2</td>
</tr>
<tr>
<td>Global memory (MB)</td>
<td>11520</td>
<td>12288</td>
<td>16276</td>
</tr>
<tr>
<td>Multiprocessors (MP)</td>
<td>13</td>
<td>24</td>
<td>56</td>
</tr>
<tr>
<td>CUDA cores per MP</td>
<td>192</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>L2 cache (MB)</td>
<td>1.572</td>
<td>3.146</td>
<td>4.193</td>
</tr>
<tr>
<td>Architecture family</td>
<td>Kepler</td>
<td>Maxwell</td>
<td>Pascal</td>
</tr>
</tbody>
</table>

- All DNN libraries built with:
  - CUDA v8.0.44
  - cuDNN 5.1
Convnet Benchmarks

- Convnet tests convolutional neural network layer
- Four notable implementations include
  - AlexNet (http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf)
  - GoogLeNet (http://deeplearning.net/tag/googlenet/)
  - OverFeat (http://cilvr.nyu.edu/doku.php?id=code:start)
  - Vgg (http://www.robots.ox.ac.uk/~vgg/research/very_deep/)

- Examine similar implementations of above benchmarks in Neon and Torch, comparing three GPU architectures
## Code Features

- Neon and Torch code features
  - Lines = lines of code
  - TAU routines = # routines intercepted by TAU (thread 1)
    - depends on which libraries get called

<table>
<thead>
<tr>
<th>Application</th>
<th>Library</th>
<th># lines</th>
<th>TAU routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>neon</td>
<td>70</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>torch</td>
<td>100</td>
<td>66</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>neon</td>
<td>92</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>torch</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Overfeat</td>
<td>neon</td>
<td>63</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>torch</td>
<td>100</td>
<td>69</td>
</tr>
<tr>
<td>Vgg</td>
<td>neon</td>
<td>67</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>torch</td>
<td>100</td>
<td>98</td>
</tr>
</tbody>
</table>
# Total Execution Time

<table>
<thead>
<tr>
<th>App</th>
<th>Library</th>
<th>K80</th>
<th>M40</th>
<th>P100</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>Neon</td>
<td>16911868.50</td>
<td>18358841.25</td>
<td>4305760.75</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td></td>
<td>7279472.25</td>
<td>13774497.00</td>
<td>5520680.00</td>
</tr>
<tr>
<td>OverFeat</td>
<td></td>
<td>7864390.75</td>
<td>9176604.75</td>
<td>8059057.00</td>
</tr>
<tr>
<td>VGG</td>
<td></td>
<td>7183726.25</td>
<td>12744961.00</td>
<td>9443090.00</td>
</tr>
<tr>
<td>AlexNet</td>
<td>Torch</td>
<td>8029959.75</td>
<td>5078672.75</td>
<td>4427608.25</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td></td>
<td>54639307.50</td>
<td>13755790.25</td>
<td>15526875.75</td>
</tr>
<tr>
<td>OverFeat</td>
<td></td>
<td>11296946.75</td>
<td>5229369.25</td>
<td>4976697.50</td>
</tr>
<tr>
<td>VGG</td>
<td></td>
<td>14956690.75</td>
<td>5059248.25</td>
<td>5177141.25</td>
</tr>
</tbody>
</table>
# Instruction Throughput per # Cycles (Neon)

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TAU</th>
<th>Inclusive TAU</th>
<th>Exclusive CUDA</th>
<th>Inclusive CUDA</th>
<th>Exclusive CUDA</th>
<th>Inclusive CUDA</th>
<th>Calls</th>
<th>Childs</th>
</tr>
</thead>
<tbody>
<tr>
<td>scarv_direct_update_64k2</td>
<td>0.132</td>
<td>0.132</td>
<td>24.082,456,533</td>
<td>24.082,456,533</td>
<td>5.523,256,387</td>
<td>5.523,256,387</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>scarv_direct_fprop_64k28</td>
<td>0.116</td>
<td>0.116</td>
<td>20.785,641,687</td>
<td>20.785,641,687</td>
<td>4.856,819,517</td>
<td>4.856,819,517</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>prepare</td>
<td>0.002</td>
<td>0.002</td>
<td>13,276,385</td>
<td>13,276,385</td>
<td>38,478,541.5</td>
<td>38,478,541.5</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>scarv_winograd_3x3_4x4_32x32_D</td>
<td>0.055</td>
<td>0.055</td>
<td>7.719,400,539</td>
<td>7.719,400,539</td>
<td>2,162,991,393</td>
<td>2,162,991,393</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>scarv_winograd_4x4_3x3_32x32_XQ18_N128</td>
<td>0.193</td>
<td>0.193</td>
<td>6.274,202,011</td>
<td>6.274,202,011</td>
<td>1,888,606,405</td>
<td>1,888,606,405</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>scarv_winograd_3x3_4x4_32x32</td>
<td>0.021</td>
<td>0.021</td>
<td>5,617,510.4</td>
<td>5,617,510.4</td>
<td>1,754,440.2</td>
<td>1,754,440.2</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>spool_bprop_max_overlap</td>
<td>0.052</td>
<td>0.052</td>
<td>3,546,408,356</td>
<td>3,546,408,356</td>
<td>915,899.5</td>
<td>915,899.5</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>sgemm_r1_128k128_vec</td>
<td>0.027</td>
<td>0.027</td>
<td>2,916,601,111</td>
<td>2,916,601,111</td>
<td>695,415,756</td>
<td>695,415,756</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>sgemm_r1_128k128_vec</td>
<td>0.027</td>
<td>0.027</td>
<td>2,916,601,111</td>
<td>2,916,601,111</td>
<td>695,415,756</td>
<td>695,415,756</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>spool_bprop_max</td>
<td>0.042</td>
<td>0.042</td>
<td>1,148,160,533</td>
<td>1,148,160,533</td>
<td>325,994,211</td>
<td>325,994,211</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>copy_transpose</td>
<td>0.008</td>
<td>0.008</td>
<td>983,450,533</td>
<td>983,450,533</td>
<td>983,450,533</td>
<td>983,450,533</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_div_mul_add_mul_add</td>
<td>0.032</td>
<td>0.032</td>
<td>501,396,367</td>
<td>501,396,367</td>
<td>321,351,799</td>
<td>321,351,799</td>
<td>6.5</td>
<td>0</td>
</tr>
<tr>
<td>update_image_3x3_4x4</td>
<td>0.026</td>
<td>0.026</td>
<td>460,891,156</td>
<td>460,891,156</td>
<td>343,536,789</td>
<td>343,536,789</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>vprop_image_4x4</td>
<td>0.032</td>
<td>0.032</td>
<td>414,720,233</td>
<td>414,720,233</td>
<td>414,346,411</td>
<td>414,346,411</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>update_delta_3x3</td>
<td>0.017</td>
<td>0.017</td>
<td>313,491,467</td>
<td>313,491,467</td>
<td>457,963,389</td>
<td>457,963,389</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_get_mem_add_mul</td>
<td>0.034</td>
<td>0.034</td>
<td>174,116,888</td>
<td>174,116,888</td>
<td>270,096,414</td>
<td>270,096,414</td>
<td>52.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_max_minimum_mul_add_add</td>
<td>0.022</td>
<td>0.022</td>
<td>173,786,881</td>
<td>173,786,881</td>
<td>251,537,914</td>
<td>251,537,914</td>
<td>52.5</td>
<td>0</td>
</tr>
<tr>
<td>bprop_filter_4x4</td>
<td>0.002</td>
<td>0.002</td>
<td>25.490</td>
<td>25.490</td>
<td>55,856,056</td>
<td>55,856,056</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>fprop_filter_4x4</td>
<td>0.002</td>
<td>0.002</td>
<td>18,390,235</td>
<td>18,390,235</td>
<td>52,126,311</td>
<td>52,126,311</td>
<td>22.5</td>
<td>0</td>
</tr>
<tr>
<td>filter_dimshuffle</td>
<td>0</td>
<td>0</td>
<td>13,220,833</td>
<td>13,220,833</td>
<td>8,705,667</td>
<td>8,705,667</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_max_sub_16384_sub_16384</td>
<td>0</td>
<td>0</td>
<td>10,366.2</td>
<td>10,366.2</td>
<td>27,102.2</td>
<td>27,102.2</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_neg_mul_table_log_mul_sum</td>
<td>0</td>
<td>0</td>
<td>8,558.2</td>
<td>8,558.2</td>
<td>11,792,933</td>
<td>11,792,933</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_sub_mul</td>
<td>0</td>
<td>0</td>
<td>3,390,233</td>
<td>3,390,233</td>
<td>2,542.2</td>
<td>2,542.2</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_onehot</td>
<td>0</td>
<td>0</td>
<td>2,733,587</td>
<td>2,733,587</td>
<td>2,524.4</td>
<td>2,524.4</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_mul</td>
<td>0</td>
<td>0</td>
<td>2,333,587</td>
<td>2,333,587</td>
<td>2,100,133</td>
<td>2,100,133</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>scarv_direct_bprop_64x4</td>
<td>0.005</td>
<td>0.005</td>
<td>2,080,333</td>
<td>2,080,333</td>
<td>2,985.9</td>
<td>2,985.9</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>write_size</td>
<td>0</td>
<td>0</td>
<td>11.5</td>
<td>11.5</td>
<td>1,273</td>
<td>1,273</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>kernel_sum_mul</td>
<td>0</td>
<td>0</td>
<td>3,433</td>
<td>3,433</td>
<td>75.6</td>
<td>75.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TAU application</td>
<td>11.027</td>
<td>2.594</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Memory copy Device to Host</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Memory copy Host to Device</td>
<td>0.091</td>
<td>0.091</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cudaCreateV2</td>
<td>0.446</td>
<td>0.446</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>cudaGetVersion</td>
<td>0.045</td>
<td>0.045</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>3.5</td>
</tr>
<tr>
<td>cudaGetCurrent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
</tr>
<tr>
<td>cudaGetDevice</td>
<td>0.002</td>
<td>0.002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>cudaGetLimit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>cudaGetCurrentV2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>cudaPushCurrentV2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>cudaGetLimit</td>
<td>0.002</td>
<td>0.002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Results – Neon and Torch on Convnet

- Neon and Torch on Convnet benchmarks.

# warps executed per issued instructions
- Lower is better (ideally should be near zero)
- P100 does well for AlexNet and GoogleNet
- M40 does (very) well for OverFeat and VGG
Results – Neon and Torch on Convnet (2)

- # issued instructions per global memory operation
  - Higher is better (eliminate GM reads as much as possible)
  - K80 is clearly the outlier (bottom figure removes K80)
  - P100 does well for Neon and Torch (16 GB versus 12 GB)
Results – Neon and Torch on Convnet (3)

- # warps issued per global memory instructions
  - More is better (same reason in minimizing GM reads)
  - K80 is clearly the outlier (bottom figure removes K80)
  - P100 does well for OverFeat and VGG (128 warps/MP)
  - M40 does better in AlexNet and GoogLeNet (64 warps/MP)
Results – Neon and Torch on Convnet (4)

- Calculate metrics for individual routines
- Allows comparison of performance factors

### Neon

<table>
<thead>
<tr>
<th>app</th>
<th>metric</th>
<th>kernel</th>
<th>k80</th>
<th>m40</th>
<th>p100</th>
</tr>
</thead>
<tbody>
<tr>
<td>alexnet</td>
<td>warps/ins</td>
<td>kernel_1h</td>
<td>12.89</td>
<td>3.73</td>
<td>7.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sconv_fp</td>
<td>12.89</td>
<td>3.73</td>
<td>7.32</td>
</tr>
<tr>
<td></td>
<td>gld/ins</td>
<td>kernel_1h</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sconv_fp</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>googlenet</td>
<td>warps/ins</td>
<td>kernel_1h</td>
<td>16.25</td>
<td>2.34</td>
<td>23.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sconv_fp</td>
<td>16.25</td>
<td>2.34</td>
<td>23.77</td>
</tr>
<tr>
<td></td>
<td>gld/ins</td>
<td>kernel_1h</td>
<td>0</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sconv_fp</td>
<td>0</td>
<td>0.00</td>
<td>0.02</td>
</tr>
</tbody>
</table>

### Torch

<table>
<thead>
<tr>
<th>app</th>
<th>metric</th>
<th>kernel</th>
<th>k80</th>
<th>m40</th>
<th>p100</th>
</tr>
</thead>
<tbody>
<tr>
<td>alexnet</td>
<td>warps/ins</td>
<td>add_tensor</td>
<td>8.36</td>
<td>4.76</td>
<td>8.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cudnn.sgemm</td>
<td>8.34</td>
<td>4.25</td>
<td>8.06</td>
</tr>
<tr>
<td></td>
<td>gld/ins</td>
<td>add_tensor</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cudnn.sgemm</td>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>googlenet</td>
<td>warps/ins</td>
<td>max_scudnn</td>
<td>30.20</td>
<td>1.42</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cudnn_dgrad</td>
<td>7.40</td>
<td>9.43</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>gld/ins</td>
<td>max_scudnn</td>
<td>0</td>
<td>0.00</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cudnn_dgrad</td>
<td>0.0</td>
<td>0.0</td>
<td>0.80</td>
</tr>
</tbody>
</table>
VGG Tuning Candidates (Hot off the press!)

Table: Candidates for performance tuning in VGG application, comparing Neon and Torch implementations.

- VGG Neon had a lot of global memory reads (previous two slides)
- Identify which routines to focus performance tuning efforts
  - `sconv_winograd` shows up a few times for both architectures!
Conclusion

- Demonstrated profiling capabilities of DNN applications using TAU Performance System
  - Real story is more complicated than this
- DL frameworks can do multi-node parallel processing
  - Do not look like your standard HPC application
  - TAU is able to support performance measurement
    - needs to support DL distributed execution models
    - Develop DL-specific performance analysis techniques
- Interested in HPC-class applications with deep learning
  - Exascale Deep Learning and Simulation Enabled Precision Medicine for Cancer (CANDLE)
  - Livermore Big Artificial Neural Network HPC Toolkit
Acknowledgements

- J-C Vasnier of NVIDIA
  - Provide access to PSG clusters

- American Society for Engineering Education

- Franco-Américaine Fulbright Commission