A Systematic Methodology for Analysis of Deep Learning Hardware and Software Platforms

Yu (Emma) Wang, Gu-Yeon Wei, David Brooks
Harvard University

Contact: ywang03@g.harvard.edu
3/3/2020
Acknowledgement

Frank Chen, Glenn Holloway, Dan Janni, Peter Mattson, Lifeng Nai, David Patterson, Francesco Pontiggia, Parthasarathy Ranganathan, Vijay Reddi, Brennan Saeta, Zak Stone, Anitha Vijayakumar, Shibo Wang, Qiumin Xu, Doe Hyun Yoon, Cliff Young
Challenges with ML Benchmarking

● Diversity in deep learning models used
  ○ Problem Domains, Models, Datasets
● Pace of field
  ○ State-of-the-art models evolve every few months
● Varying evaluation metrics
  ○ Accuracy, Time to train, Latency of inference
● Multi-disciplinary field
  ○ Algorithms, Systems, Hardware, ML Software Stacks
### State of the art: MLPerf 0.6

<table>
<thead>
<tr>
<th>Area</th>
<th>Benchmark</th>
<th>Dataset</th>
<th>Model</th>
<th>Reference Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vision</strong></td>
<td>Image classification</td>
<td>ImageNet</td>
<td>ResNet-50</td>
<td>TensorFlow</td>
</tr>
<tr>
<td></td>
<td>Object detection</td>
<td>COCO 2017</td>
<td>Mask R-CNN</td>
<td>Pytorch</td>
</tr>
<tr>
<td></td>
<td>Object detection</td>
<td>COCO 2017</td>
<td>SSD-ResNet34</td>
<td>Pytorch</td>
</tr>
<tr>
<td><strong>Language/Audio</strong></td>
<td>Translation</td>
<td>WMT Eng-Germ</td>
<td>Transformer</td>
<td>TensorFlow</td>
</tr>
<tr>
<td></td>
<td>Speech recognition</td>
<td>WMT Eng-Germ</td>
<td>GNMT</td>
<td>PyTorch</td>
</tr>
<tr>
<td><strong>Commerce</strong></td>
<td>Recommendation</td>
<td>MovieLens-20M</td>
<td>NCF</td>
<td>PyTorch</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Reinforcement Learning</td>
<td>Go</td>
<td>Mini-go</td>
<td>TensorFlow</td>
</tr>
</tbody>
</table>
State of the art: MLPerf 0.6

<table>
<thead>
<tr>
<th>Area</th>
<th>Benchmark</th>
<th>Dataset</th>
<th>Model</th>
<th>Reference Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td>Image classification</td>
<td>ImageNet</td>
<td>ResNet-50</td>
<td>TensorFlow</td>
</tr>
<tr>
<td></td>
<td>Object detection</td>
<td>COCO 2017</td>
<td>Mask R-CNN</td>
<td>Pytorch</td>
</tr>
<tr>
<td></td>
<td>Object detection</td>
<td>COCO 2017</td>
<td>SSD-ResNet34</td>
<td>Pytorch</td>
</tr>
<tr>
<td>Language/Audio</td>
<td>Translation</td>
<td>WMT Eng-Germ</td>
<td>Transformer</td>
<td>TensorFlow</td>
</tr>
<tr>
<td></td>
<td>Speech recognition</td>
<td>WMT Eng-Germ</td>
<td>GNMT</td>
<td>PyTorch</td>
</tr>
<tr>
<td>Commerce</td>
<td>Recommendation</td>
<td>MovieLens-20M</td>
<td>NCF</td>
<td>PyTorch</td>
</tr>
<tr>
<td>Action</td>
<td>Reinforcement Learning</td>
<td>Go</td>
<td>Mini-go</td>
<td>TensorFlow</td>
</tr>
</tbody>
</table>
Our Methodology

ParaDnn
Our Methodology

Paradnn
ParaDnn vs MLPerf

ParaDnn

- Avoid drawing conclusions based on several arbitrary models
- Generate **thousands** of parameterized, end-to-end models
- Prepare hardware designs for future models
- Complement the use of existing real-world models, i.e. MLPerf

MLPerf

- Good for studying accuracy or convergence with real datasets
- Represent the specific models some people care about
ParaDnn Canonical Models

Fully Connected (FC)

CNNs: Residual, Bottleneck

RNNs: RNN, LSTM, GRU
Models

![Graph showing the number of trainable parameters for different models: Transformer, RetinaNet, ResNet-50, DenseNet, MobileNet, SqueezeNet. The x-axis represents the number of trainable parameters on a logarithmic scale, while the y-axis lists the model names. The graph shows an increasing trend in trainable parameters as the model complexity increases.]
Models

- ParaDnn covers a larger range than the real models
  - from 10k to ~1 billion parameters
Analysis Enabled by ParaDnn

- Roofline analysis of TPU v2
- Homogenous Platform Comparison: TPU v2 vs v3
- Heterogeneous Platform Comparison: TPU vs GPU
The Roofline Model
The Roofline Model

Peak FLOPS
The Roofline Model

- Peak FLOPS
- Memory Bandwidth
- Floating Ops/Byte
- GFLOPS
- 10^0 to 10^5
- 10^3 to 10^4
The Roofline Model

![Graph showing the relationship between GFLOPS and Floating Ops/Byte]
The Roofline Model

![Diagram showing the Roofline Model with axes labeled GFLOPS and Floating Ops/Byte, illustrating the intersection between memory-intensive and compute-intensive regions.]
Transformer

![Graph showing the relationship between GFLOPS and Floating Ops/Byte. The graph has a logarithmic scale on both axes. The Transformer model is marked with a star.](image)
ParaDnn sweeps a large range of models, from memory-bound to compute-bound.
FC Models

Compute-bound

bs: 512 → 16k
FC Models

Memory-bound

![Graph showing GFLOPS vs Floating Ops/Byte with data points for different models and the Transformer.
TPU v2 vs v3?
How to upgrade to TPU v3?
How to upgrade to TPU v3?
How to upgrade to TPU v3?

TPU v3 (FLOPS↑)
TPU v3 (Mem BW↑)
TPU v2
How to upgrade to TPU v3?
How to upgrade to TPU v3?
Architecture of TPU v2 vs v3

Figure is from https://cloud.google.com/tpu/docs/system-architecture
Google’s Choice of TPU v3

- TPU v3
- TPU v2

Comparison:
- TPU v3 is 2.3 times faster than TPU v2.
- There is a question mark indicating uncertainty in the performance comparison.
TPU v3 vs v2: FC Operation Breakdown

![Diagram showing the comparison between TPU v3 and v2 for FC (fully connected) operations. The graph plots FLOPS v3/v2 against Floating Ops/Byte, indicating Mem-Bound performance. MatMul data points are highlighted with stars.]
TPU v3 vs v2: FC Operation Breakdown

Compute-bound: 2.3x speedup
TPU v3 vs v2: FC Operation Breakdown

Memory-bound: 1.5x speedup
TPU v3 vs v2: FC Operation Breakdown

Memory-bound, but benefit from 2x memory capacity:

3x speedup
Google’s Choice of TPU v3

- TPU v2
- TPU v3

1.5 x
2.3 x
ParaDnn provides diverse set of operations, and shows different operations are sensitive to different system component upgrades.
TPU vs GPU?
# Hardware Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unit</th>
<th>Version</th>
<th>Mem Type</th>
<th>Mem (GB)</th>
<th>Mem Bdw (GB/s)</th>
<th>Peak FLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (DGX-1)</td>
<td>1 Pkg</td>
<td>V100 (SXM2)</td>
<td>HBM2</td>
<td>16</td>
<td>900</td>
<td>125T</td>
</tr>
<tr>
<td>TPU</td>
<td>1 Board (8 cores)</td>
<td>v2</td>
<td>HBM</td>
<td>8</td>
<td>2400</td>
<td>180T</td>
</tr>
</tbody>
</table>
## Hardware Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unit</th>
<th>Version</th>
<th>Mem Type</th>
<th>Mem (GB)</th>
<th>Mem Bdw (GB/s)</th>
<th>Peak FLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (DGX-1)</td>
<td>1 Pkg</td>
<td>V100 (SXM2)</td>
<td>HBM2</td>
<td>16</td>
<td>900</td>
<td>125T</td>
</tr>
<tr>
<td>TPU</td>
<td>1 Board (8 cores)</td>
<td>v2</td>
<td>HBM</td>
<td>8</td>
<td>2400</td>
<td>180T</td>
</tr>
</tbody>
</table>

300 GB/s per core
FC and CNN
FC and CNN

Fewer Weights

Larger Conv ops
## Hardware Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unit</th>
<th>Version</th>
<th>Mem Type</th>
<th>Mem (GB)</th>
<th>Mem Bdw (GB/s)</th>
<th>Peak FLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (DGX-1)</td>
<td>1 Pkg</td>
<td>V100 (SXM2)</td>
<td>HBM2</td>
<td>16</td>
<td>900</td>
<td>125T</td>
</tr>
<tr>
<td>TPU</td>
<td>1 Board (8 cores)</td>
<td>v2</td>
<td>HBM</td>
<td>8</td>
<td>2400</td>
<td>180T</td>
</tr>
</tbody>
</table>

300 GB/s per core
FC TPU/GPU Speedup colored with Batch Size

[Graph showing speedups vs. parameters for different batch sizes (bs-1k, bs-2k, bs-4k, bs-8k, bs-16k)]
FC TPU/GPU Speedup colored with **Batch Size**

The diagram shows the speedup of FC operations on TPU and GPU, with speedup on the y-axis and parameters on the x-axis. The color of the data points represents different batch sizes: bs-1k, bs-2k, bs-4k, bs-8k, and bs-16k. The line at 0.35 indicates the point where TPU is better, and the line at 1.0 indicates where GPU is better.
FC TPU/GPU Speedup colored with Batch Size

FC: TPU/GPU

Speedups

10^1

10^0

10^{-1}

Params

10^7

10^8

TPU is better

GPU is better

9

0.35

bs-1k

bs-2k

bs-4k

bs-8k

bs-16k
FC TPU/GPU Speedup colored with **Node Size**

- More nodes
  - More weights
  - More memory-bound
## Hardware Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unit</th>
<th>Version</th>
<th>Mem Type</th>
<th>Mem (GB)</th>
<th>Mem Bdw (GB/s)</th>
<th>Peak FLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (DGX-1)</td>
<td>1 Pkg (SXM2)</td>
<td>V100</td>
<td>HBM2</td>
<td>16</td>
<td>900</td>
<td>125T</td>
</tr>
<tr>
<td>TPU</td>
<td>1 Board (8 cores)</td>
<td>v2</td>
<td>HBM</td>
<td>8</td>
<td>2400</td>
<td>180T</td>
</tr>
</tbody>
</table>

300 GB/s per core
CNN TPU/GPU Speedup colored with Batch Size
CNN TPU/GPU Speedup colored with **Batch Size**

- Up to 6x speedup
- TPU architecture and software is highly optimized for CNNs
CNN TPU/GPU Speedup colored with **Batch Size**

- All models run faster on TPU.
- Larger batch sizes lead to higher speedups.
CNN TPU/GPU Speedup colored with Filters

- More filters have higher speedup lower bounds
Conclusion

- Parameterized methodology: ParaDnn + a set of analysis methods
- Single platform analysis: TPU v2
- Homogenous platform comparison: TPU v2 vs v3
- Heterogeneous platform comparison: TPU vs GPU
Limitations of this Work

- Does not include:
  - Inference
  - Multi-node system: multi-GPU, or TPU pods
  - Accuracy, convergence
  - Cloud overhead

- Tractability
  - Limit the range of hyperparameters and datasets
    - Small batch sizes (<16) and large batch sizes (> 2k) are not studied
    - Synthetic datasets do not include data infeed overhead
  - Iterations of TPU loop is 100. Larger numbers can slightly increase the performance.
ParaDnn
Available: github.com/Emma926/paradnn

Questions?