S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters (OSU-Caffe)

Horovod: fast and easy distributed deep learning in TensorFlow
S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters (OSU-Caffe)

https://dl.acm.org/citation.cfm?id=3018769
Background

DL Resurgence:

1. Public availability of versatile training data sets: ImageNet, CIFAR...

2. Affordability of modern high-performance and data-parallel hardware: GPU, Accelerator…
Compute Unified Device Architecture (CUDA):
Scale up efficiency within single address space using threads to utilize multiple GPUs in a single node.

Issue:
Saturation of performance of single-node multi-GPU training

Jeffrey Dean:
*Training time is a key challenge at the root of the development of new DNN architectures.*

Distributed Address Space across multi-GPUs and multi-Nodes
CUDA-Aware

Transparencyly provide support for GPU-based data transfers within and across cluster nodes

MPI Implementations with CUDA-Aware support: MVAPICH2, OpenMPI, CrayMPI...

CUDA: Used to offload computation to the GPU device
MPI: responsible for inter-processes communication

Issues:

Most current DL frameworks have not been designed with such CUDA-Aware (MPI + CUDA) techniques
S-Caffe

Efficient scale-up and scale-out for distributed training on accelerated HPC systems

Co-design with MPI runtimes

1. Distributed-memory space efficiency
2. CUDA-Awareness to S-Caffe

Redesigned and Enhanced MPI protocols
Satisfy new requirements pertaining movement of very large messages in DL framework
<table>
<thead>
<tr>
<th>Deep Learning Frameworks</th>
<th>Distributed Address Space Systems</th>
<th>Framework Features and Platforms Supported</th>
<th>Parallelization Strategy / Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic MPI Support</td>
<td>CUDA-Aware MPI</td>
<td>Overlapped Designs (NBC Support)</td>
</tr>
<tr>
<td>Caffe [33]</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>FireCaffe [30]</td>
<td>✓</td>
<td>Unknown</td>
<td>×</td>
</tr>
<tr>
<td>MPI-Caffe [37]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CNTK [12]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Inspur-Caffe [31]</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>S-Caffe (proposed)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1. Design and Features Space for Modern Deep Learning Frameworks
**Figure 1.** Caffe Architecture: Data Propagation, Forward/Backward Computation, and Gradient Aggregation
Requirements for Distributed Address-Space Design & Parallel Data Reading

Current Caffe:

1. Single address space system limited to only work for intra-node scale-up system

2. Single reader sharing the data with different solvers using shared queue

Need for a Distributed Address-Space design to work across nodes, thus more GPUs

Need for efficient parallel data reading mechanism which take advantage of parallel file systems (PFS)
Overlap of Computation and Communication

Current Caffe:

Separation of computation and communication limits system efficiency and scalability

Necessary redesign of workflow to enable overlap between different phases

Figure 1. Caffe Architecture: Data Propagation, Forward/Backward Computation, and Gradient Aggregation
Designing DL-Aware Communication Runtimes

Current MPI protocols and runtimes:
1. Optimized for message size up to 4MB
2. Able to use CPU to perform small reductions on 16~64 byte buffer

Need to handle DL frameworks’ extensively large message size (up to 256 MB)

Need to involve GPU systems to perform large reductions on 256 MB buffer
Proposing S-Caffe:

1. Basic CUDA-Aware MPI Design (SC-B)

2. Multi-stage Non-blocking Collectives for Maximal Overlap (SC-OB)

3. Co-design Gradient Aggregation for Maximal Overlap and Performance: Helper control thread (SC-OBR)
S-Caffe Basic

Explicit Data Movement Operations:

Identify minimal data exchange between different C++ objects

Using GPU-based communication buffers to communicate across processes in a CUDA-Aware fashion

Avoids unnecessary copies between CPU and GPUs
Parallel Readers

Design choice:

Parallel data reader threads for each process

Separate distributed queues

Benefits:

1. Optimized for Parallel file systems (PFS)

2. ImageDataLayer of Caffe allows direct read from storage & arbitrary number of processes

Outcome:

Achieves scalability up to 160 GPUs with ImageDataLayer
Collective Operation Patterns for Data Propagation and Aggregation

Data Propagation Phase:

MPI broadcast primitive (MPI_Bcast) with CUDA Aware

Gradient Aggregation Phase:

MPI_Reduce operation

Efficiency for small scale and small data sets: MNIST, CIFAR

Co-design needed to push scalability and efficiency for large DL models
Multi-stage Non-blocking Collectives for Maximal Overlap (SC-OB)

Challenges

Communication overhead when increasing scale
Strictly marked phase limits efficiency

Proposal

On-demand fashion in communicating model parameters and the gradient data

Computation and communication overlap with Non-Blocking Collective (NBC)
Data Propagation overlaps with Forward Pass

MPI_Ibcast() & MPI_Wait()

Calling strategy for MPI_Wait()

**Figure 4.** Naive NBC Design for Data Propagation

**Figure 5.** Overlapped Data Propagation with Forward
Challenges for Gradient Aggregation overlap with Backward Pass

No efficient NBC reduction primitives provided by MPI

1. CPU is required to progress and perform the computation

2. No overlap when trying to split blocking reduce operation
   • $N$ medium sized reductions = 1 big reduction
   • Need to wait for GPU based backward layer completion before reduce

Proposal:

1. A helper thread to separate kernel completion and wait for communication progress

2. Novel DL-Aware MPI Reduce design exploiting multi-level communicators and kernel-based computation
Helper control thread: splitting up communication and computation phases

Figure 6. Overlapped Gradient Aggregation with Backward
Drive force for Efficient DL-Aware Hierarchical Reduce (HR)

Modern MPI runtimes:

Collective communication and reduction primitives with intra-node and inter-node

Not sufficient for GPU-based communication and large-scale reductions.

Need for a hierarchical mechanism for MPI communicators so that lower level communicators can span multiple nodes
**Chain-size:**

**The number of GPUs in the communicator**

**Chunked Chain Algorithm vs Binomial Tree Algorithm**

\[ T(Bin) = \log(P) \times t(b) \quad \ldots \quad (1) \]
\[ T(CC') = (P - 1) \times t(c) + (n - 1) \times t(c) \]
\[ T(CC') = (n + P - 2) \times t(c) \quad \ldots \quad (2) \]

for large \( P \) and small \( b \), \( T(CC') > > T(Bin) \)

for small \( P \) and large \( b \), \( T(CC') < < T(Bin) \)

Hybrid approaches needed for final design to be skew-tolerant and size-tolerant

---

**Figure 7. Hierarchical DL-Aware Reduction Design with a Chain-Binomial Combination**

Let,

- \( P \) = number of processes
- \( n \) = number of chunks
- \( t(b) \) = time to reduce buffer of size \( b \)
- \( t(c) \) = time to reduce buffer of size \( c \) \( [c = b/n] \)

And,

- \( T(CC') \) = Total Time for Chunked Chain Algorithm
- \( T(Bin) \) = Total Time for Binomial Tree Algorithm
Result from experiments

Ideal parameter value for best chunked chain performance:
1. buffer size > 8M
2. number of processes = 8

The performance began to decrease beyond 8 processes

Choices depending on system size:
1. Two-level chains scaling up to a process count of 64 (chain-of-chain)
2. Upper level using a binomial tree reduction, lower level using a chunked-chain for process count > 64 (chain-binomial)

Figure 7. Hierarchical DL-Aware Reduction Design with a Chain-Binomial Combination
Performance Evaluation

Computational Testbed and Environment

1. Cluster A: KESCH at Swiss National Supercomputing Center
   1. 12 hybrid nodes each containing 8 NVIDIA K-80 GK210GL GPU devices
   2. A total of 192 GPUs and 24 conventional CPUs
   3. 16 CUDA devices per node

2. Cluster B:
   1. 20 nodes with one K-80 GPU per node
   2. 40 GPUs utilization

Evaluation Metrics and Data Sets

1. Training time – parallel training time trend for a fixed number of iterations
2. GoogLeNet & AlexNet on ImageNet (ILSVRC 2012) & CIFAR 10 for comparison
3. Strong scaling results
4. Weak scaling results can be obtained using “–scal weak” command
S-Caffe primary benefits for GoogLeNet model

100 Iterations

3.3x over 16 GPUs with 128 GPUs

2.5x over 32 GPUs with 160 GPUs

**Figure 8.** GoogLeNet: Comparison of S-Caffe (up to 160 GPUs) and Caffe (up to 16 GPUs) on Cluster-A
S-Caffe primary benefits for CIFAR10 Quick Solver

Caffe: 1 node up to 16 GPUs

S-Caffe: 64 GPUs across 4 nodes

1000 iterations

8192 batch-size

32x over single GPU

Figure 9. CIFAR10: Comparison of S-Caffe (up to 64 GPUs) and Caffe (up to 16 GPUs) on Cluster-A
Performance Comparison of S-Caffe, CNTK and Inspur-Caffe

Figure 10. AlexNet: Comparison of S-Caffe, CNTK, and Inspur-Caffe (up to 16 GPUs) on Cluster-B
Performance of Hierarchical Reductions

**Figure 11.** Performance for 160 Processes (GPUs): MVA-PICH2, Chain-Binomial, Chain-Chain, and Proposed HR (Tuned) on Cluster-A
Performance of Hierarchical Reductions

Figure 12. Performance Comparison: MVAPICH2, OpenMPI, and Proposed on Cluster-A

3x faster than MVAPICH2
133x faster than OpenMPI
Impact of various S-Caffe Co-Designs
Overlapped Data Propagation (SC-OB)

**Figure 13.** Comparison of SC-B with SC-OB

15% improvement for SC-OB
Impact of various S-Caffe Co-Designs
Overlapped Gradient Aggregation (SC-OBR) and HR

<table>
<thead>
<tr>
<th>Algorithm / Communicator</th>
<th>SC-B SC-B (+HR)</th>
<th>Aggregation Time</th>
<th>Total Time</th>
<th>Speedup for Aggregation</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>SC-B</td>
<td>40.6</td>
<td>113.6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CC-8</td>
<td>SC-B (+HR)</td>
<td>28.6</td>
<td>101.6</td>
<td>1.47</td>
<td>1.11</td>
</tr>
<tr>
<td>CB-4</td>
<td>SC-B (+HR)</td>
<td>19.8</td>
<td>92.8</td>
<td>2.04</td>
<td>1.22</td>
</tr>
<tr>
<td>CB-8</td>
<td>SC-B (+HR)</td>
<td>17.6</td>
<td>90.6</td>
<td>2.3</td>
<td>1.25</td>
</tr>
</tbody>
</table>

**Table 2.** Comparison of SC-B vs. SC-B with HR

2.3x speedup with CB-8
Horovod: fast and easy distributed deep learning in TensorFlow

https://arxiv.org/abs/1802.05799
Motivations for Horovod

Scaling computation from one GPU to many can enable much faster training

But with two complications:

1. Need efficient inter-GPU communication with little overhead
2. Need little modification to training code to invoke

With Horovod:

1. Employs efficient inter-GPU communication via ring reduction
2. Only a few lines of modification to enable distributed experience
Background

Training using standard distributed TensorFlow technique

1. Various new concepts influx with distributed TensorFlow package

2. Subtle, hard-to-diagnose bugs were introduced which require user to climb a steep learning curve to fix

3. Unable to efficiently utilize hardware resource
Benchmarking standard TensorFlow on multiple GPUs

128 NVIDIA Pascal GPUs
Nearly half of the GPU resources are lost
The Rise of Data Parallelism

1. Read Data
2. Compute Model Updates (Gradients)
3. Average Gradients
4. Update Model
Data-Parallel Distributed Training Paradigm

1. Run multiple copies of the training script and each copy:
   1. Reads a chunk of the data
   2. Runs it through the model
   3. Computes model updates (gradients)

2. Average gradients among those multiple copies (parameter server approach by TensorFlow)

3. Update the model

4. Repeat
Identifying the right ratio of worker to parameter servers
one server: networking or computational bottleneck
multiple server: network interconnects saturated

Handling increased TensorFlow program complexity
  tf.Server()
  tf.ClusterSpec()
  tf.train.device_replica_setter()
Steep learning curve required
Bandwidth-optimal
Beneficial when utilized with MPI implementation: OpenMPI
Introducing Horovod

1. Stand-alone Python package called Horovod
   1. Installing time cut from an hour to a few minutes
   2. Available to leverage the algorithm across different version of TensorFlow

2. Ring-allreduce implemented with NCCL
   Ability to run algorithm across multiple machines

3. Extended support for models that fit in a single GPUs-> multiple GPUs

4. Broadcast operation introduced
   Enforced consistent initialization of the model on all workers
   Cut down the number of operations a user had to introduce to their GPU program -> 4
```python
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.

with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                       config=config,
                                       hooks=hooks) as mon_sess:
    while not mon_sess.should_stop():
        # Perform synchronous training.
        mon_sess.run(train_op)

```
Provide a high-level understanding of operation timelines across nodes –
good for debugging

Compatible with Chrome’s about:tracing trace event profiling viewer
Motivation:
Many tiny allreduce operations need for models with large amount of tensors: ResNet-101…
Need for efficient design to work on very small tensors

Proposal:
Tensor Fusion - Fuse multiple tiny tensors together before ring-allreduce

1. Determine which tensors are ready to be reduced. Select the first few tensors that fit in the buffer and have the same data type.
2. Allocate a fusion buffer if it was not previously allocated. Default fusion buffer size is 64 MB.
3. Copy data of selected tensors into the fusion buffer.
4. Execute the allreduce operation on the fusion buffer.
5. Copy data from the fusion buffer into the output tensors.
6. Repeat until there are no more tensors to reduce in the cycle.

65% improvement on models with large number of layers on unoptimized TCP network
Benchmarks Again

88% efficiency mark for both models

About twice as fast as standard distributed TensorFlow
TCP vs RDMA

Inception V3 and ResNet-101: exceed 90% scaling efficiency

VGG-16: significant 30% speedup
Future Goals

• Make it easier to install MPI
• Learnings about adjusting model parameters for distributed DL
• More examples of very large models