# On Scale-out Deep Learning Training for Cloud and HPC

Srinivas Sridharan, Karthikeyan Vaidyanathan, Dhiraj Kalamkar, Dipankar Das,
Mikhail E. Smorkalov, Mikhail Shiryaev, Dheevatsa Mudigere, Naveen Mellempudi,
Sasikanth Avancha, Bharat Kaul, Pradeep Dubey

Intel Corporation

[srinivas.sridharan,karthikeyan.vaidyanathan,dhiraj.kalamkar,dipankar.das,mikhail.e.smorkalov,mikhail.shiryaev,
dheevatsa.mudigere,naveen.mellempudi,sasikanth.avancha,bharat.kaul,pradeep.dubey]@intel.com

## ABSTRACT

This paper describes the motivation, design, and implementation of Intel® *Machine Learning Scaling Library* (MLSL) and presents proof-points demonstrating DL training on 100s to 1000s of nodes across Cloud and HPC systems.

## 1 INTRODUCTION

Deep Learning (DL) is driving the adoption of Machine Learning (ML) and Artificial Intelligence (AI) across a wide range of application domains such as image recognition, natural language processing, and autonomous driving. The exponential growth in use of large deep neural networks has accelerated the need for training these deep neural networks in hours or even minutes. This can only be achieved through scalable and efficient distributed training, since a single node/card cannot satisfy the compute, memory, and I/O requirements of today's state-of-the-art deep neural networks. However, scaling synchronous *Stochastic Gradient Descent* (SGD) is still a challenging problem and requires continued research/development. This entails innovations spanning algorithms [6, 11], frameworks [1, 7, 9], communication libraries [2, 8, 15], and system design [10]. In this paper, we describe the philosophy, design, and implementation of Intel® *Machine Learning Scaling Library* (MLSL) and present proof-points demonstrating DL training on 100s to 1000s of nodes across Cloud and HPC systems.

## 2 DESIGN CHOICES AND INSIGHTS

The common parallelization techniques for partitioning work across multiple nodes, are *data parallelism* (replicating the entire model) and *model parallelism* (distributing the model). In [4], we present a detailed theoretical analysis of computation and communication involved in DL training. Based on this analysis, we derived the *compute to communication ratio* that captures the number of compute operations per layer to the communication volume. The goal is to maximize this ratio for best scaling. For data parallelism, we observe that this ratio is a function of the size of output feature-maps, mini-batch size and effectiveness of overlap. Interestingly, it does not depend on the kernel size or number of input/output feature maps or stride. We use these insights to guide different design choices for realizing scalable distributed training.

***Choosing the right work partitioning strategy:*** First, using the methodology in [4], we identify the optimal parallelization strategy for each layer depending on the type of the layer (convolutional, fully connected, etc.), size of output feature maps, and so on. Further, we developed a novel work partition strategy, called *hybrid parallelism*, which partitions the work across both the data and model using the concept of *node groups*; i.e. nodes within a group employ model parallelism and data parallelism is used across groups. One could consider data and model parallelism as two extreme design points of hybrid parallelism with node group size being one and all nodes respectively.

***Increasing concurrency with large batch training:*** For data parallelism, we observe that the compute to communication ratio is proportional to the mini-batch size. This implies, scaling will be negatively impacted as we strong-scale the mini-batch and the mini-batch per node drops. More specifically, communication starts dominating total execution time since communication tends to become more latency bound and there's lesser compute to hide communication. Hence, large batch training is essential for efficient scaling and this observation is in line with recent efforts enabling large mini-batch in training without affecting accuracy [6, 11, 18].

***Overlapping communication and computation:*** Unlike model parallelism, in data parallelism, there's significant opportunity to overlap communication with compute. Each node computes partial weight gradients for its mini-batch in the back-propagation step in each layer and aggregates these partial gradients across all nodes using an *allreduce* operation. These aggregated weight gradients are used to update the weights and only required right before the forward propagation step for that layer in the next iteration. This is captured in the compute to communication ratio and relies on networking library/HW's ability to *asynchronously progress* communication and framework's ability to schedule communication to maximize compute-communication overlap.

***Prioritizing latency-bound communication:*** While overlapping communication with computation across layers is indispensable, the overheads of the first layer's weight gradient communication in data parallelism is fully exposed given lack of useful compute to overlap communication. In other words, while network bandwidth is critical for all other layers, optimizing for network latency is essential for the first layer since size of the weight gradients are typically small(er). This motivates the need for further prioritizing and completing the first layers communication operations before communication operations from later layers even though they were issued earlier. Similarly, in the case of model/hybrid parallelism, activation communication must be prioritized as they may block the next layer's compute.

*Reducing communication volume:* Finally, scaling can be further improved by reducing the volume of communicated data. For instance, this can be achieved through message compression and/or quantization [5, 13, 16]. The growing adoption of lower precision for training, has an impact of communication/scaling as well. At a minimum, while communication should support the same precision as the compute, the precision for communication could be further reduced allowing for improved scaling. However, this entails frameworks, libraries and HW to natively support low precision communication, for guaranteeing correctness and realizing the performance potential.

We now present Intel® *Machine Learning Scaling Library* (MLSL), a core component of our solution stack embodying many of the optimizations and design choices just described.
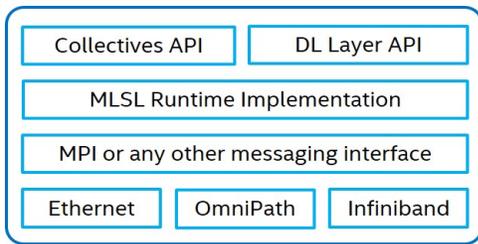
## 3 MACHINE LEARNING SCALING LIBRARY



**Figure 1: Intel MLSL Architecture**

Figure 1 presents the MLSL SW architecture. At the highest level, MLSL exposes two interfaces for frameworks: *collectives* and *DL layer*. The collectives API is similar to *Message Passing Interface* (MPI) collectives interface and supports commonly used collective operations found in DL/ML workloads. The DL Layer API is a higher-level interface that abstracts the exact communication operation depending on the type of parallelism chosen (data, model, or hybrid) for each layer of the neural network at runtime, thus reducing the hassle of supporting these different scenarios within each framework explicitly.

Regardless of the chosen interface, MLSL's runtime implementation enables novel DL specific optimizations unavailable in MPI and other communication libraries, such as asynchronous progress for compute-communication overlap, dedicating one or more cores for driving the network in an optimal manner, message prioritization, and collectives in low-precision data types. MLSL's flexible API enables these runtime optimizations to be applied across frameworks and lowers the effort required in optimizing each framework independently. Furthermore, MLSL uses existing communication libraries, such as MPI, for commonly used control path operations but only implements performance critical data path operations in an optimal manner.

The benefits of MLSL's design and implementation becomes self-evident when examining one of the DL-specific communication optimizations in greater detail. Like mentioned earlier, with data parallelism the weight gradient communication in the first layer is latency bound and the updated weights are required immediately in the forward pass. However, MPI interface and implementations do not support prioritizing such messages. MLSL's *message prioritization* feature overcomes this limitation by preempting an ongoing large weight gradient exchange operation from one of the later layers and instead prioritizes the smaller weight gradient allreduce from the first layer to proceed. The preempted operations are completed in an optimal manner as and when they are required in the forward pass and not necessarily the order in which they were originally issued. This optimization resulted in 1.8x to 2.2x reduction in exposed communication time for standard topologies such as Resnet-50, VGG-16, and Googlenet on Intel® Xeon® Gold 6148 processors (code-named Skylake) and 10Gbps Ethernet. Additional DL specific optimizations, such as message quantization and persistent collectives [14], are currently being evaluated and will be made available as part of upcoming MLSL SW releases.
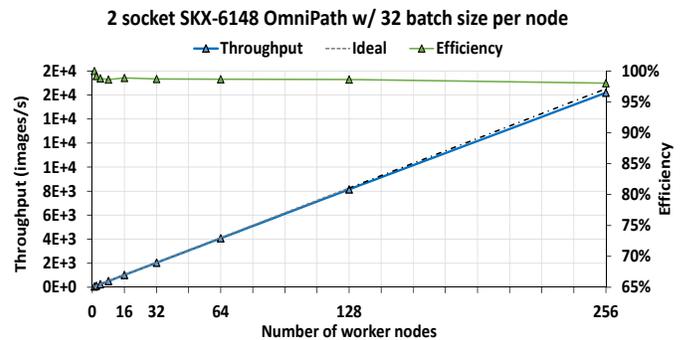


**Figure 2: Resnet-50 scaling on Intel Xeon / Omnipath**

MLSL has been integrated with numerous DL frameworks, including but not limited to, BVLC Caffe [7], Google⋆TensorFlow, and Intel® nGraph™[9]. While the integration strategy differs in each case, the use of a single library facilitates common set of optimizations across all these frameworks. For instance, Figure 2 presents Resnet-50 scaling on Intel® Xeon® Gold 6148 processors (code-named Skylake) and Intel Omnipath fabric using Intel Caffe and MLSL demonstrate 90% scaling on 256 nodes (75.8% top-1 validation accuracy). Further, this solution has been used to scale deep neural networks solving scientific pattern classification problems to 9600 Xeon-Phi nodes [12] and to train Resnet-50 in 40 minutes on 256 nodes on the MareNostrum system at Barcelona Supercomputing Center [3]. For TF, we have developed a new distributed solution that adopts Uber Horovod [17] interface but uses MLSL to achieve higher scaling performance over the out-of-box Horovod MPI implementation. We observe >93% scaling efficiency on the fore-mentioned Intel® Xeon® system on 64 nodes. For nGraph, we added new graph passes to insert non-blocking MLSL collective operations and novel scheduling optimizations to ensure maximum compute-communication overlap. More details on MLSL with TF and nGraph will be shared in the near future.

We plan to continue extending MLSL with novel DL features and optimizations. We are actively looking for collaborating with researchers/developers interested in using MLSL and extending the scaling envelope for DL workloads.

# REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Minsik Cho, Ulrich Finkler, Sameer Kumar, David S. Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI DDL. *CoRR* abs/1708.02188 (2017). arXiv:1708.02188 http://arxiv.org/abs/1708.02188

[3] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. 2017. Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train. (2017). https://arxiv.org/abs/1711.04291

[4] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidyanathan, Srinivas Sridharan, Dhiraj D. Kalamkar, Bharat Kaul, and Pradeep Dubey. 2016. Distributed Deep Learning Using Synchronous Stochastic Gradient Descent. *CoRR* abs/1602.06709 (2016). arXiv:1602.06709 http://arxiv.org/abs/1602.06709

[5] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* (2015).

[6] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017). arXiv:1706.02677 http://arxiv.org/abs/1706.02677

[7] Intel. 2018. Intel Caffe. (2018). https://github.com/intel/caffe

[8] Intel. 2018. Intel Machine Learning Scalability Library (MLSL). (2018). https://github.com/intel/MLSL

[9] Intel. 2018. Intel Nervana Graph. (2018). https://github.com/NervanaSystems/ngraph

[10] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *CoRR* abs/1609.04836 (2016). arXiv:1609.04836 http://arxiv.org/abs/1609.04836

[12] Thorsten Kurth, Jian Zhang, Nadathur Satish, Evan Racah, Ioannis Mitliagkas, Md. Mostofa Ali Patwary, Tareq Malas, Narayanan Sundaram, Wahid Bhimji, Mikhail Smorkalov, Jack Deslippe, Mikhail Shiryaev, Srinivas Sridharan, Prabhat, and Pradeep Dubey. 2017. Deep Learning at 15PF: Supervised and Semi-supervised Classification for Scientific Data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 7, 11 pages. https://doi.org/10.1145/3126908.3126916

[13] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. *arXiv preprint arXiv:1712.01887* (2017).

[14] Bradley Morgan, Daniel J. Holmes, Anthony Skjellum, Purushotham Bangalore, and Srinivas Sridharan. 2017. Planning for Performance: Persistent Collective Operations for MPI. In *Proceedings of the 24th European MPI Users' Group Meeting (EuroMPI '17)*. ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3127024.3127028

[15] NVIDIA. 2018. NVIDIA Collective Communications Library (NCCL). (2018). https://developer.nvidia.com/nccl

[16] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[17] Alex Sergeev and Mike Del Balso. 2017. Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow. (2017). https://eng.uber.com/horovod/

[18] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2017. Don't Decay the Learning Rate, Increase the Batch Size. *CoRR* abs/1711.00489 (2017). arXiv:1711.00489 http://arxiv.org/abs/1711.00489