Salus

Fine-grained GPU Sharing Primitives for Deep Learning Applications

Advisor: Mosharaf Chowdhury

2020-03-03 By Peifeng Yu
Deep Learning Becomes Ubiquitous

- Computer vision
- Natural language processing
- Speech
- Robotics

Applications
- Intelligent assistant: Google Now, Siri, Cortana
- Face recognition
- Video content understanding
A Brief Introduction to Deep Learning

- Training:
  - Forward & backward pass
  - Iterative
A Brief Introduction to Deep Learning

- Training:
  - Forward & backward pass
  - Iterative

- Inference:
  - Forward pass
# Accelerate Deep Learning with GPUs

<table>
<thead>
<tr>
<th>Neural Networks</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherently Parallel</td>
<td></td>
</tr>
<tr>
<td>Matrix Operations</td>
<td></td>
</tr>
<tr>
<td>FLOPS</td>
<td></td>
</tr>
</tbody>
</table>
Exclusive Access to GPU

An application can have multiple GPUs, but each GPU usually belongs to exactly one application at a time.

Advantages
- Simplifies hardware design
- Efficiency

Disadvantages
- Lack of flexibility
Exclusive Access: Lack of Flexibility

- Hinders the scheduling ability of GPU cluster managers
- Underutilization
  - Hyper-parameter tuning (AutoML)
  - Model serving (inference)
Exclusive Access: Lack of Flexibility

• Hinders the scheduling ability of GPU cluster managers
  • Starting or suspending job is expensive
  • Often easier to just do non-preemptive scheduling → FIFO
    • Head-of-line blocking
Exclusive Access: Lack of Flexibility

- Underutilization
  - Variance in memory usage → Overprovision

<table>
<thead>
<tr>
<th>Model</th>
<th>Peak Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE</td>
<td>28M</td>
</tr>
<tr>
<td>Super Resolution</td>
<td>529M</td>
</tr>
<tr>
<td>Deep Speech</td>
<td>3993M</td>
</tr>
<tr>
<td>Inception4</td>
<td>11355M</td>
</tr>
</tbody>
</table>
How Can We Efficiently Share a GPU for Deep Learning Applications?
### GPU Sharing

- Existing sharing solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Efficiency</th>
<th>Dynamic Memory</th>
<th>Flexible Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Partitioning (SP)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-Process Service (MPS)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Design Goals

<table>
<thead>
<tr>
<th>Approach</th>
<th>Efficiency</th>
<th>Dynamic Memory</th>
<th>Flexible Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Partitioning (SP)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-Process Service (MPS)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Ideal</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Minimize deployment overhead
  - No new hardware
  - No modification from user side
Salus Fine-grained GPU Sharing Primitives for Deep Learning

A consolidated execution service enabling sharing primitives

- Fast job switching,
- Memory sharing

without modifying any

- User scripts,
- Operating systems, or
- Hardware

with the goal to

- Support new scheduler for GPU,
- Improve GPU utilization
Salus in DL Stack

User scripts

Deep Learning Frameworks

Salus Execution Service

- CPU
- GPU
- FPGA
- ... ASIC

Salus Adaptor

Others
- CNTK
- PyTorch
- Tensorflow

ASIC
- FPGA
- GPU
- CPU
Salus Components

1. Salus Adaptor
   - Transfer computation graph

2. Salus Execution Service
   - Consolidates all GPU accesses
Salus in One Slide

- Create session
- Send computation graph
- For each iteration:
  - Send input
  - Check memory
  - Queue in scheduler
Sharing Primitives

- Efficient job switching
- Memory sharing: GPU lane abstraction
### Existing Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Time Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop and restart (checkpointing)</td>
<td>10~100s</td>
</tr>
<tr>
<td>Generate snapshot[1]</td>
<td>~1s</td>
</tr>
</tbody>
</table>

Bottleneck: data (memory) transfer

Understand DL Job Memory

- 3 types of memory:
  - Model
  - Ephemeral
  - Framework internal
Understand DL Job Memory

• 3 types of memory:
  • Model
  • Ephemeral
  • Framework-internal
• Data transfer time is non-negligible
  • Can be over 2X of corresponding inference latency
• Model memory << GPU memory capacity

Why not keep multiple jobs’ model in memory for fast switching?
Sharing Primitives: Efficient Job Switching

Job switching is done by determine which job’s iteration to run next.

- Minimal switching overhead
- Flexible scheduling policies

A trade-off between maximum utilization and execution performance
Sharing Primitives

- Efficient job switching

![Diagram showing sharing primitives]
Sharing Primitives

- Efficient job switching
- Memory sharing: GPU lane
Sharing Primitives: Memory Sharing

• Efficient job switching
• Memory sharing: GPU lane

= Continuous physical memory + GPU stream

• Time-slicing within lane, parallel across lanes
• Dynamic re-partitioning (lane assignment)
• Avoid in-lane fragmentation
GPU Lane: Best Fit & Safety Condition

- A lane cannot accept arbitrary number of jobs
- The Safety Condition determines whether a job can go in a lane

\[ \sum_i P_i + \max_i T_i \leq C_l \]

- \( P_i \): Model and framework-internal memory for job \( i \)
- \( T_i \): Ephemeral memory for job \( i \)
- \( C_l \): Memory capacity of lane \( l \)
GPU Lane: Best Fit & Safety Condition

- A lane cannot accept arbitrary number of jobs
- The Safety Condition determines whether a job can go in a lane

\[
\sum_i P_i + \sum_i T_i \leq C_l
\]

- \( P_i \): Model and framework-internal memory for job \( i \)
- \( T_i \): Ephemeral memory for job \( i \)
- \( C_l \): Memory capacity of lane \( l \)
FIFO is suboptimal
- HOL blocking
- Underutilization

With Salus
- Packing: achieves higher utilization
- Preemption: enables prioritization
- Fairness: equalizes the resource usage
- ...
- What's more? Still a huge design space!
Evaluation

1. Flexible scheduler
2. Faster hyper-parameter tuning
3. High GPU utilization for inference

Deployment and evaluation on Intel E5-2670 with 2x NVIDIA Tesla P100 with 15 workloads
A Production Trace

- 100 jobs from a production trace\([1]\)
- 4 schedulers implemented as demo
- SRTF vs FIFO: 3.19x improvement in Avg. JCT

Sub-second Level Switching

- Slice of the 100 job trace, time is normalized
- Sub-second switching
Hyper-parameter Exploration

- 2 sets of hyper-parameter exploration
- 300 exploration jobs in each set
- Makespan is important
Pack Inference Applications

- 42 DL inference applications in 1 GPU
- User facing services: latency
Salus  Fine-grained GPU Sharing Primitives for Deep Learning

Open sourced at: https://github.com/SymbioticLab/Salus
• Prebuilt docker image available