**Benanza: Automatic μBenchmark Generation to Compute “Lower-bound” Latency and Inform Optimizations of Deep Learning Models on GPUs**

Cheng Li*¹, Abdul Dakkak*¹, Jinjun Xiong², Wen-mei Hwu¹

University of Illinois Urbana-Champaign¹, IBM Research²

{cli99, dakkak, w-hwu}@illinois.edu, jinjun@us.ibm.com
Motivation

- The benchmarking → optimization process for Deep Learning (DL) workloads is ad-hoc and slow
- There is a need for a DL optimization advising design that can systematically guide researchers to potential optimization opportunities and assess hypothetical execution scenarios
Answers to the following are highly desired

- What is the potential latency speedup if optimizations are performed?
- Are independent layers executed in parallel?
- Are the optimal algorithms used for convolution layers?
- Is there any inefficiency or unexpected behavior in frameworks?
- Does the execution fuse layers or leverage Tensor Cores? And what are the benefits?
Observations

- Layers are the performance building blocks
- Frameworks use cuDNN & cuBLAS to execute layers on GPUs
  - Given a specific model/HW/SW, the cuDNN & cuBLAS functions invoked are fixed

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>cuDNN / cuBLAS API</th>
<th>Tensor Core Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>cudnnConvolutionForward</td>
<td>✓</td>
</tr>
<tr>
<td>Activation</td>
<td>cudnnActivationForward</td>
<td>×</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>cudnnBatchNormalizationForwardInference</td>
<td>×</td>
</tr>
<tr>
<td>Conv+Bias+Activation</td>
<td>cudnnConvolutionBiasActivationForward</td>
<td>✓</td>
</tr>
<tr>
<td>RNN</td>
<td>cudnnRNNForwardInference</td>
<td>✓</td>
</tr>
<tr>
<td>Dropout</td>
<td>cudnnDropoutForward</td>
<td>×</td>
</tr>
<tr>
<td>Pooling</td>
<td>cudnnPoolingForward</td>
<td>×</td>
</tr>
<tr>
<td>Softmax</td>
<td>cudnnSoftmaxForward</td>
<td>×</td>
</tr>
<tr>
<td>Add</td>
<td>cudnnAddTensor</td>
<td>×</td>
</tr>
<tr>
<td>Element-wise</td>
<td>cudnnOpTensor</td>
<td>×</td>
</tr>
<tr>
<td>Rescale</td>
<td>cudnnScaleTensor</td>
<td>×</td>
</tr>
<tr>
<td>GEMM</td>
<td>cublas*Gemm / cublasGemmEx</td>
<td>✓</td>
</tr>
<tr>
<td>GEMV</td>
<td>cublasSgemv</td>
<td>×</td>
</tr>
</tbody>
</table>
cuDNN and cuBLAS Dominate the Compute

The percentage of layers that are supported in cuDNN and cuBLAS

GPU kernel time breakdown for all 30 models on Volta GPU
Our Approach

- Knowing the ideal helps understand how to improve the latency
- We introduce a new metric, “lower-bound” latency (LBL)
  - Defined by the latencies of the cuDNN & cuBLAS functions corresponding to the model layers
  - Estimates the ideal latency of a model given a specific GPU HW/SW
- (measured_latency - LBL) indicates optimization opportunities
“Lower-bound” Latency (LBL)

- Computed under different scenarios.

- Data-independent layers might be executed sequentially or in parallel
  - $\text{LBL}_{\text{sequential}} = \text{sum of all layer latencies}$
  - $\text{LBL}_{\text{parallel}} = \text{sum of layer latencies on the critical path}$

- $\text{LBL}_{\text{sequential}} > \text{LBL}_{\text{parallel}}$ for models with parallel modules, otherwise equal
Benanza

- A benchmarking and analysis design that speeds up the benchmarking/optimization cycle of DL models on GPUs
- Consists of 4 modular components:
  - Model Processor
  - Benchmark Generator
  - Performance Database
  - Analyzer
Design and Workflows

Benanza Design and Workflows
The Benchmarking Workflow

Benanza Design and Workflows
The Benchmarking Workflow

Benanza Design and Workflows
Benchmark Generator

- Generates C++ code using the layer information to measure the cuDNN or cuBLAS API corresponding to the layer
- Algorithm Instantiation
  - 8 different algorithms for the cuDNN convolution API
  - Frameworks rely on a heuristic function to select algorithm
  - Generates benchmarks for all available algorithms

```cpp
algorithm = cudnnGetConvolutionForwardAlgorithm(layer)
cudnnConvolutionForward(layer, algorithm)
```
Benchmark Generator

- **Layer Fusion Support**
  - Generates benchmarks that target the cuDNN fused API
  - E.g. `convolution->bias->activation, convolution->bias`

- **Data Type Support**
  - Generates benchmarks that target different data types
  - E.g. `float16 for Tensor Cores`
The Benchmarking Workflow

Indexed by the system, data type, and layer information

Benanza Design and Workflows
The Analysis Workflow

The user runs the target model using the SW/HW of interest to get the model execution profile.
The Analysis Workflow

The user inputs the model execution profile along with the model, system, data type.

The Analyzer queries the DB to inform optimizations.

Benanza Design and Workflows
Evaluation Setup

- MXNet, ONNX Runtime and PyTorch
- 7 GPU systems from Kepler to the latest Turing
- CUDA 10.1 and cuDNN7.6.3
- Unless specified, batch size = 1

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Task</th>
<th>MACs</th>
<th># Layers</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arcface [16]</td>
<td>FR</td>
<td>12.08G</td>
<td>412</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>BVLC-Alexnet [17]</td>
<td>IC</td>
<td>656M</td>
<td>24</td>
<td>2012</td>
</tr>
<tr>
<td>4</td>
<td>BVLC-Googlenet [18]</td>
<td>IC</td>
<td>1.59G</td>
<td>143</td>
<td>2014</td>
</tr>
<tr>
<td>5</td>
<td>BVLC-RCNN-ILSVRC13 [19]</td>
<td>IC</td>
<td>718M</td>
<td>23</td>
<td>2013</td>
</tr>
<tr>
<td>6</td>
<td>Densenet-121 [20]</td>
<td>IC</td>
<td>2.87G</td>
<td>910</td>
<td>2016</td>
</tr>
<tr>
<td>7</td>
<td>DUC [21]</td>
<td>SS</td>
<td>34.94G</td>
<td>355</td>
<td>2017</td>
</tr>
<tr>
<td>8</td>
<td>Emotion Ferplus [22]</td>
<td>ER</td>
<td>877M</td>
<td>52</td>
<td>2016</td>
</tr>
<tr>
<td>9</td>
<td>Inception-v1 [23]</td>
<td>IC</td>
<td>1.44G</td>
<td>144</td>
<td>2015</td>
</tr>
<tr>
<td>10</td>
<td>Inception-v2 [24]</td>
<td>IC</td>
<td>2.03G</td>
<td>509</td>
<td>2015</td>
</tr>
<tr>
<td>11</td>
<td>LeNet [25]</td>
<td>HR</td>
<td>796K</td>
<td>12</td>
<td>2010</td>
</tr>
<tr>
<td>12</td>
<td>MobileNet-v2 [26]</td>
<td>IC</td>
<td>437M</td>
<td>155</td>
<td>2017</td>
</tr>
<tr>
<td>13</td>
<td>ResNet18-v1 [27]</td>
<td>IC</td>
<td>1.82G</td>
<td>69</td>
<td>2015</td>
</tr>
<tr>
<td>14</td>
<td>ResNet18-v2 [28]</td>
<td>IC</td>
<td>1.82G</td>
<td>69</td>
<td>2019</td>
</tr>
<tr>
<td>15</td>
<td>ResNet34-v1 [27]</td>
<td>IC</td>
<td>3.67G</td>
<td>125</td>
<td>2015</td>
</tr>
<tr>
<td>17</td>
<td>ResNet50-v1 [27]</td>
<td>IC</td>
<td>3.87G</td>
<td>175</td>
<td>2015</td>
</tr>
<tr>
<td>19</td>
<td>ResNet101-v1 [27]</td>
<td>IC</td>
<td>7.58G</td>
<td>345</td>
<td>2015</td>
</tr>
<tr>
<td>21</td>
<td>ResNet152-v1 [27]</td>
<td>IC</td>
<td>11.30G</td>
<td>515</td>
<td>2015</td>
</tr>
<tr>
<td>23</td>
<td>Shufflenet [29]</td>
<td>IC</td>
<td>127M</td>
<td>203</td>
<td>2015</td>
</tr>
<tr>
<td>25</td>
<td>Tiny Yolo-v2 [31]</td>
<td>OD</td>
<td>3.13G</td>
<td>32</td>
<td>2016</td>
</tr>
<tr>
<td>26</td>
<td>Vgg16-BN [32]</td>
<td>IC</td>
<td>15.38G</td>
<td>54</td>
<td>2014</td>
</tr>
<tr>
<td>27</td>
<td>Vgg16 [32]</td>
<td>IC</td>
<td>15.38G</td>
<td>41</td>
<td>2014</td>
</tr>
<tr>
<td>30</td>
<td>Zfnet512 [33]</td>
<td>IC</td>
<td>1.48G</td>
<td>22</td>
<td>2013</td>
</tr>
</tbody>
</table>

7 GPU systems

30 ONNX models
Q1&2. LBL and Parallel Execution Analysis

- $\text{LBL}_{\text{sequential}} = \text{sum of all layer benchmark latencies}$
- $\text{LBL}_{\text{parallel}} = \text{sum of layer benchmark latencies on the critical path}$
Q1&2. LBL and Parallel Execution Analysis

- Benanza Ratio (BR) = LBL / measured latency
  - $BR_{\text{sequential}} = BR_{\text{parallel}}$ for models without parallel modules
  - $BR_{\text{parallel}} < BR_{\text{sequential}} < 1$ for models with parallel modules

Data-independent layers are executed sequentially even though they could be run in parallel.

The sequential and parallel BR of 30 models using MXNet on Tesla V100
Overall, the software stack is more optimized on the recent GPUs (Turing and Volta) and for smaller batch sizes.
Q3. Convolution Algorithm Selection Analysis

- Recall that in benchmark generation the convolution API is invoked with all available algorithms.
- The Analyzer parses the cuDNN log to determine if the cuDNN algorithm used by the framework is optimal.
- Cases where the choice is suboptimal, and the potential latency improvement are reported.
Q3. Convolution Algorithm Selection Analysis

- Both recent and older GPU architectures can benefit from better cuDNN heuristics
Q4. Framework Inefficiency Inspection

- The expected cuDNN and cuBLAS API calls are known
- The Analyzer compares the model execution profile against the expected execution to pinpoint inefficiencies within the framework
- The Analyzer presents any deviation observed in cuDNN or cuBLAS API invocation’s parameters or their execution order
Q4. An Inefficiency in MXNet

- Using Benanza we observed that MXNet ONNX model loader adds a padding layer before every convolution layer
- Unnecessary if the convolution does not use asymmetric padding

The speedup achieved for ResNet50-v1 by applying the MXNet optimization
Q5. Layer Fusion Analysis

- Recall that Benanza generates benchmarks that target the cuDNN fused API
- The Analyzer traverses the model layers and looks for the fusion patterns
  - \( \text{Profit}_{\text{layer fusion}} = \text{LBL}_{\text{non-fused}} - \text{LBL}_{\text{fused}} \)
Q5. Layer Fusion Analysis

- ResNet50-v1 has the layer sequence pattern Conv -> Bias -> BatchNorm -> Activation
- Benanza reports the Conv -> Bias can be fused for better latency

The latency speedup for ResNet50-v1 if layer fusion was performed
Q6. Tensor Core Analysis

- Recall that Benanza generates benchmarks that target Tensor Cores
- The Analyzer determines if the target model execution utilizes Tensor Cores by looking at kernel names
  - \( \text{Profit}_{\text{TensorCore}} = \text{LBL}_{\text{non-TensorCore}} - \text{LBL}_{\text{TensorCore}} \)
Q6. Tensor Core Analysis

- TITAN V achieves significant speedup, up to 1.72x
- For smaller batch sizes, Tesla T4 benefits most from Tensor Cores

The latency speedup for ResNet50-v1 if Tensor Cores were used
Q1,2,3,5,6. Joint Optimizations

- Benanza can perform the analyses jointly
- Up to 1.95x and 1.8x speedup can be achieved by TITAN V and Tesla V100 respectively

The latency speedup for ResNet50-v1 if parallel execution, optimal algorithm selections, layer fusion, and Tensor Cores were used
The workflow is automated, and the user only needs to compile and run the generated code.

The Performance Database is continuously updated:
- For new models, only the newly introduced layers are benchmarked.
- Layer repeatability keeps the number of entries in the database in check.

Components are modular and can be extended with:
- New model parsers
- New cuDNN/cuBLAS API or algorithm
- Other runtimes that target other SW libraries or HW
Conclusion

- Benanza automatically generates layer-wise benchmarks for DL models to compute the “lower-bound” latency and inform optimizations on GPUs
- The design is sustainable and extensible, not limited to GPUs and affords other usages, e.g.
  - Helping DL compiler optimizations
  - Improving work scheduling for DLaaS
  - Model/framework/system advising for DL tasks
Thank you
More information in the paper

Cheng Li*¹, Abdul Dakkak*¹, Jinjun Xiong², Wen-mei Hwu¹
University of Illinois Urbana-Champaign¹, IBM Research²