Optimizing Efficiency of Deep Learning Workloads through GPU Virtualization

Presenters:
Tim Kaldewey – Performance Architect, Watson Group
Michael Gschwind – Chief Engineer ML & DL, Systems Group
David K. Tam – Performance Analyst, Watson Health Group
Disclaimer

The author's views expressed in this presentation do not necessarily reflect the views of IBM.
DL @ scale
DL Scaling

Scaling #GPUs

Communication overhead
(16 MPI workers)
Efficiency

### Table: Efficiency [GFLOPS/W]

<table>
<thead>
<tr>
<th></th>
<th>K10</th>
<th>K20</th>
<th>K20X</th>
<th>K40</th>
<th>K80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency [GFLOPS/W]</td>
<td>20.4</td>
<td>15.6</td>
<td>16.82</td>
<td>21.3</td>
<td>29.1</td>
</tr>
</tbody>
</table>

**Lighting Facts Per Bulb**

- **Brightness**: 800 lumens
- **Estimated Yearly Energy Cost**: $1.57
- **Cost**: Based on 3 hrs/day, 11c/kWh
- **Life**: Based on 3 hrs/day
- **Light Appearance**: Warm, 2700K, Cool, 5000K
- **Energy Used**: 13 watts
Efficiency

Are DL workloads making efficient use of compute resources?
A Workload:

Ask a question about the weather

Watch the Natural Language Classifier categorize your weather-related question. In this demo, the classifier is trained to determine whether the question is related to temperature or conditions. The output includes the top classification and a confidence score.

Do I need an umbrella

Output

<table>
<thead>
<tr>
<th>Results</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Natural Language Classifier is 96% confident that the question submitted is talking about conditions.
A Workload:

Training setup
- Multi-layer CNN for language classification
- 100s of intents
- 1000s of sentences
- Small batches sizes
- Training 200 epochs
- Torch-based implementation using single GPU*

Hardware config*
- IBM S822LC
- 20 cores @ 3.5 GHz
- 512 GB RAM
- 2 x K80 GPUs
- 4 VMs with 5 cores, 120 GB mem, 1 GPU

*for more details on hardware/software configuration(s) please refer to S7368 - PowerAI: A Co-Optimized Software Stack for AI on Power
System utilization as a measure of efficiency

Utilization (%)

Time (seconds)

1 GPU chip, reported by nvidia-smi
5 POWER8 cores, reported by mpstat
**GPU computation as a measure of efficiency**

- Nvprof reports the #FLOPs

<table>
<thead>
<tr>
<th>Invocations</th>
<th>Metric Name</th>
<th>Metric Description</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel: void indexSelectLargeIndex</td>
<td>flop_count_sp</td>
<td>Floating Point Operations (Single Precision)</td>
<td>1700</td>
<td>6300</td>
<td>2343</td>
</tr>
<tr>
<td>737</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: sgemm_sm35_1dg_nt_128x8x128x16x16</td>
<td>flop_count_sp</td>
<td>Floating Point Operations (Single Precision)</td>
<td>6537216</td>
<td>23052288</td>
<td>8080324</td>
</tr>
<tr>
<td>4916</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: void THCudaTensor_pointwise</td>
<td>flop_count_sp</td>
<td>Floating Point Operations (Single Precision)</td>
<td>3847422</td>
<td>3847422</td>
<td>3847422</td>
</tr>
<tr>
<td>2459</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GPU computation as a measure of efficiency

- Nvprof reports the #FLOPs

<table>
<thead>
<tr>
<th>Invocations</th>
<th>Metric Name</th>
<th>Metric Description</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel: void indexSelectLargeIndex 737</td>
<td>flop_count_sp</td>
<td>Floating Point Operations(Single Precision)</td>
<td>1700</td>
<td>6300</td>
<td>2343</td>
</tr>
<tr>
<td>Kernel: sgemm_sm35_ldg_nt_128x8x128x16x16 4916</td>
<td>flop_count_sp</td>
<td>Floating Point Operations(Single Precision)</td>
<td>6537216</td>
<td>23052288</td>
<td>8080324</td>
</tr>
<tr>
<td>Kernel: void THCudaTensor_pointwis 2459</td>
<td>flop_count_sp</td>
<td>Floating Point Operations(Single Precision)</td>
<td>3847422</td>
<td>3847422</td>
<td>3847422</td>
</tr>
</tbody>
</table>

- Or knowing the network layout, #FLOP can be calculated ...

→ 670 Billion FLOP in 5.6 seconds (1 epoch) ≈ \(\text{120 GFLOPs}\) vs. \(\text{4.4 TFLOPs}\) peak on single K80 chip
Runtime in CUDA libraries as a measure of efficiency

On S822LC (K80)
- Training ~20 mins for 200 epochs
- 56 % spend on GPU computation

On S822LC for HPC (P100)
- Training ~12 mins for 200 epochs
- 38% spend on GPU computation
Increasing GPU utilization

- Better exploitation by single program
  - Alternative frameworks*

- GPU resource sharing
  - Concurrent tasks and virtualization

*S. Gupta et al., “Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study”, IEEE International Conference on Data Mining 2016 (ICDM 2016)
GPU Resource Sharing Options

• Assignment of GPUs to VMs (PCIe Pass-Through)
• Multiple GPU Contexts (Streams)
• Multi-Process Service (MPS)
• Multi-Process Service and Docker (MPS + Docker)
Assignment of GPUs to VMs (PCIe Pass-Through)

- Assign each physical GPU to a virtual machine (VM)
- Hypervisor driver maps GPU directly into assigned VM
- GPU driver runs inside VM

👍 Isolation

👎 Insufficient resource management granularity
  ⇒ Inefficient GPU utilization
Multiple GPU Contexts (Streams)

- Separate GPU execution contexts for compute kernels
- Improved parallelism between streams
  - Up to 32 streams

👍 True parallel execution
 الغذائي Limited to execution within a single process
Multi-Process Service (MPS)

- MPS Server implements full “virtual” CUDA interface for multiple client processes
  - Multiple processes share GPU
  - Non-preemptive time-shared scheduling
  - No isolation between data
Multi-Process Service and Docker (MPS + Docker)

- MPS Server implements full “virtual” CUDA interface for multiple docker containers
  - Multiple containers/VMs share GPU
  - Non-preemptive time-shared scheduling between containers
  - No isolation between data
Our Solution:
Multiple MPI clients per MPS instance

- Use MPI to scale to more processes
  - Attach multiple MPI processes to a single MPS GPU instance

- Expand GPU parallelism using existing MPI parallelism
Using MPS with MPI

Communication Network

Message Passing Interface

Processes

mps0 mps1 mps2 mps3

mps0 mps1 mps2 mps3

Processes

Message Passing Interface

Communication Network
MPS x MPI parallelism implementation

- Start up MPS instance for all GPUs (4 on our system)
- Increase # MPI workers
- Map (#MPI workers / #GPUs) MPI workers to a GPU – intercepted by MPS
Evaluating GPU sharing Efficiency

1. Multiple distinct training jobs sharing a single GPU – MPS

2. Single parallel/distributed training job on (multiple) GPU – MPS x MPI
Experiment setup

- **Training**: Language classification network
- **Network**: Multi-layer CNN
- **Implementation**: Torch-based
- **Training Set**: 100s of intents, 1000s of sentences, small batch size(s)
- **Experiment**: Measure time to train 200 epochs with increasing # of training jobs

“Firestone”
IBM Power System S822LC
- POWER8 CPUs
- 2 sockets
- 20 cores @ 3.5 GHz
- 512 GB RAM (DDR3)
- 2 K80 GPU cards
- PCIe gen3 CPU↔GPU
Training on Firestone (K80s)

Restricted to:
- 1 GPU chip
- 5 CPU cores
Training on Firestone (K80s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Number of Training Jobs vs. Training Time and Throughput Gain graph.
Training on Firestone (K80s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Throughput Gain

- 1.0x
- 1.5x
- 1.7x
- 1.9x
- 1.9x
- 1.9x
- 2.0x

CPU Utilization

- 12%
- 24%
- 45%
- 55%
- 65%
- 84%
- 100%

Training Time

- 20 mins
- 27 mins
- 46 mins
- 53 mins
- 62 mins
- 83 mins
- 102 mins

Number of Training Jobs
Training on Firestone (K80s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Number of Training Jobs

Throughput Gain

GPU Utilization

CPU Utilization

Training Time

Throughput Gain

Number of Training Jobs
How About Parallel/Distributed Training?

• **Training:** Language classification network
• **Network:** Multi-layer CNN
• **Implementation:** Torch-based, *distributed algorithm* using MPI
• **Training Set:** 10s of intents, **100,000s** of sentences
• **Experiment:** Measure time to train 200 epochs increasing the # MPI workers
Training Time with Distributed Algorithm(s) – MPS x MPI
How About the Latest Architecture?

“Firestone”
IBM Power System S822LC
- POWER8 CPUs
- 2 sockets
- 20 cores @ 3.5 GHz
- 512 GB RAM (DDR3)
- 2 K80 GPU cards
- PCIe gen3 CPU↔GPU

“Minsky”
IBM Power System S822LC for HPC
- POWER8 CPUs
- 2 sockets
- 20 cores @ 3.5 GHz
- 1 TB RAM (DDR4)
- 4 P100 GPU chips
- NVLink CPU↔GPU
Training on *Minsky* (P100s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Number of Training Jobs vs. Training Time

- 12 mins
- 12 mins
- 17 mins
- 19 mins
- 24 mins
- 35 mins
- 50 mins

![Graph showing the relationship between the number of training jobs and training time.](image)
Training on *Minsky* (P100s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

![Graph showing training time and throughput gain for various numbers of training jobs and throughput gains.](image-url)
Training on **Minsky** (P100s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

![Graph showing training time, CPU utilization, and throughput gain as a function of the number of training jobs. The graph includes data points for CPU utilization, training time, and throughput gain at different job counts.](image-url)
Training on Minsky (P100s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Throughput Gain, 5 cores

- 1.0x
- 2.0x
- 2.8x
- 3.0x
- 3.2x
- 3.0x
- 2.7x
- 2.4x

Number of Training Jobs:
- 12 mins
- 17 mins
- 19 mins
- 24 mins
- 35 mins
- 50 mins
Training on *Minsky* (P100s)

Restricted to:
- 1 GPU chip
- 5 CPU cores

Training on *Minsky* (P100s) with number of training jobs from 1 to 10, showing GPU utilization, CPU utilization, training time, and throughput gain for 5 and 10 cores.
Conclusions
Conclusions

• Optimization is a multidimensional space – need to consult multiple(!) metrics
Conclusions

• Optimization is a multidimensional space – need to consult multiple(!) metrics

• GPU sharing
  a) increases system efficiency by running multiple training jobs
  b) reduces training time by enabling multiple #MPI workers per GPU
Conclusions

• Optimization is a multidimensional space – need to consult multiple(!) metrics

• GPU sharing
  a) increases system efficiency by running multiple training jobs
  b) reduces training time by enabling multiple #MPI workers per GPU

• Sharing GPUs yields >3x efficiency gains
  ⇒ off-the-shelf 4 GPU workstation becomes a 12 GPU DL supercomputer
Conclusions

• Optimization is a multidimensional space – need to consult multiple(!) metrics

• GPU sharing
  a) increases system efficiency by running multiple training jobs
  b) reduces training time by enabling multiple #MPI workers per GPU

• Sharing GPUs yields >3x efficiency gains
  ⇒ off-the-shelf 4 GPU workstation becomes a 12 GPU DL supercomputer

The full details of this work will appear, in final form, in the IBM Journal of Research and Development, vol. 61, no. 4/5, 2017, as part of a special issue on “Deep Learning.” Please cite the IBM Journal official paper version of record. For more information on the journal, see: http://www.research.ibm.com/journal/.