XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs

Cheng Li*, Abdal Dakak*, Jinjun Xiong, Wei Wei, Lingjie Xu, Wen-mei Hwu

University of Illinois Urbana-Champaign, IBM Research, Alibaba Group

{cli99, dakkak, w-hwu}@illinois.edu, jinjun@us.ibm.com, {w.wei, lingjie.xu}@alibaba-inc.com

Video: https://youtu.be/v95JfmM66eE
Background

- Machine Learning (ML) models are used in many application domains
- Understanding ML inference performance is an increasingly pressing but challenging task

Slow adoption of DL innovations
ML Model

A graph where each vertex is a layer (or operator) and an edge represents data transfer.

Example: ResNet50
ML Inference Pipeline

Input Image

Pre-processing
- Image decoding
- Resizing
- Normalization
- Type conversion

Prediction
Input Tensor
Model prediction using framework API
Output Tensor

Post-processing
Unpacking into pairs (label, probabilities) and sorting

Top1
(dog, 0.99)
A holistic view of the model execution is needed

Existing profiling tools are disjoint
  – Profiling at different granularities means switching between tools
  – No correlation between profiles
XSP Motivation

- Inference is impacted by the interplay between levels of the HW/SW stack
- Any of them can be a bottleneck
Current DL Profiling on GPUs

1. Using code insertion

2. Using framework profiler

3. Using nvprof or Nsight

One has to manually perform the difficult task of correlating these disjoint profiles

Model-, layer-, and GPU kernel-level profiles of MLPerf ResNet50 v1.5 with batch size 256 on a Volta GPU
An Approach - Modifying Frameworks

- NGC frameworks (TensorFlow, PyTorch, etc.) are instrumented with NVTX markers
  - GPU profile with layer annotations, lacks framework profiling
  - May inhibit frameworks from performing some optimizations
  - Does not work for DL models that use customized frameworks

- TensorFlow profiler
  - framework profile with some GPU profiling
  - Does not work for other frameworks

- **Vendor lock-in & limited applicability**
XSP: Across-stack Profiling

- Incorporates profile data from different sources to obtain a holistic and hierarchical view of DL workloads
  - Innovatively leverages distributed tracing
- Accurately captures the profiles at each HW/SW stack level despite the profiling overhead
  - Leveled experimentation methodology
- Coupled with an automated analysis pipeline
- Reveals insights that would otherwise be difficult to discern
Distributed Tracing

- Designed to monitor distributed applications (e.g. microservices)

- Key Concepts
  - **Span**: a named, timed operation representing a piece of the workflow
    - **Start & end timestamps**
    - **Tags & Logs**: key-value pairs of user-defined annotation or logging messages for spans
    - **SpanContext**: a state to refer to a distinct span
  - **Trace**: a tree of spans
  - **Tracer**: an object that creates and publishes spans
An Example

An application with services (A, B, C, D, E, F) that have causal relationships.

Application Timeline:
- A
- B
- C
- D
- E
- F

Tracing Workflow:
- Host 0
  - Application
  - Tracer(s)
  - Publish Spans
- Host 1
  - Tracer(s)
  - Publish Spans
  - Tracing Server

Spans:
- A
- B
- C
- D
- E
- F
Leveraging Distributed Tracing in XSP

- Observe the similarity between profiling and distributed tracing
- Turn profilers into tracers
- Convert profiled events into spans
- Multiple tracers can exist within a stack level
- Tracers can be enabled/disabled
Constructing Parent/Child Relationships

- Tracers use the system clock
- Spans are time intervals and assigned with levels
- During the profile analysis, check interval inclusion
  - If interval s1 contains interval s2 and s1 is a level higher than s2, then s1 is a parent of s2
Capturing Asynchronous Events

- E.g. Asynchronous GPU kernel launches
- Capture both the kernel launch and execution spans
  - Use the kernel launch span to figure out the parent span
  - Use the kernel execution span to get performance information or figure out its children spans

```
conv
```

```
cudaLaunchKernel ...
```

```
kernel execution
```
Capturing Parallel Events

- E.g. Two conv layers overlap, and each invokes GPU kernels
- Serialize the conv layers to get their correlations to GPU kernels
- Or more complex post-processing
XSP for ML Inference on GPUs

Global Tracer:
User inserts tracing API (startSpan & finishSpan) to capture code sections

Framework Tracer:
Built on top of the framework profiling capability to capture layer level information

GPU Tracer:
Built on top of CUPTI to capture CUDA runtime API, GPU activities, GPU metrics

No change to DL frameworks or libraries

Model
Input Pre-Process → Model Inference → Output Post-Process

Layer
Data → Conv → BN → Relu → … → SoftMax

GPU Kernel

Kernel1
Name=ShuffleTensor
Grid=[10,1,1]
Block=[1024,1,1]

Kernel2
Name=OffsetComp
Grid=[99,1,1]
Block=[1024,1,1]

Kernel3
Name=VoltaCUDNN_128x64
Grid=[99,1,1]
Block=[256,1,1]

GPU Metrics
SP Flop Count=62GFlop
DRAM Read Bytes=12.1MB
DRAM Write Bytes=296MB
Achieved Occupancy=13.2%

Model-, layer-, and GPU kernel-level profiles of MLPerf ResNet50 v1.5 with batch size 256 on a Volta GPU
Dealing with Profiling Overhead

- Profiling always comes with overhead
- XSP uses leveled experimentation to get accurate timing for all levels


Profiling always comes with overhead

XSP uses leveled experimentation to get accurate timing for all levels.
Leveled Experimentation

- Profilers at level \( n \) accurately capture events at level \( n \)
- Use traces from runs with different profiling levels enabled
  - Overhead\(_n\) = Profile\(_{0/.../n-1}\) – Profile\(_{0/.../n}\)
Automated Across-stack Analysis

The 15 analyses performed by XSP using profiles from one or more levels

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Profiling Provider</th>
<th>End-to-End Benchmarking</th>
<th>Framework Profilers</th>
<th>NVIDIA Profilers</th>
<th>XSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 Model throughput and latency</td>
<td>M</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A2 Layer information</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A3 Layer latency</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A4 Layer allocated memory</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A5 Layer type distribution</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A6 Layer aggregated latency</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A7 Layer aggregated allocated memory</td>
<td>L</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A8 GPU information</td>
<td>G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A9 GPU roofline</td>
<td>G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A10 GPU aggregated information</td>
<td>G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A11 Layer aggregated GPU information</td>
<td>L/G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A12 Layer aggregated GPU metrics</td>
<td>L/G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A13 GPU vs CPU latency</td>
<td>L/G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A14 Layer roofline</td>
<td>L/G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>A15 Model roofline</td>
<td>M/L/G</td>
<td>x</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
The top 5 most time-consuming GPU kernel invocations

<table>
<thead>
<tr>
<th>Kernel Name</th>
<th>Layer Index</th>
<th>Layer Latency (ms)</th>
<th>Kernel Gflops</th>
<th>Kernel DRAM Reads (MB)</th>
<th>Kernel DRAM Writes (MB)</th>
<th>Kernel Achieved Occupancy (%)</th>
<th>Kernel Arithmetic Intensity (flops/byte)</th>
<th>Kernel Arithmetic Throughput (Tflops/s)</th>
<th>Memory Bound?</th>
</tr>
</thead>
<tbody>
<tr>
<td>volta_cgemm_32x32_tn</td>
<td>221</td>
<td>6.04</td>
<td>77.42</td>
<td>40.33</td>
<td>43.86</td>
<td>12.18</td>
<td>876.97</td>
<td>12.82</td>
<td>X</td>
</tr>
<tr>
<td>volta_cgemm_32x32_tn</td>
<td>208</td>
<td>6.03</td>
<td>77.42</td>
<td>43.93</td>
<td>43.81</td>
<td>12.19</td>
<td>841.59</td>
<td>12.83</td>
<td>X</td>
</tr>
<tr>
<td>volta_scudnn_128x128_relu_interior_nn_v1</td>
<td>195</td>
<td>5.48</td>
<td>59.20</td>
<td>27.71</td>
<td>8.40</td>
<td>15.49</td>
<td>1,563.30</td>
<td>10.80</td>
<td>X</td>
</tr>
<tr>
<td>volta_scudnn_128x64_relu_interior_nn_v1</td>
<td>3</td>
<td>4.91</td>
<td>62.89</td>
<td>11.55</td>
<td>283.05</td>
<td>13.20</td>
<td>203.58</td>
<td>12.81</td>
<td>X</td>
</tr>
<tr>
<td>volta_scudnn_128x128_relu_interior_nn_v1</td>
<td>57</td>
<td>4.56</td>
<td>59.24</td>
<td>34.83</td>
<td>37.64</td>
<td>15.15</td>
<td>779.55</td>
<td>12.99</td>
<td>X</td>
</tr>
</tbody>
</table>
XSP Extensibility

- Other profiling tools or methods can be integrated
  - More tracers at each stack level, e.g. CPU+GPU
  - Capture more stack levels, e.g. ML library level and application level
  - Work with accelerators and simulators
- Add more types of analyses
- Add ML training support
Conclusion

- XSP is an across-stack profiling design that aggregates profile data from different sources and correlates them to construct a holistic and hierarchical view of ML model execution
  - A smooth hierarchical step-through of model performance at different levels within the HW/SW stack to identify bottlenecks
  - Systematic comparisons of models, frameworks, and hardware through the consistent profiling and automated analysis workflows
  - Extensible to accommodate different use cases
Thank you
More information in the paper

Cheng Li*¹, Abdul Dakkak*¹, Jinjun Xiong², Wei Wei³, Lingjie Xu³, Wen-mei Hwu¹
University of Illinois Urbana-Champaign¹, IBM Research², Alibaba Group