SW/HW CO-OPTIMIZATION ON THE IPU: AN MLPERF™ CASE STUDY

Dr. Mario Michael Krell
OUTLINE

• Introduction
• Packed BERT
• Parallelism and recomputation
• Other features
• Conclusion
INTRODUCTION
RAISING THE BAR: GRAPHCORE’S FIRST MLPERF™ RESULTS

**BERT MLPerf v1.0 Training**

<table>
<thead>
<tr>
<th>IPU-POD_{16}</th>
<th>Closed</th>
<th>34.49 mins</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Open</td>
<td>27.75 mins</td>
</tr>
<tr>
<td>IPU-POD_{64}</td>
<td>Closed</td>
<td>11.96 mins</td>
</tr>
<tr>
<td></td>
<td>Open</td>
<td>9.39 mins</td>
</tr>
</tbody>
</table>

**ResNet-50 MLPerf v1.0 Training**

<table>
<thead>
<tr>
<th>IPU-POD_{16}</th>
<th>Closed</th>
<th>37.12 mins</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPU-POD_{64}</td>
<td>Closed</td>
<td>14.48 mins</td>
</tr>
</tbody>
</table>

MLPerf v1.0 Training Results | MLPerf ID: 1.0-1025, 1.0-1026, 1.0-1027, 1.0-1028, 1.0-1098, 1.0-1099
The MLPerf name and logo are trademarks. See www.mlperf.org for more information.
MK2 GC200 IPU

- Massively parallel MIMD
- Supports
  - FP32,
  - FP16, and
  - FP16 with stochastic rounding

Image: https://www.graphcore.ai/products/ipu
IPU-Machine: M2000

- 4 x Colossus™ GC200 IPU
- 1 petaFLOPS AI compute
- Up to 526GB Exchange Memory™
- 2.8Tbps IPU-Fabric™

Each Colossus™ GC200 IPU

- 59.4Bn transistors, TSMC 7nm @ 823mm²
- 250 teraFLOPS AI compute
- 1472 independent processor cores
- 8832 separate parallel threads

IPU-Gateway SoC

- Arm Cortex-A quad-core SoC
- Super low latency IPU-Fabric™ interconnect

Board Management Controller

- M.2 Slot
- PCIe FH3/4L G4x8 Slot (RNIC/SmartNIC)

DDR4 DIMM DRAM x 2

For model states streaming

Network-based disaggregated architecture for host vs. IPU resources

Single server in reference design

TOR switch (Arista 7060X, 32x100G + 2 10G)

Management switch (Arista 7010T, 48p 1G+ 4x1/10G)

Advanced air cooling system

Power Supply Unit (x2)

Sixteen M2000s

Ultra compact 1U server chassis

- 100Gbe Host-Link (QSFP)
- 1Gbe Management (Cat5)
- Sync-Link (Cat5)
- IPU-Link (OSFP)

eMMC 32G Flash device

POD 64

Left: https://www.graphcore.ai/hubfs/IPU-M2000SVG_2.svg
MORE DETAILS ON HARDWARE?

Visit presentation by Graphcore CTO Simon Knowles on Tuesday or visit our webpage
OUR MLPERF™ IMPLEMENTATION:
FULL STACK SOFTWARE/HARDWARE CO-OPTIMIZATION

- Algorithm: packed BERT
- Software: Parallelism + Recompute
- Hardware: FP16 + stochastic rounding, low batch size
**MOTIVATION (MODEL)**

- **BERT**
  - Transformer based model
  - Bidirectional encoder with self-attention
- Efficient use of ALU on the HW requires
  - Batching
  - Padding to same length
- Trade-off: wasted compute for speed-up from the parallelism
- **How much compute are we actually wasting?**

MOTIVATION (DATA)

- Wikipedia
- SQuAD 1.1
- GLUE

max. sequence length 512
theoretical max. speed-up: 2.001

SQuAD 1.1

Sequence length distributions of GLUE Datasets
SOLUTION

**Packing** in three core steps:

1. Pack the data
2. Adjust the model
3. Adjust the hyperparameters
Packing N sequences is an NP hard problem even when combining 3 sequences.

Impossible on raw sequences -> use histogram of sequences.

Online shortest-pack-first histogram-packing (SPFHP)

Offline Non-negative least-squares histogram-packing (NNLSHP)

SPFHP video from: https://towardsdatascience.com/introducing-packed-bert-for-2x-faster-training-in-natural-language-processing-eadb749962b1
ADJUSTING THE MODEL

Mandatory: Masking attention

```
1 mask = np.array([[1, 1, 1, 2, 2]])  # input
2 zero_one_mask = tf.equal(mask, mask.T)  # 0, 1 mask
3 # for use with softmax:
4 softmax_mask = tf.where(zero_one_mask, 0, -1000)
```

Mandatory: Adjust positional encoding

Optional: Vectorized unpacking for sequence based evaluation like NSP
ADJUSTING HYPERPARAMETERS

LAMB

- Standard optimizer used in BERT
- Calculates weighted moments m and v
  \[
  g_t = \frac{1}{|S_t|} \sum_{s_t \in S_t} \nabla \ell(x_t, s_t).
  \]
  \[
  m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t
  \]
  \[
  v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
  \]
- weights for moments
  power-dependent on packing factor p

\[
\beta_1 := \beta_1^p, \quad \beta_2 := \beta_2^p
\]

PACKED BERT EXPERIMENTS
# EXPERIMENTS: PACKING WIKIPEDIA

<table>
<thead>
<tr>
<th>pack. depth</th>
<th>pack. algo.</th>
<th># packs [M]</th>
<th>efficiency (%)</th>
<th>pack. factor</th>
<th>overhead (%)</th>
<th>realized speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
<td>16.280</td>
<td>49.97</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>SPFHP</td>
<td>10.102</td>
<td>80.52</td>
<td>1.612</td>
<td>4.283</td>
<td>1.544</td>
</tr>
<tr>
<td>3</td>
<td>SPFHP</td>
<td>9.095</td>
<td>89.44</td>
<td>1.790</td>
<td>4.287</td>
<td>1.716</td>
</tr>
<tr>
<td>3</td>
<td>NNLSHP</td>
<td>8.155</td>
<td>99.75</td>
<td>1.996</td>
<td>4.287</td>
<td><strong>1.913</strong></td>
</tr>
<tr>
<td>4</td>
<td>SPFHP</td>
<td>8.659</td>
<td>93.94</td>
<td>1.880</td>
<td>4.294</td>
<td>1.803</td>
</tr>
<tr>
<td>8</td>
<td>SPFHP</td>
<td>8.225</td>
<td>98.90</td>
<td>1.979</td>
<td>4.481</td>
<td>1.895</td>
</tr>
<tr>
<td>16/max</td>
<td>SPFHP</td>
<td>8.168</td>
<td>99.60</td>
<td>1.993</td>
<td>4.477</td>
<td>1.905</td>
</tr>
</tbody>
</table>
SO, DO WE GET THE DESIRED 2X SPEED-UP?
Yes!

- FLOPS only reduced by a factor of 1.913,
- Less overhead of IO
- This leads to an additional speed-up which pushes us >2x

Plot shows runtime compared to packedBERT
Original is around 2x slower
ANY QUESTIONS ON PACKING?

• Read the blog and the paper
  • https://towardsdatascience.com/introducing-packed-bert-for-2x-faster-training-in-natural-language-processing-eadb749962b1
HYBRID MODEL- AND DATA PARALLELISM + RECOMPUTATION
HOST DISAGGREGATION

- BERT is not host bound
  - 1 host even for 64 IPUs
- Power and cost savings
- Combines model and data parallelism for job distribution
- 1 M2000 for POD 16 setup
PIPELINE MODEL PARALLELISM FOR BERT (FORWARD PASS)

IPU0
- Embeddings / 3 Encoder Layers
- Projection / Losses

IPU1
- 7 Encoder Layers

IPU2
- 7 Encoder Layers

IPU3
- 7 Encoder Layers

Left: https://www.graphcore.ai/posts/bert-large-training-on-the-ipu-explained
Note: To save memory, we recompute the local forward passes F1, F2, F3 as part of the backward passes B1, B2, B3.
Data parallelism supported in all frameworks by specifying the number of used IPUs.

Images: https://www.graphcore.ai/posts/bert-large-training-on-the-ipu-explained
**REPLICATED TENSOR SHARDING (RTS)**

- Reduce overall memory footprint
- Can be combined with variable offloading to streaming memory
- Path for even larger models

<table>
<thead>
<tr>
<th></th>
<th>IPU-0</th>
<th>IPU-i</th>
<th>IPU-N-1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td><img src="image1" alt="Baseline" /></td>
<td><img src="image2" alt="Baseline" /></td>
<td><img src="image3" alt="Baseline" /></td>
</tr>
<tr>
<td>RTS on optimizer states (os)</td>
<td><img src="image4" alt="RTS on optimizer states" /></td>
<td><img src="image5" alt="RTS on optimizer states" /></td>
<td><img src="image6" alt="RTS on optimizer states" /></td>
</tr>
<tr>
<td>RTS on gradients (g) + os</td>
<td><img src="image7" alt="RTS on gradients" /></td>
<td><img src="image8" alt="RTS on gradients" /></td>
<td><img src="image9" alt="RTS on gradients" /></td>
</tr>
<tr>
<td>RTS on parameters + os + g</td>
<td><img src="image10" alt="RTS on parameters" /></td>
<td><img src="image11" alt="RTS on parameters" /></td>
<td><img src="image12" alt="RTS on parameters" /></td>
</tr>
</tbody>
</table>
OTHER FEATURES
LOW BATCH SIZE TRAINING

• Large local batch size requires storing plenty of unused activations

• LBS 2 for BERT

• Gradient accumulation for larger global batch size to save update costs
LOW PRECISION TRAINING

- FP16
- Stochastic rounding
- Weight aggregation in FP32
- Master weights in FP16
- LAMB moments in FP16 (streamed from DRAM on demand)
RESNET 50 SETUP

BATCH SIZE & DISTRIBUTION

• Single IPU data parallel setup with 1-4 hosts

• Introduced convenient recomputation API in our TensorFlow 1.15 framework to boost BS to 20

DISTRIBUTED BATCH NORM

• Batch norm requires statistics of at least 32 samples

• We combine data from 2 IPUs for a sufficient batch of 40 samples

• Simple configuration variable assignment
CONCLUSION

• Several SW strategies applied for amazing performance
• Strategies not benchmark specific: Generalize to variety of models/datasets
• Intuitive support of hybrid data- and model-parallel training including recomputation in TensorFlow, PyTorch, and our own Poplar Advanced Run Time (PopART)
• Try PackedBERT!
THANK YOU

Dr. Mario Michael Krell

Looking for new applications!
We are hiring!
### EXPERIMENTS:

#### PACKING SQUAD 1.1

<table>
<thead>
<tr>
<th>pack. depth</th>
<th>pack. algo.</th>
<th># strat. used</th>
<th># packs</th>
<th># tokens</th>
<th># padding tokens</th>
<th>efficiency (%)</th>
<th>pack. factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
<td>348</td>
<td>88641</td>
<td>34038144</td>
<td>18788665</td>
<td>44.801</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>SPFHP</td>
<td>348</td>
<td>45335</td>
<td>17408640</td>
<td>2159161</td>
<td>87.597</td>
<td>1.955</td>
</tr>
<tr>
<td>3</td>
<td>NNLSHP</td>
<td>398</td>
<td>40808</td>
<td>15670272</td>
<td>420793</td>
<td>97.310</td>
<td>2.172</td>
</tr>
<tr>
<td>3/max</td>
<td>SPFHP</td>
<td>344</td>
<td>40711</td>
<td>15633024</td>
<td>383545</td>
<td>97.547</td>
<td>2.177</td>
</tr>
</tbody>
</table>