MLPerf
Jun Huang
10.22
What performance number do we expect from a paper?

Experiments show RAMMER significantly outperforms state-of-the-art compilers such as TensorFlow XLA and TVM by up to $20.1 \times$. It also outperforms TensorRT, a vendor optimized proprietary DNN inference library from NVIDIA, by up to $3.1 \times$.

Does it really possible have $\sim 20x$ speed up?
Reasons:
1. Cherry-picked results
2. Compared to bad baseline
3. Extreme case
Simple comparisons between Data Parallelism and Model Parallelism:

Common sense: DP > MP, if memory allow

In extreme case: MP > DP, MP send activation, DP send gradient
Methods to eliminate such over-clarified experiment

1. Standardized everything (MLperf)

2. A better measuring method (Task Bench)
Mlperf training requirements:

1. Specified model.

2. Specified dataset.

3. Converge to predefined loss.
## Benchmarks for Training v1.0

<table>
<thead>
<tr>
<th>Area</th>
<th>Problem</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td>Image classification</td>
<td>ResNet-50 v1.5</td>
</tr>
<tr>
<td></td>
<td>Image segmentation (medical)</td>
<td>3D-UNET (new)</td>
</tr>
<tr>
<td></td>
<td>Object detection (light weight)</td>
<td>SSD</td>
</tr>
<tr>
<td></td>
<td>Object detection (heavy weight)</td>
<td>Mask R-CNN</td>
</tr>
<tr>
<td>Language</td>
<td>Speech recognition</td>
<td>RNN-T (new)</td>
</tr>
<tr>
<td></td>
<td>NLP</td>
<td>BERT</td>
</tr>
<tr>
<td>Commerce</td>
<td>Recommendation</td>
<td>DLRM</td>
</tr>
<tr>
<td>Research</td>
<td>Reinforcement learning</td>
<td>Mini Go</td>
</tr>
</tbody>
</table>

*Retired GNMT and Transformer in v1.0 to make room for 3D-UNET and RNN-T*

Training metric is simple: time to converge

source: MLCommons
Processor(accelerator): A100, TPU, Ascend 910, Graphcore, Habana

Framework: Tensorflow, Pytorch, Mxnet, Jax, Marlin, Mindspore

Distributed strategies: Data parallelism, model parallelism, spatial parallelism, async training

Reference:

Leaderboard on Mlperf v1.0: NVIDIA lead 4 best results. (4096 A100s)
Google lead 4 best results (3456 TPUv4).

Benchmarks for Training v1.0

<table>
<thead>
<tr>
<th>Area</th>
<th>Problem</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td>Image classification</td>
<td>ResNet-50 v1.5</td>
</tr>
<tr>
<td></td>
<td>Image segmentation (medical)</td>
<td>3DUNET (new)</td>
</tr>
<tr>
<td></td>
<td>Object detection (light weight)</td>
<td>SSD</td>
</tr>
<tr>
<td></td>
<td>Object detection (heavy weight)</td>
<td>Mask R-CNN</td>
</tr>
<tr>
<td>Language</td>
<td>Speech recognition</td>
<td>RNN-T (new)</td>
</tr>
<tr>
<td>NLP</td>
<td></td>
<td>BERT</td>
</tr>
<tr>
<td>Commerce</td>
<td>Recommendation</td>
<td>DLRM</td>
</tr>
<tr>
<td>Research</td>
<td>Reinforcement learning</td>
<td>Mini Go</td>
</tr>
</tbody>
</table>

*Retired GNMT and Transformer in v1.0 to make room for 3D-UNET and RNN-T*
Insights:

1. Full-stack challenge

2. Individual optimizations are simple (optimizing communication collectives, kernel fusion, overlap computation and communication)

3. Combing different optimizations into an ordered system are hard. (optimization may infer to each other, huge search space, no good solution for now)
Inference is all about QoS

Inference metrics: one metric for each scenario

- **Single stream**
  - e.g. cell phone augmented vision
  - **Latency**

- **Multiple stream**
  - e.g. multiple camera driving assistance
  - **Number streams**
    - subject to latency bound

- **Server**
  - e.g. translation site
  - **QPS**
    - subject to latency bound

- **Offline**
  - e.g. photo sorting
  - **Throughput**

MLPerf
Task Bench

Jun Huang

10.22
Two issues:
1. Benchmark took so much time
2. Indicator not clear
Normal way to do benchmark

Benchmark 1, Benchmark 2, ..., Benchmark N-1, Benchmark N

System 1, System 2, ..., System M-1, System M

It takes $n \times m$ runs to finish.
Charactize core representations of task into API

Benchmark 1, Benchmark 2, ..., Benchmark N-1, Benchmark N

Task Bench core API

System 1, System 2, ..., System M-1, System M

Only n + m runs needed, measure each API runtime in system, then apply to benchmark.
**Task = Kernel**

Executed at every point in a task graph. Available kernels:

- Empty
- Compute-bound (achieves peak compute)
- Memory-bound (achieves peak memory BW)
- Load-imbalanced (randomly varying duration)
- Also an *extensible* set
Indicators used most often:


2. tasks per second (TPS): hard to interpret. Empty task to show overhead. Larger task hide overhead.
Better indicator: minimum effective task granularity (METG)

- METG(50%) is the task granularity such that an application achieves overall 50% efficiency
- Parameterized on the efficiency metric:
  - E.g.: machine’s peak performance is 1.2 TFLOP/s, so 50% is 0.6 TFLOP/s
  - E.g.: application’s peak is $1 \times 10^9$ mesh cells/s, so 50% is $0.5 \times 10^9$
- **Efficiency constrained**: by definition, useful work is performed
- **Exposes overhead**: naturally finds the limit of the system under load
**Step 1:** measure performance with decreasing problem size

**Step 2:** convert to efficiency vs. task granularity and intersect with 50%

---

**Problem size:** repeat how many kernels per task.

**Task granularity:** \(\frac{\text{Time}_{\text{task}}}{\text{Number}_{\text{repeat kernel}}}\)
Intuition:

METG: Task granularity in an acceptable efficiency.

If System A and System B have average METG 1ms and 10ms. For application has X ms in METG(oracle).

1. X=100, good efficiency.
2. X=10, A and B have efficiency>50%
3. X=1, A has 50% efficiency while B has very low efficiency

Conclusion: METG is a good indicator for overhead. Problem size free, realistic.
Limitation:

1. Only focus on performance.

2. Using standardized/simplified kernel.
Insights:

1. Characterized benchmark into api make sense, clearly over-simplify the problem, but a must-have for METG as a research paper.

2. METG is a genius indicator, very insightful, simple and clear. Simple number but contains many preconditions.
Reference:
5. https://github.com/mlcommons/policies