Scheduling for Efficient Large-Scale Machine Learning Training

Jinliang Wei
Carnegie Mellon University

Carnegie Mellon University
Computer Science Department
Machine Learning Training: Quest for Efficiency

Growing data size

Growing model complexity

Challenges:

- Machine learning models take long time to train.
- Machine learning training consumes large amount of memory.
- Implementing parallel/distributed programs is hard.
My Work: More Efficient ML Training via Scheduling

Key Idea:
Leverage general structural properties in ML computation to improve efficiency
My Work: More Efficient ML Training via Scheduling

**Key Idea:**
Leverage general structural properties in ML computation to improve efficiency

**Challenges:**
What structural properties are helpful?
My Work: More Efficient ML Training via Scheduling

Key Idea:
Leverage general structural properties in ML computation to improve efficiency
My Work: More Efficient ML Training via Scheduling

**Key Idea:**
Leverage general structural properties in ML computation to improve efficiency

**Challenges:**
What structural properties are helpful?
My Work: More Efficient ML Training via Scheduling

Key Idea:
Leverage general structural properties in ML computation to improve efficiency

Challenges:
What structural properties are helpful?
Generalizability across models / algorithms
My Work: More Efficient ML Training via Scheduling

**Key Idea:**
Leverage general structural properties in ML computation to improve efficiency

**Challenges:**
- What structural properties are helpful?
- Generalizability across models / algorithms
- How to leverage it with no / little burden to users?
My Work: More Efficient ML Training via Scheduling

Key Idea:
Leverage general structural properties in ML computation to improve efficiency

Challenges:
- What structural properties are helpful?
- Generalizability across models / algorithms
- How to leverage it with no / little burden to users?

Systems developed:
- Bösen: (parameter server) [SoCC’15] ~20K LoC (C++)
- Orion: (auto-parallelization) [EuroSys’19] ~23K LoC (C++, Julia)
- Non-trivial work on TensorFlow core
Scheduling within a Single Training Job
Scheduling within a Single Training Job
Scheduling within a Single Training Job

- Network Communication: When and what to send?
- Computation: What to compute in parallel?
- Memory Allocation: When and where to allocate?

Lead: [SoCC’15, Best paper] [EuroSys’19] [In preparation]
Coauthor: [ATC’17] [SysML’19]

Highlights of results:
- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- Scheduling memory: 10x bigger model on the same hardware
Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

repeat until convergence
    foreach mini-batch in dataset
        update model parameters
Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

repeat until convergence
  foreach mini-batch in dataset
    update model parameters

Many passes over training data
Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

repeat until convergence
foreach mini-batch in dataset
update model parameters

Many passes over training data
Many updates per data pass
Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

repeat until convergence
    foreach mini-batch in dataset
        update model parameters

Many passes over training data
Many updates per data pass
Machine Learning Training Is A Search Process

Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn’t mean it’s wrong
It often means more steps
Machine Learning Training Is A Search Process

Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn’t mean it’s wrong
It often means more steps

Convergence speed = samples/sec * convergence/sample
Machine Learning Training Is A Search Process

Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn’t mean it’s wrong
It often means more steps

Trade-off is possible

Convergence speed = samples/sec * convergence/sample
Background: Data Parallelism for Computation Throughput

Simply run some/all mini-batches in parallel, regardless of dependence.

repeat until convergence
  in parallel foreach mini-batch in dataset
  update model parameters

Convergence speed = \frac{\text{samples/sec} \times \text{convergence/sample}}{\text{sample size}}
Background: Data Parallelism for Computation Throughput

Simply run some / all mini-batches in parallel, regardless of dependence.

Repeat until convergence:
- In parallel, for each mini-batch in the dataset,
- Update model parameters

Convergence speed = samples/sec \times convergence/sample
Background: Data Parallelism for Computation Throughput
Simply run some/all mini-batches in parallel, regardless of dependence

```
repeat until convergence
  in parallel foreach mini-batch in dataset
  update model parameters
```

Convergence speed = samples/sec \times convergence/sample
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

- mini-batch #1
- mini-batch #2
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations.
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Data Parallelism

- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations.
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Data Parallelism
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Synchronization

Later iterations observe updates from earlier iterations
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Synchronization

Inconsistency: parallel iterations do not observe updates from each other
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Data Parallelism
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Synchronization

Later iterations observe updates from earlier iterations

Inconsistency: parallel iterations do not observe updates from each other

\[ W_1 = W_0 + \Delta W_1 \]
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations.

Data Parallelism:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Inconsistency: parallel iterations do not observe updates from each other.

Synchronization:

\[ W_1 = W_0 + \Delta W_1 \]

Serial:
\[ W_2 = W_0 + \Delta W_1 + \Delta W_2 \]
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations.

Data Parallelism:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Synchronization

Serial: $W_2 = W_0 + \Delta W_1 + \Delta W_2$

Parallel: $W_1 = W_0 + \Delta W_1$

Inconsistency: parallel iterations do not observe updates from each other.
Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism:
- mini-batch #1
- mini-batch #2
- mini-batch #3
- mini-batch #4

Synchronization

Data parallel:
- $\Delta W_1$  
  $W_1 = W_0 + \Delta W_1$

Serial:
- $\Delta W_2$
  $\Delta W_2$
  $\Delta W_2$
  $\Delta W_2$

$\bar{W}_2 = W_0 + \Delta W_1 + \Delta W_2$

Serial $W_2 = W_0 + \Delta W_1 + \Delta W_2$

Inconsistency: parallel iterations do not observe updates from each other
Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
Simple and highly sparse

Problem:
Light computation per mini-batch vs. heavy communication
Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
- Sparse Logistic Regression
- Latent Dirichlet Allocation (LDA)
- Matrix Factorization (MF)...

Simple and highly sparse

Problem:
Light computation per mini-batch vs. heavy communication

Compute
Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
Sparse Logistic Regression
Latent Dirichlet Allocation (LDA)
Matrix Factorization (MF)...

Simple and highly sparse

Problem:
Light computation per mini-batch vs. heavy communication

Compute: $D_1$ $D_2$ $D_3$

Communicate: $C_1$ $C_2$ $C_3$
Background: Trade Even More Consistency for Throughput

\[ w_1 \quad +2 \quad +3 \quad -1 \]
\[ w_2 \quad +1 \quad +2 \quad -2 \]
\[ w_3 \quad -3 \quad +1 \quad -3 \]
\[ w_4 \quad -3 \quad -3 \quad \]
Background: Trade Even More Consistency for Throughput

$w_1$  $w_2$  $w_3$  $w_4$

$\Delta W_1$  $\Delta W_2$  $\Delta W_3$

+2  +3  -1
+1  +2  +1
-3  -3  -2

Coalesce deltas to reduce communication
Background: Trade Even More Consistency for Throughput

\[
\begin{align*}
\Delta W_1 &\rightarrow +2 \\
\Delta W_2 &\rightarrow +3 \\
\Delta W_3 &\rightarrow -1 \\
\Delta W_1 + \Delta W_2 + \Delta W_3 &\rightarrow +4 \\
\end{align*}
\]

Coalesce deltas to reduce communication
Background: Trade Even More Consistency for Throughput

\[
\begin{align*}
    w_1 & : +2 & w_2 & : +3 & w_3 & : -1 & w_4 & : +4 \\
    \Delta W_1 & & \Delta W_2 & & \Delta W_3 & & \Delta W_1 + \Delta W_2 + \Delta W_3
\end{align*}
\]

Coalesce deltas to reduce communication

+ Local Buffering: communicate every N mini-batches, coalescing deltas

Compute

\[
\begin{align*}
    D_1 & & D_2 & & D_3 & & D_4 & & D_5 & & D_6
\end{align*}
\]

Communicate

\[
\begin{align*}
    C_1 & & C_2
\end{align*}
\]
Background: Trade Even More Consistency for Throughput

+ Local Buffering: communicate every N mini-batches, coalescing deltas

Compute

\begin{align*}
\Delta W_1 & \quad \Delta W_2 & \quad \Delta W_3 & \quad \Delta W_1 + \Delta W_2 + \Delta W_3 \\
+2 & \quad +3 & \quad -1 & \quad +4 \\
+1 & \quad +2 & \quad +3 & \quad +3 \\
+1 & \quad +1 & \quad -2 & \quad -1 \\
-3 & \quad -3 & \quad -2 & \quad -6
\end{align*}

Communicate

+ Bounded Staleness: block iff the fastest is T steps ahead of the slowest
Background: Trade Even More Consistency for Throughput

$w_1$ $w_2$ $w_3$ $w_4$  
$+2$ $+1$ $+1$ $-3$  
$+3$ $+2$ $-2$ $-3$  
$\Delta W_1$ $\Delta W_2$ $\Delta W_3$  
$-1$ $+2$ $-2$  
$\Delta W_1 + \Delta W_2 + \Delta W_3$  
$+4$ $+3$ $-1$ $-6$

Coalesce deltas to reduce communication

+ **Local Buffering**: communicate every N mini-batches, coalescing deltas

Compute

$D_1$ $D_2$ $D_3$  
$D_4$ $D_5$ $D_6$

Communicate

$C_1$ $C_2$

+ **Bounded Staleness**: block iff the fastest is T steps ahead of the slowest

Compute
Background: Trade Even More Consistency for Throughput

+ Local Buffering: communicate every N mini-batches, coalescing deltas

+ Bounded Staleness: block iff the fastest is T steps ahead of the slowest
Background: Trade Even More Consistency for Throughput

\[
\begin{align*}
&\Delta W_1 = +2 \\
&\Delta W_2 = +3 \\
&\Delta W_3 = -1 \\
&\Delta W_1 + \Delta W_2 + \Delta W_3 = +4
\end{align*}
\]

Coalesce deltas to reduce communication

+ **Local Buffering**: communicate every N mini-batches, coalescing deltas

Compute 
\[
\begin{align*}
&D_1 \\
&D_2 \\
&D_3 \\
&D_4 \\
&D_5 \\
&D_6
\end{align*}
\]

Communicate
\[
\begin{align*}
&C_1 \\
&C_2
\end{align*}
\]

+ **Bounded Staleness**: block iff the fastest is T steps ahead of the slowest

Compute 
\[
\begin{align*}
&D_1 \\
&D_2 \\
&D_3
\end{align*}
\]

Communicate
Background: Trade Even More Consistency for Throughput

\[ \Delta W_1, \Delta W_2, \Delta W_3 \]

Coalesce deltas to reduce communication

\[ \Delta W_1 + \Delta W_2 + \Delta W_3 \]

+ **Local Buffering**: communicate every N mini-batches, coalescing deltas

Compute: \( D_1, D_2, D_3 \)

Communicate: \( C_1 \)

Compute: \( D_4, D_5, D_6 \)

Communicate: \( C_2 \)

+ **Bounded Staleness**: block iff the fastest is T steps ahead of the slowest

Compute: \( D_1, D_2, D_3, D_4, D_5, D_6 \)

Communicate: \( C_1 \)
Background: Trade Even More Consistency for Throughput

\[ \begin{align*}
\Delta W_1 & = +2 \\
\Delta W_2 & = +3 \\
\Delta W_3 & = -1 \\
\Delta W_1 + \Delta W_2 + \Delta W_3 & = +4
\end{align*} \]

Coalesce deltas to reduce communication

+ **Local Buffering**: communicate every N mini-batches, coalescing deltas

Compute:
\[ D_1, D_2, D_3, D_4, D_5, D_6 \]

Communicate:
\[ C_1, C_2 \]

+ **Bounded Staleness**: block if the fastest is T steps ahead of the slowest

Compute:
\[ D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, \ldots \]

Communicate:
\[ C_1, C_2, \ldots \]
Reduce Inconsistency via Scheduling

Network Communication
When and what to send?

Computation
What to compute in parallel?

Publication
[SoCC’15, Best paper] [EuroSys’19]

Systems Developed
Bösen: parameter server
Orion: parallelization framework

Highlights of results:
• Scheduling communication: up to 30% faster convergence
• Scheduling computation: even faster convergence with less programmer effort
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, ...$
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

Compute:  \( D_1 \) \( D_2 \) \( D_3 \) \( D_4 \) \( D_5 \) \( D_6 \) \( D_7 \) \( D_8 \) \( D_9 \) \( \ldots \)

Communicate:  \( C_1 \) \( C_2 \) \( \ldots \)

Idle network
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, ...$

Idle network

Manually tuning communication frequency:

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, C_3, ...$
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, ...$

Idle network

Manually tuning communication frequency:

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, C_3, ...$

$D_3$ updates become available, more effective to communicate coalesced updates
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

Compute

\[ D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, \ldots \]

Communicate

\[ C_1, C_2, C_3, \ldots \]
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, C_3, ...$

Ours: fine-grained communication

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$

Communicate: $C_1, C_2, C_3, ...$
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

Compute: $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, $D_7$, $D_8$, $D_9$, ...

Communicate: $C_1$, $C_2$, $C_3$, ...

Ours: fine-grained communication

Compute: $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, $D_7$, $D_8$, $D_9$, ...

Communicate: $C_3$, $C_5$, ...
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

Compute: $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, $D_7$, $D_8$, $D_9$, ...

Communicate: $C_1$, $C_2$, $C_3$, ...

Ours: fine-grained communication

Compute: $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, $D_7$, $D_8$, $D_9$, ...

Communicate: $C_3$, $C_5$, ...

Periodic synchronization to ensure convergence
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, \ldots$

Communicate: $C_1, C_2, C_3, \ldots$

Ours: fine-grained communication

Periodic synchronization to ensure convergence

Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, \ldots$

Communicate: $C'_1, C'_2, C_3, C_4, C_5, C'_6, \ldots$
Fine-Grained Comm. + Prioritization [Wei et al., SoCC’15]

Existing: manually tuned communication frequency

- Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$
- Communicate: $C_1, C_2, C_3, ...$

Ours: fine-grained communication

- Compute: $D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, ...$
- Communicate: $C'_1, C'_2, C_3, C_4, C_5, C'_6, ...$

Periodic synchronization to ensure convergence

Prioritize update communication based on relative magnitude
# Experiment Results: Time to Convergence

<table>
<thead>
<tr>
<th>1 sync / pass</th>
<th>2 syncs / pass</th>
<th>4 syncs / pass</th>
<th>8 syncs / pass</th>
<th>320Mbps</th>
<th>640Mbps</th>
<th>320Mbps</th>
<th>640Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>PowerGraph [OSDI'12], IterStore [SoCC'14], etc</td>
<td>Sched. comm. w/ random</td>
<td>Sched. comm. w/ relative mag</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bösen on 16 machines, 1 Gbps, topic modeling
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps, topic modeling

synchronize all parameters N times for each pass over local data
Experiment Results: Time to Convergence

- Synchronize all parameters N times for each pass over local data

Bösen on 16 machines, 1 Gbps, topic modeling

Baseline
- PowerGraph [OSDI’12], IterStore [SoCC’14], etc

Sched. comm. w/random
- Shed. comm. w/relative mag

Network Throughput:
- 320Mbps
- 640Mbps
Experiment Results: Time to Convergence

- Synchronize all parameters N times for each pass over local data.

- No improvement from more frequent synchronizations.

Baseline
PowerGraph [OSDI ’12],
IterStore [SoCC ’14], etc

Sched. comm. w/ random
Shed. comm. w/ relative mag

Bösen on 16 machines, 1 Gbps, topic modeling

Data points:
- 1 sync/pass: 2863
- 2 syncs/pass: 1200
- 4 syncs/pass: 811
- 8 syncs/pass: 752

Bandwidths:
- 320 Mbps
- 640 Mbps
Experiment Results: Time to Convergence

- Bösen on 16 machines, 1 Gbps, topic modeling

- Synchronize all parameters $N$ times for each pass over local data

- No improvement from more frequent synchronizations

- Schedule under a bandwidth budget; random prioritization

- Baseline
  - PowerGraph [OSDI'12]
  - IterStore [SoCC'14], etc

- Sched. comm. w/ random
- Shed. comm. w/ relative mag
Experiment Results: Time to Convergence

- Synchronize all parameters $N$ times for each pass over local data.

- No improvement from more frequent synchronizations.

- Schedule under a bandwidth budget; random prioritization.

- Baseline, PowerGraph [OSDI'12], IterStore [SoCC'14], etc.

- Sched. comm. w/ random

- Shed. comm. w/ relative mag

Bösen on 16 machines, 1 Gbps, topic modeling.
Experiment Results: Time to Convergence

- Synchronize all parameters $N$ times for each pass over local data.
- No improvement from more frequent synchronizations.
- Schedule under a bandwidth budget; random prioritization.

- 2420MB per data pass.
- 1715MB per data pass.

Baseline:
- PowerGraph [OSDI’12],
- IterStore [SoCC’14], etc

Sched. comm. w/ random
- 320Mbps
- 640Mbps

Shed. comm. w/ relative mag
- 320Mbps
- 640Mbps
Experiment Results: Time to Convergence

- Bösen on 16 machines, 1 Gbps, topic modeling

Synchronize all parameters N times for each pass over local data:
- No improvement from more frequent synchronizations

- 2420MB per data pass
- 1715MB per data pass

Schedule under a bandwidth budget; prioritize based on relative magnitude of random prioritization.

Baseline
- PowerGraph [OSDI’12], IterStore [SoCC’14], etc

Latency:
- 320MBps: 2343 s
- 640MBps: 1195 s

Sched. comm. w/ random
- 320MBps: 905 s
- 640MBps: 719 s

Shed. comm. w/ relative mag
Experiment Results: Time to Convergence

- Synchronize all parameters $N$ times for each pass over local data
- 2420 MB per data pass
- No improvement from more frequent synchronizations
- 1715 MB per data pass
- 840 MB per data pass

Schedule under a bandwidth budget; prioritize based on relative magnitude

Baseline
PowerGraph [OSDI’12],
IterStore [SoCC’14], etc

Sched. comm. w/ random
Sched. comm. w/ relative mag
Experiment Results: Time to Convergence

- Synchronize all parameters N times for each pass over local data
- 2420MB per data pass
- No improvement from more frequent synchronizations
- 1715MB per data pass
- 840MB per data pass

- Schedule under a bandwidth budget; prioritize based on relative magnitude

- Baseline
  - PowerGraph [OSDI'12]
  - IterStore [SoCC'14], etc

- Sched. comm. w/ random
- Shed. comm. w/ relative mag
Schedule Computation To Reduce Inconsistency

Execute only independent mini-batches in parallel

repeat until convergence
   in parallel foreach mini-batch in dataset
      update model parameters

Servers

Workers
Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:
Model: Matrix Factorization
Application: Recommender systems
Parameters: User Latent Vectors, Item Latent Vectors

Data sample:

<table>
<thead>
<tr>
<th>UserID</th>
<th>ItemID</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>#38</td>
<td>#65</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:

Model: Matrix Factorization
Application: Recommender systems
Parameters: User Latent Vectors, Item Latent Vectors

Data sample:

<table>
<thead>
<tr>
<th>UserID</th>
<th>ItemID</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>#38</td>
<td>#65</td>
<td>5.0</td>
</tr>
</tbody>
</table>

There exist $f_1, f_2, ..., f_k$, such that
if $d_i[f_1] != d_j[f_1], d_i[f_2] != d_j[f_2], ...,$ and $d_i[f_k] != d_j[f_k],
$d_i$ and $d_j$ don’t access the same parameters.
Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:
Model: Matrix Factorization
Application: Recommender systems
Parameters: User Latent Vectors, Item Latent Vectors

Data sample:

<table>
<thead>
<tr>
<th>UserID</th>
<th>ItemID</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>#38</td>
<td>#65</td>
<td>5.0</td>
</tr>
</tbody>
</table>

There exist $f_1, f_2, ..., f_k$, such that
if $d_i[f_1] \neq d_j[f_1], d_i[f_2] \neq d_j[f_2], ..., \text{and } d_i[f_k] \neq d_j[f_k],$

$d_i$ and $d_j$ don’t access the same parameters.

Other examples: topic modeling, gradient boosted trees, etc.
Partition The Dataset for Nonconflicting Accesses

There exist $f_1, f_2, ..., f_k$, such that

if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,

$d_i$ and $d_j$ don’t access the same parameters.

Partition the dataset by those fields
Partition The Dataset for Nonconflicting Accesses

There exist $f_1, f_2, ..., f_k$, such that

if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,

$d_i$ and $d_j$ don’t access the same parameters.

Partition the dataset by those fields
Partition The Dataset for Nonconflicting Accesses

There exist $f_1, f_2, ..., f_k$, such that

if $d_i[f_1] \neq d_i[f_2]$, $d_i[f_2] \neq d_i[f_2]$, ..., and $d_i[f_k] \neq d_i[f_k]$,

$d_i$ and $d_j$ don’t access the same parameters.

Partition the dataset by those fields

Dataset
Partition The Dataset for Nonconflicting Accesses

There exist $f_1, f_2, ..., f_k$, such that

if $d_i[f_1] \neq d_i[f_2]$, $d_i[f_2] \neq d_i[f_3]$, ..., and $d_i[f_k] \neq d_i[f_1]$,

$d_i$ and $d_j$ don’t access the same parameters.

Partition the dataset by those fields

Nonconflicting parameter accesses
Partition The Dataset for Nonconflicting Accesses

There exist $f_1, f_2, ..., f_k$, such that 
if $d_i[f_1] \neq d_j[f_1], d_i[f_2] \neq d_j[f_2], ...,$ and $d_i[f_k] \neq d_j[f_k],$

$d_i$ and $d_j$ don’t access the same parameters.

Partition the dataset by those fields

Nonconflicting parameter accesses

Special case of automatic parallelizing compilers
Challenges for Scheduling Computation
Challenges for Scheduling Computation

Challenge #1: applicable only when certain property holds
Solution: fall back to data parallelism otherwise
Challenges for Scheduling Computation

Challenge #1: applicable only when certain property holds
Solution: fall back to data parallelism otherwise

Challenge #2: implementation requires non-trivial programmer effort
Solution: automatically parallelize serial programs
Orion: Automatic Parallelization [Wei, et al., EuroSys’19]

Our goals:
1. A parallel_for construct and users implement a serial program;
2. Preserves sequential semantics when possible;
3. Data parallelism otherwise with user permission

Orion’s abstraction:
A single thread w/ huge memory

Serial ML program in Julia
Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine
40 Gbps Ethernet
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm
Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine
40 Gbps Ethernet
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm
Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine
40 Gbps Ethernet
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm
Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine
40 Gbps Ethernet
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm
Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine
40 Gbps Ethernet
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm

thousands of lines of C++
vs. a few hundreds of lines of Julia
Compare Orion vs. TensorFlow

1 machines, 32 vCPU cores / machine
Matrix Factorization (MF) for recommendations + SGD

TensorFlow suffers due to
- Data parallelism
- Slower per-sample convergence
- Poor support for sparsity
- 2x longer time per iteration
The Developing View: More And More Complex ML Models

repeat until convergence
  foreach mini-batch in dataset
    update model parameters

We’ve focused improving computation across mini-batches.

**The Machine Learning Trend:**
The mini-batch computation is becoming more and more complex
The Developing View: More And More Complex ML Models

repeat until convergence
foreach mini-batch in dataset
update model parameters

We’ve focused improving computation across mini-batches.

The Machine Learning Trend:
The mini-batch computation is becoming more and more complex

Deep Neural Networks:
Heavy computation per mini-batch
Dense parameter access
Synchronize after each mini-batch
The Developing View: More And More Complex ML Models

repeat until convergence
foreach mini-batch in dataset
update model parameters

We’ve focused improving computation across mini-batches.

The Machine Learning Trend:
The mini-batch computation is becoming more and more complex

Deep Neural Networks:
Heavy computation per mini-batch
Dense parameter access
Synchronize after each mini-batch

Opportunity:
Improve DNN efficiency without sacrificing computation quality
The Mini-Batch Computation of Deep Neural Networks

Forward:
\[ f_1(w_1, x_0) \rightarrow f_2(w_2, x_1) \rightarrow f_3(w_3, x_2) \rightarrow y \]

Backward:
\[ \frac{\partial y}{\partial w_1} \rightarrow \frac{\partial y}{\partial w_2} \rightarrow \frac{\partial y}{\partial w_3} \]
The Mini-Batch Computation of Deep Neural Networks

Forward:
\[ f_1(w_1, x_0) \rightarrow f_2(w_2, x_1) \rightarrow f_3(w_3, x_2) \rightarrow y \]

Backward:
\[ \frac{\partial y}{\partial w_1} \rightarrow \frac{\partial y}{\partial w_2} \rightarrow \frac{\partial y}{\partial w_3} \]

Opportunity:
Not all parameters (updates) are needed (generated) at the same time.
Schedule Communication Within A Mini-Batch for DNNs

Wait-Free Back Propagation: [Zhang et al., ATC’17] (coauthor)
Send updates layer by layer in the backward order, i.e., as soon as they are generated

Overlapping backward computation with communication within a mini-batch
Ideally, computation is idle only during the first layer’s communication
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute

L₁  L₂  L₃  L₃  L₂  L₁

Communicate

L₃  L₂  L₁
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

<table>
<thead>
<tr>
<th>Compute</th>
<th>Forward</th>
<th>Backward</th>
<th>Idle</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₁</td>
<td>L₂</td>
<td>L₃</td>
<td>L₂</td>
<td>L₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₃</td>
</tr>
<tr>
<td>L₂</td>
</tr>
<tr>
<td>L₁</td>
</tr>
</tbody>
</table>

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute

* L_1 * L_2 * L_3 * L_3 * L_2 * L_1

Communicate

* L_3 * L_2 * L_1

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed

Compute

* L_1 * L_2 * L_3
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute
- $L_1$
- $L_2$
- $L_3$
- $L_3$
- $L_2$
- $L_1$

Communicate
- $L_3$
- $L_2$
- $L_1$

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

- But the first layer’s communication could be delayed by the previous layers.

Compute:
- Forward: $L_1$, $L_2$, $L_3$,
- Backward: $L_2$, $L_3$, $L_1$,
- Idle:
  - Forward: $L_1$, $L_2$, $L_3$.

Communicate:
- $L_3$, $L_2$, $L_1$.

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
- Prioritize communication based on when the value is needed.

Compute:
- Forward: $L_1$, $L_2$, $L_3$, $L_3$, $L_2$.
- Backward: $L_{3,1}$.
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers.

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed.
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer's communication could be delayed by the previous layers

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed

Compute
- L₁
- L₂
- L₃
- L₃
- L₂
- L₁
- L₁

Communicate
- L₃
- L₂
- L₁

Compute
- L₁
- L₂
- L₃
- L₃
- L₂
- L₁

Communicate
- L₃,₁
- L₂,₁
- L₁
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

-but the first layer’s communication could be delayed by the previous layers-

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute

<table>
<thead>
<tr>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
<th>L₃</th>
<th>L₂</th>
<th>L₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Backward</td>
<td>Idle</td>
<td>Forward</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Communicate

<table>
<thead>
<tr>
<th>L₃</th>
<th>L₂</th>
<th>L₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Backward</td>
<td></td>
</tr>
</tbody>
</table>

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed

Compute

<table>
<thead>
<tr>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
<th>L₃</th>
<th>L₂</th>
<th>L₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Backward</td>
<td>Idle</td>
<td>Forward</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Communicate

<table>
<thead>
<tr>
<th>L₃,₁</th>
<th>L₂,₁</th>
<th>L₁</th>
<th>L₂,₂</th>
<th>L₃,₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Backward</td>
<td></td>
<td>Forward</td>
<td></td>
</tr>
</tbody>
</table>
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
<th>Idle</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₁</td>
<td>L₂</td>
<td>L₃</td>
<td>L₁</td>
</tr>
<tr>
<td>L₂</td>
<td>L₁</td>
<td></td>
<td>L₁</td>
</tr>
</tbody>
</table>

Communicate

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
<th>Idle</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₃</td>
<td>L₂</td>
<td>L₁</td>
<td></td>
</tr>
</tbody>
</table>

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed
P3 Experiment Results on MXNet

Baseline: MXNet (w/ Wait-Free Backpropagation)
Model: ResNet-50
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers

Compute

<table>
<thead>
<tr>
<th>Compute</th>
<th>Forward</th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
<th>L₃</th>
<th>L₂</th>
<th>L₁</th>
<th>Forward</th>
</tr>
</thead>
</table>

Communicate

<table>
<thead>
<tr>
<th>Communicate</th>
<th>L₃</th>
<th>L₂</th>
<th>L₁</th>
</tr>
</thead>
</table>

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed
Scheduling within a Single Training Job

Network Communication
When and what to send?

Computation
What to compute in parallel?

Memory Allocation
When and where to allocate?

Lead [SoCC’15, Best paper] [EuroSys’19] [In preparation]
Coauthor [ATC’17] [SysML’19]

Highlights of results:
- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- Scheduling memory: 10x bigger model on the same hardware
P3 Experiment Results on MXNet

Throughput (images/sec) vs Bandwidth (Gbps)

Baseline: MXNet (w/ Wait-Free Backpropagation)
Model: ResNet-50
Scheduling within a Single Training Job

- **Network Communication**: When and what to send?
- **Computation**: What to compute in parallel?
- **Memory Allocation**: When and where to allocate?

**Lead** [SoCC’15, Best paper] | **EuroSys’19** | **In preparation**
---|---|---
**Coauthor** [ATC’17] [SysML’19]

**Highlights of results:**
- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- Scheduling memory: 10x bigger model on the same hardware
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)

But the first layer’s communication could be delayed by the previous layers.

Priority-based Parameter Propagation: [Jayarajan et al., SysML’19] (coauthor)
Prioritize communication based on when the value is needed.
Larger Models Lead To Better Performance

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer

Nour Shazeer¹, Azalia Mirhoseini⁴, Krzysztof Maziarczyk³, Andy Davis¹, Quoc Le¹, Geoffrey Hinton⁷ and Jeff Dean¹

138 Billion Params
128 GPUs
GPU Memory Is Limited And Expensive

![GPU Memory Cost Over Time Graph]

- DRAM
- Desktop GPU
- Data Center GPU

$ / MB

Year


K20c
Kepler K40
V100 PCIe
Titan V
Titan Black
GTX 580
Titan X
Titan X2000
1080 Ti
GPU Memory Is Limited And Expensive

- DRAM
- Desktop GPU
- Data Center GPU

Memory Capacity | Price
--- | ---
16GB | $7399
32GB | $8799

Nvidia V100 (PCIe) GPU Price
Source: thinkmate.com
2019/8/12
GPU Memory Is Limited And Expensive

<table>
<thead>
<tr>
<th>Year</th>
<th>Memory Capacity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>16GB</td>
<td>$7399</td>
</tr>
<tr>
<td>2005</td>
<td>32GB</td>
<td>$8799</td>
</tr>
</tbody>
</table>

Nvidia V100 (PCIe) GPU Price
Source: thinkmate.com
2019/8/12

$0.085 per extra MB
Many Previous Works on Improving Memory Efficiency

**Gradient checkpointing** (leveraging recomputation)
- Training Deep Nets with Sublinear Memory Cost [Chen et al., arXiv’16]
- Memory-Efficient Backpropagation Through Time [Gruslys et al., arXiv’16]

**Memory swapping** (leveraging cheaper host memory)
- Dynamic Control Flow in Large-Scale Machine Learning [Yu, EuroSys’19]
- vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design [Rhu et al., MICRO’16]
- Training Deeper Models by GPU Memory Optimization on TensorFlow [Meng et al., MLSys’17]
- Superneurons: dynamic GPU memory management for training deep neural networks [Wang et al., PPoPP’18]
- TensorFlow Grapper memory optimizer
Background: Gradient Checkpointing

Original computation graph for backpropagation, $O(M)$ memory cost

With gradient checkpointing, $O(\sqrt{N})$ memory cost

Recomputed when needed
Background: Memory Swapping

Original computation graph for backpropagation, $O(M)$ memory cost

With memory swapping, $O(1)$ memory cost
They Work Well for Linear Graphs

Most nodes are “graph separator nodes”: removing each one separates the graph into two disjoint subgraphs
Gradient checkpointing: easy to determine which nodes to checkpoint.

Limited freedom regarding scheduling
Memory swapping: easy to determine what and when to swap

Problem: many neural network graphs are not linear!
Some layers have an excessive amount of parallelism.
Emerging Non-linear Neural Networks

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean

Millions of parameters per expert. Experts are sparsely activated.
Goal: General Memory-Efficient DL On TensorFlow

- Linear and nonlinear computation graphs
- Implement and evaluate on TensorFlow
- Transparent to applications.

Existing memory optimizations for TensorFlow:

- Gradient checkpointing (Bolatov et al., GitHub’17):
  Limited to linear graphs; requires non-trivial changes to application program

- Grappler memory swapping pass:
  Limited to linear graphs

- WhileLoop memory swapping ([Yuan et al., EuroSys’18]):
  Operation specific memory reduction
Idea #1: Limit Memory Consumption by Limiting Parallelism

TensorFlow

Breath-first traversal
Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

TensorFlow
Breath-first traversal
Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

TensorFlow

Breath-first traversal
Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 5 operations

TensorFlow
- Breath-first traversal
  - Max. parallelism
  - Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 5 operations

TensorFlow

Breath-first traversal

Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 5 operations

TensorFlow

Breath-first traversal
Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

TensorFlow
Breath-first traversal
Max. parallelism
Max. memory

Peak memory: 5 operations
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 5 operations

TensorFlow

Breath-first traversal
Max. parallelism
Max. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

TensorFlow
- Breath-first traversal: Max. parallelism, Max. memory
- Linearize the graph: No parallelism, Min. memory

Peak memory: 5 operations
Idea #1: Limit Memory Consumption by Limiting Parallelism

- TensorFlow
  - Breath-first traversal
    - Max. parallelism
    - Max. memory
  - Linearize the graph
    - No parallelism
    - Min. memory

Peak memory: 5 operations
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 4 operations

TensorFlow
- Breath-first traversal
  - Max. parallelism
  - Max. memory

Ours
- Partition the graph, Linearize among partitions
  - Parallelism with partitions
  - Control parallelism vs. memory

- Linearize the graph
  - No parallelism
  - Min. memory
Idea #2: Offload GPU Tensors To Host Memory

Transformer
Use MoE as the Feed Forward layer
12 MoEs
32 experts per MoE
2M params per expert
~800M parameters total

Peak memory: 9.5GB to 6.8GB
Idea #2: Offload GPU Tensors To Host Memory

Transformer
Use MoE as the Feed Forward layer
12 MoEs
32 experts per MoE
2M params per expert
~800M parameters total

Peak memory: 9.5GB to 6.8GB
Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed

Transformer w/ MoE Peak memory: 6.8GB to 3.3GB
Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed

Transformer w/ MoE Peak memory:
6.8GB to 3.3GB
Implementation & Experiment Setup

- Application
- TensorFlow API
  - Python
- TensorFlow C++ Core
  - Grappler Optimizers
  - Executor
  - GraphPartition
    - Graph Partition & Memory Swapping
    - Scheduling
    - Send, Recv nodes

Experiment platform:
- 32 vCPU cores
- 64GB memory
- 1 GPU per machine
- Nvidia TitanX Maxwell
- 12GB GPU Memory
Experiment Results

- Transformer
- Transformer + MoE
- ResNet-152
- WGAN-GP
- DeepSpeech
- Avg
- Avg-NoMoE

<table>
<thead>
<tr>
<th>Model</th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
<th>Recurrent / Statically unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>4.9</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>Transformer + MoE</td>
<td>9.5</td>
<td>4.2</td>
<td>6.8</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>ResNet-152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WGAN-GP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepSpeech</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Vanilla
- + Partition
- + Placement
Experiment Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
<th>Recurrent / Statically unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>9.5</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>4.9</td>
<td>4.2</td>
<td>3.3</td>
<td>6.7</td>
<td>6.4</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>4.2</td>
<td>3.8</td>
<td>1.6</td>
<td>1.4</td>
<td>4.4</td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td>4.8</td>
<td>3.8</td>
<td>4.4</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>6.7</td>
<td>6.4</td>
<td>4.8</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>4.4</td>
<td>4.0</td>
<td>4.4</td>
<td>4.0</td>
<td></td>
</tr>
</tbody>
</table>

- **Vanilla**, **+Partition**, **+Placement**
Experiment Results

- Transformer
- TransformerMoE
- ResNet-152
- WGAN-GP Model
- DeepSpeech
- Avg
- Avg-NoMoE

- Attention
- Attention + MoE
- Convolution
- GAN
- Recurrent / Statically unrolled

800 Million parameters

Peak Memory (GB)

Transformer: 10.9, 4.9, 4.2, 6.8, 3.3, 4.2, 1.4, 2.8, 4.8, 4.4, 4.4, 4.0
TransformerMoE: 9.5, 4.2, 3.3, 6.8, 3.3, 4.2, 1.4, 2.8, 4.8, 4.4, 4.0
ResNet-152: 11.0, 11.0, 11.0, 11.0
WGAN-GP Model: 6.7, 6.4, 6.7, 6.4
DeepSpeech: 4.8, 3.8, 4.4, 3.8
Avg: 11.0, 11.0, 11.0, 11.0
Avg-NoMoE: 11.0, 11.0, 11.0, 11.0

Vanilla + Partition + Placement
Experiment Results

- **Attention**
- **Attention + MoE**
- **Convolution**
- **GAN**
- **Recurrent / Statically unrolled**

800 Million parameters

Peak Memory (GB)

<table>
<thead>
<tr>
<th>Model</th>
<th>Vanilla + Partition + Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td></td>
</tr>
<tr>
<td>TransformerMoE</td>
<td></td>
</tr>
<tr>
<td>ResNet-152</td>
<td></td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td></td>
</tr>
<tr>
<td>DeepSpeech</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td></td>
</tr>
</tbody>
</table>

Values:
- Transformer: 4.9, 4.2
- TransformerMoE: 4.2, 3.8
- ResNet-152: 6.8, 6.4
- WGAN-GP Model: 3.3, 3.8
- DeepSpeech: 1.6, 1.4
- Avg: 4.8, 4.0
- Avg-NoMoE: 4.4, 4.0
Experiment Results

- Attention
- Attention + MoE (Convolution)
- GAN
- Recurrent / Statically unrolled

800 Million parameters

<table>
<thead>
<tr>
<th>Model</th>
<th>Vanilla</th>
<th>Partition</th>
<th>Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>4.9</td>
<td>4.2</td>
</tr>
<tr>
<td>Transformer + MoE</td>
<td>9.5</td>
<td>6.8</td>
<td>3.3</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>11.0</td>
<td>4.2</td>
<td>3.8</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>11.0</td>
<td>1.6</td>
<td>1.4</td>
</tr>
<tr>
<td>Model</td>
<td>11.0</td>
<td>6.7</td>
<td>6.4</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>10.8</td>
<td>4.8</td>
<td>4.0</td>
</tr>
<tr>
<td>Avg</td>
<td>11.0</td>
<td>4.4</td>
<td>4.0</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peak Memory (GB)
Experiment Results

800 Million parameters

<table>
<thead>
<tr>
<th>Method</th>
<th>Peak Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attention</td>
<td>10.9</td>
</tr>
<tr>
<td>Attention + MoE</td>
<td>4.9</td>
</tr>
<tr>
<td>Convolution</td>
<td>4.2</td>
</tr>
<tr>
<td>GAN</td>
<td>6.8</td>
</tr>
<tr>
<td>Recurrent /</td>
<td>11.0</td>
</tr>
<tr>
<td>Statically unrolled</td>
<td>11.0</td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td>1.6</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>4.8</td>
</tr>
<tr>
<td>Avg</td>
<td>6.4</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Legend:
- Vanilla
- Partition
- Placement
Experiment Results

- Attention
- Attention + MoE
- Convolution
- GAN
- Recurrent / Statically unrolled
- Average

Peak Memory (GB)

- Transformer: 10.9
- Transformer+MoE: 4.9
- ResNet-152: 6.6
- WGAN-GP Model: 11.0
- DeepSpeech: 1.6
- Avg: 10.8
- Avg-NoMoe: 4.4

800 Million parameters
Experiment Results

- Attention
- Attention + MoE
- Convolution
- GAN
- Recurrent / Statically unrolled
- Average

Peak Memory (GB):
- Transformer: 10.9
- Transformer MoE: 4.9
- ResNet-152: 6.8
- WGAN-GP: 11.0
- DeepSpeech: 4.2
- Avg: 11.0
- Avg-NoMoE: 11.0

Runtime Overhead (w.r.t. TensorFlow):
- Transformer: 1.6
- Transformer MoE: 2.4
- ResNet-152: 2.6
- WGAN-GP: 1.8
- DeepSpeech: 2.2
- Avg: 1.3
- Avg-NoMoE: 1.85

Note: 800 Million parameters
**Experiment Results**

- **800 Million parameters**

### Peak Memory (GB)

<table>
<thead>
<tr>
<th>Method</th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
<th>Recurrent / Statically unrolled</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>9.5</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>4.9</td>
<td>6.8</td>
<td>4.2</td>
<td>4.7</td>
<td>3.8</td>
<td>4.2</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>11.0</td>
<td>4.2</td>
<td>3.8</td>
<td>6.7</td>
<td>6.4</td>
<td>6.1</td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>1.4</td>
<td>1.4</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Avg</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>4.4</td>
<td>4.4</td>
<td>4.4</td>
<td>4.4</td>
<td>4.4</td>
<td>4.4</td>
</tr>
</tbody>
</table>

### Runtime Overhead (w.r.t. TensorFlow)

<table>
<thead>
<tr>
<th>Method</th>
<th>Transformer</th>
<th>TransformerMoE</th>
<th>ResNet-152</th>
<th>WGAN-GP</th>
<th>DeepSpeech</th>
<th>Avg</th>
<th>Avg-NoMoE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.8</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>2.4</td>
<td>2.6</td>
<td>2.6</td>
<td>2.2</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>1.8</td>
<td>1.5</td>
<td>1.5</td>
<td>1.3</td>
<td>1.3</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>1.8</td>
<td>1.5</td>
<td>1.5</td>
<td>1.3</td>
<td>1.3</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>1.5</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>Avg</td>
<td>2.2</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>2.4</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>

### Notes

- **Vanilla** + Partition + Placement
- **Vanilla** - Partition - Placement
Experiment Results

800 Million parameters

<table>
<thead>
<tr>
<th>Method</th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
<th>Recurrent /</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>9.5</td>
<td>6.8</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>4.9</td>
<td>4.2</td>
<td>4.2</td>
<td>4.2</td>
<td>1.6</td>
<td>1.4</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>3.3</td>
<td>3.3</td>
<td>3.8</td>
<td>3.8</td>
<td>3.8</td>
<td>3.8</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>6.7</td>
<td>6.4</td>
<td>6.7</td>
<td>6.4</td>
<td>6.4</td>
<td>6.4</td>
</tr>
<tr>
<td>Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
<td>4.8</td>
</tr>
<tr>
<td>Avg</td>
<td>4.4</td>
<td>4.0</td>
<td>4.4</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experiment Results

- Transformer
- TransformerMoE
- ResNet-152
- WGAN-GP
- DeepSpeech
- Avg
- Avg-NoMoE

Peak Memory (GB)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
<th>Recurrent / Statically unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>4.9</td>
<td>9.5</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>4.2</td>
<td>6.8</td>
<td>4.2</td>
<td>3.8</td>
<td>6.7</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>3.3</td>
<td>4.2</td>
<td>3.8</td>
<td>1.6</td>
<td>4.8</td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td>1.4</td>
<td>6.4</td>
<td>4.8</td>
<td>3.8</td>
<td>4.4</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>1.4</td>
<td>11.0</td>
<td>11.0</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>3.8</td>
<td></td>
<td>11.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

800 Million parameters

Vanilla + Partition + Placement
Experiment Results

<table>
<thead>
<tr>
<th></th>
<th>Attention</th>
<th>Attention + MoE</th>
<th>Convolution</th>
<th>GAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>4.9</td>
<td>6.8</td>
<td>11.0</td>
</tr>
<tr>
<td>Transformer+MoE</td>
<td>4.2</td>
<td>9.5</td>
<td>3.3</td>
<td>11.0</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>4.2</td>
<td>4.2</td>
<td>3.8</td>
<td>11.0</td>
</tr>
<tr>
<td>WGAN-GP Model</td>
<td>1.6</td>
<td>1.4</td>
<td>6.7</td>
<td>10.8</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>4.8</td>
<td>3.8</td>
<td>6.4</td>
<td>11.0</td>
</tr>
<tr>
<td>Avg</td>
<td>4.4</td>
<td>4.4</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Vanilla
- +Partition
- +Placement

Peak Memory (GB)
Experiment Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Attention</th>
<th>Attention + MoE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>4.9</td>
</tr>
<tr>
<td>TransformerMoE</td>
<td>9.5</td>
<td>4.2</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>6.8</td>
<td>3.3</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>11.0</td>
<td>4.2</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>11.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Avg</td>
<td>10.8</td>
<td>4.8</td>
</tr>
<tr>
<td>Avg-NoMoE</td>
<td>11.0</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Peak Memory (GB)
Implementation & Experiment Setup

Experiment platform:
- 32 vCPU cores
- 64GB memory
- 1 GPU per machine

Nvidia TitanX Maxwell
- 12GB GPU Memory

---

TensorFlow API

Application

Python

---

TensorFlow C++ Core

Grappler Optimizers
Graph Partition & Memory Swapping

Executor
Scheduling

GraphPartition
Send, Recv nodes
Idea #1: Limit Memory Consumption by Limiting Parallelism

- Peak memory: 4 operations
- TensorFlow: Breath-first traversal
  - Max. parallelism
  - Max. memory
- Linearize the graph
  - No parallelism
  - Min. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 4 operations

TensorFlow

Breath-first traversal
Max. parallelism
Max. memory

Linearize the graph
No parallelism
Min. memory
Idea #1: Limit Memory Consumption by Limiting Parallelism

Peak memory: 4 operations

TensorFlow
- Breath-first traversal
  - Max. parallelism
  - Max. memory
- Partition the graph, Linearize among partitions
- Parallelism with partitions
- Control parallelism vs. memory

Ours
- Linearize the graph
  - No parallelism
  - Min. memory
Idea #2: Offload GPU Tensors To Host Memory

Transformer
Use MoE as the Feed Forward layer
12 MoEs
32 experts per MoE
2M params per expert
~800M parameters total

Peak memory: 9.5GB to 6.8GB
Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed

Transformer w/ MoE Peak memory: 6.8GB to 3.3GB
Implementation & Experiment Setup

- Application
- TensorFlow API
  - Python
- TensorFlow C++ Core
  - Grappler Optimizers
  - Executor
  - GraphPartition

Experiment platform:
- 32 vCPU cores
- 64GB memory
- 1 GPU per machine
- Nvidia TitanX Maxwell
- 12GB GPU Memory

Graph Partition & Memory Swapping
- Scheduling
- Send, Recv nodes
Experiment Results

- **Attention**: 10.9 GB, 4.9 GB, 4.2 GB
- **Attention + MoE**: 9.5 GB, 6.8 GB, 3.3 GB
- **Convolution**: 11.0 GB, 4.2 GB, 3.8 GB
- **GAN**: 11.0 GB, 1.6 GB, 1.4 GB
- **Recurrent / Statically unrolled**: 11.0 GB, 6.7 GB, 4.8 GB
- **Average**: 11.0 GB, 4.4 GB, 3.8 GB

- **Runtime Overhead (wrt. TensorFlow)**
  - **Transformer**: 1.6x, 2.4x, 2.6x
  - **TransformerMoE**: 1.6x, 2.6x, 2.6x
  - **ResNet-152**: 1.8x, 2.2x, 2.2x
  - **WGAN-GP Model**: 1.8x, 2.2x, 2.2x
  - **DeepSpeech**: 1.8x, 2.2x, 2.2x
  - **Avg**: 1.8x, 2.2x, 2.2x
  - **Avg-NoMoE**: 1.5x, 1.85x, 1.85x

- **Vanilla + Partition + Placement**

- **800 Million parameters**
Experiment Results

- Attention
- Attention + MoE
- Convolution
- GAN
- Recurrent / Statically unrolled
- Average

**Peak Memory (GB)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Average</th>
<th>Vanilla + Partition + Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>10.9</td>
<td>10.8</td>
</tr>
<tr>
<td>Transformer MoE</td>
<td>9.5</td>
<td>9.4</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>11.0</td>
<td>11.0</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>11.0</strong></td>
<td><strong>11.0</strong></td>
</tr>
<tr>
<td><strong>Average-NoMoE</strong></td>
<td><strong>10.8</strong></td>
<td><strong>10.8</strong></td>
</tr>
</tbody>
</table>

**Runtime Overhead (w.r.t. TensorFlow)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Average</th>
<th>Vanilla + Partition + Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Transformer MoE</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>2.6</td>
<td>2.6</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>DeepSpeech</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>2.2</strong></td>
<td><strong>2.2</strong></td>
</tr>
<tr>
<td><strong>Average-NoMoE</strong></td>
<td><strong>1.85</strong></td>
<td><strong>1.85</strong></td>
</tr>
</tbody>
</table>

800 Million parameters
Experiment Results

- 800 Million parameters

**Peak Memory (GB)**
- Attention: 10.9
- Attention + MoE: 4.9
- Convolution: 6.8
- GAN: 11.0
- Recurrent / Statically unrolled: 11.0
- Average: 10.8
- Average-NoMoE: 11.0

**Runtime Overhead (w.r.t. TensorFlow)**
- Transformer: 1.6
- TransformerMoE: 2.4
- ResNet-152: 3.4
- WGAN-GP Model: 1.8
- DeepSpeech: 2.2
- Avg: 1.8
- Average-NoMoE: 1.85
Experiment Results

800 Million parameters

Peak Memory (GB)

- Attention
- Attention + MoE
- Convolution
- GAN
- Recurrent / Statically unrolled
- Average
- Average-NoMoE

Runtime Overhead (s/TF)

- Transformer
- TransformerMoE
- ResNet-152
- WGANGP
- DeepSpeech
- Avg
- Avg-NoMoE

Legend:
- Vanilla
- +Partition
- +Placement
## Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert
## Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert

Maximum ResNet Depth
### Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE

12 MoEs, 4M parameters per expert

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
<tr>
<td>TensorFlowMem + Optimized MoE</td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Maximum ResNet Depth
Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
<tr>
<td>Batch</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

Maximum ResNet Depth

Distributed Transformer w/ MoE
12 MoEs, 2M parameters per expert
## Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
<tr>
<td><strong>TensorFlowMem + Optimized MoE</strong></td>
<td><strong>4</strong></td>
<td><strong>512</strong></td>
<td><strong>12 Billion</strong></td>
</tr>
</tbody>
</table>

Maximum ResNet Depth

Distributed Transformer w/ MoE
12 MoEs, 2M parameters per expert
### Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE

12 MoEs, 4M parameters per expert

### Maximum ResNet Depth

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
</tbody>
</table>

| TensorFlowMem + Optimized MoE | 4    | 512  | 12 Billion |

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Partition big tensors
## Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

**Transformer w/ MoE**
12 MoEs, 4M parameters per expert

**Maximum ResNet Depth**

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td><strong>TensorFlowMem</strong></td>
<td><strong>4</strong></td>
<td><strong>256</strong></td>
<td><strong>6 Billion</strong></td>
</tr>
<tr>
<td>TensorFlowMem + Optimized MoE</td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

**Distributed Transformer w/ MoE**
12 MoEs, 2M parameters per expert
## Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
<tr>
<td>Batch TensorFlow</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>Batch TensorFlowMem</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
<tr>
<td>TensorFlowMem + Optimized MoE</td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

Maximum ResNet Depth

Distributed Transformer w/ MoE
12 MoEs, 2M parameters per expert

Partition big tensors
### Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
<th>Batch</th>
<th>TensorFlow</th>
<th>TensorFlowMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

Transformer w/ MoE
12 MoEs, 4M parameters per expert

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td><strong>TensorFlowMem</strong></td>
<td><strong>4</strong></td>
<td><strong>256</strong></td>
<td><strong>6 Billion</strong></td>
</tr>
<tr>
<td>TensorFlowMem + Optimized MoE</td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

Maximum ResNet Depth

Distributed Transformer w/ MoE
12 MoEs, 2M parameters per expert
### Scaling Model Size

<table>
<thead>
<tr>
<th>System</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>0.24 Billion</td>
<td>19.0</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>48</td>
<td>2.5 Billion</td>
<td>1.9</td>
</tr>
<tr>
<td>Batch</td>
<td>16</td>
<td>504</td>
<td>1916</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>32</td>
<td>235</td>
<td>1001</td>
</tr>
</tbody>
</table>

**Transformer w/ MoE**

12 MoEs, 4M parameters per expert

**Maximum ResNet Depth**

<table>
<thead>
<tr>
<th>System</th>
<th>#Machines</th>
<th>#Experts / MoE</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>4</td>
<td>128</td>
<td>3 Billion</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>16</td>
<td>208</td>
<td>5 Billion</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>4</td>
<td>256</td>
<td>6 Billion</td>
</tr>
<tr>
<td><strong>TensorFlowMem + Optimized MoE</strong></td>
<td>4</td>
<td>512</td>
<td>12 Billion</td>
</tr>
</tbody>
</table>

**Distributed Transformer w/ MoE**

12 MoEs, 2M parameters per expert

Partition big tensors
Recurrent Neural Networks – Scaling Sequence Length

<table>
<thead>
<tr>
<th>Sequence Length</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>500</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>1.15</td>
<td>2.31</td>
<td>4.65</td>
<td>OOM</td>
<td>OOM</td>
</tr>
<tr>
<td>TensorFlowMem</td>
<td>1.56</td>
<td>3.03</td>
<td>6.03</td>
<td>N/A</td>
<td>12.01</td>
</tr>
</tbody>
</table>

Mozilla DeepSpeech, statically unrolled RNN
Mini-batch size = 128
Time per mini-batch (seconds)
Summary
Summary

Fine-grained communication
Prioritization based on relative magnitude
Prioritization based on when values are used
Summary

- Fine-grained communication
- Prioritization based on relative magnitude
- Prioritization based on when values are used

- Statically analyze memory accesses
- Schedule independent computation in parallel
Summary

Fine-grained communication
Prioritization based on relative magnitude
Prioritization based on when values are used

Statically analyze memory accesses
Schedule independent computation in parallel

Partitioned computation graph
Leverage cheap host memory
Machine Learning Is Still Fast Advancing
Machine Learning Is Still Fast Advancing
Machine Learning Is Still Fast Advancing

ML Models / Algorithms

CNNs, RNNs, residual, MoE, capsule, etc...
Machine Learning Is Still Fast Advancing

ML Models / Algorithms
- CNNs, RNNs, residual, MoE, capsule, etc...

Hardware
- CPU, GPU, FPGA, ASICs, etc
Machine Learning Is Still Fast Advancing

ML Models / Algorithms
- CNNs, RNNs, residual, MoE, capsule, etc...

Systems for ML
- pushing the boundaries of many CS disciplines

Hardware
- CPU, GPU, FPGA, ASICs, etc
Machine Learning Is Still Fast Advancing

ML Models / Algorithms
- CNNs, RNNs, residual, MoE, capsule, etc...

Systems for ML
- pushing the boundaries of many CS disciplines

Hardware
- CPU, GPU, FPGA, ASICs, etc

Compilers

Architecture

Distributed systems

HPC

Networking
Machine Learning Is Still Fast Advancing

ML Models / Algorithms
- CNNs, RNNs, residual, MoE, capsule, etc...

Systems for ML pushing the boundaries of many CS disciplines

Hardware
- CPU, GPU, FPGA, ASICs, etc

Compilers

Architecture

Distributed systems

HPC

Networking

How to support the expanding ML computation?

How to take advantage of new hardware?
Future Directions
Future Directions

Programming support and compilation
Future Directions

Programming support and compilation

- New operations, e.g., capsule?
Future Directions

Programming support and compilation

- New operations, e.g., capsule?
Future Directions

Programming support and compilation
• New operations, e.g., capsule?
• New control flow primitives, e.g., functions?
Future Directions

Programming support and compilation
- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?
- New hardware, e.g., ASICs
Future Directions

Programming support and compilation
• New operations, e.g., capsule?
• New control flow primitives, e.g., functions?
• New hardware, e.g., ASICs

Model parallelism
• Operation partitioning
• Device placement, even dynamic placement for dynamic control flow
Future Directions

Programming support and compilation
- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?
- New hardware, e.g., ASICs

Model parallelism
- Operation partitioning
- Device placement, even dynamic placement for dynamic control flow

ML-driven optimizations for ML systems
- Complex design space
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs
Example: Fast & Memory-Efficient Deep Learning

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many ways to reduce memory consumption, with different trade-offs</td>
<td></td>
</tr>
</tbody>
</table>
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
</tbody>
</table>
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
<tr>
<td>Gradient checkpointing &amp; <strong>Constant folding</strong></td>
<td>Computation</td>
</tr>
</tbody>
</table>
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
<tr>
<td>Gradient checkpointing &amp; <strong>Constant folding</strong></td>
<td>Computation</td>
</tr>
<tr>
<td>Memory swapping &amp; Device placement</td>
<td>Communication</td>
</tr>
</tbody>
</table>
Example: Fast & Memory-Efficient Deep Learning

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
<tr>
<td>Gradient checkpointing &amp; Constant folding</td>
<td>Computation</td>
</tr>
<tr>
<td>Memory swapping &amp; Device placement</td>
<td>Communication</td>
</tr>
<tr>
<td>Quantization</td>
<td>Accuracy</td>
</tr>
</tbody>
</table>
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
<tr>
<td>Gradient checkpointing &amp; <strong>Constant folding</strong></td>
<td>Computation</td>
</tr>
<tr>
<td>Memory swapping &amp; Device placement</td>
<td>Communication</td>
</tr>
<tr>
<td>Quantization</td>
<td>Accuracy</td>
</tr>
</tbody>
</table>

Challenges:
1) Scheduling is NP-complete;
2) Best configuration depends on the program and hardware;
3) Techniques are inter-dependent
Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Trade-off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>Degree of Parallelism</td>
</tr>
<tr>
<td>Gradient checkpointing &amp; Constant folding</td>
<td>Computation</td>
</tr>
<tr>
<td>Memory swapping &amp; Device placement</td>
<td>Communication</td>
</tr>
<tr>
<td>Quantization</td>
<td>Accuracy</td>
</tr>
</tbody>
</table>

Challenges:
1) Scheduling is NP-complete;
2) Best configuration depends on the program and hardware;
3) Techniques are inter-dependent

Minimize training time subject to memory constraints?