Scalable CNN Training on Large-Scale HPC Systems

Nikoli Dryden[1,2], Naoya Maruyama[2], Tom Benson[2], Tim Moon[2], Marc Snir[1], Brian Van Essen[2]
[1]University of Illinois at Urbana-Champaign, [2]Lawrence Livermore National Laboratory
{dryden2,snir}@illinois.edu, {maruyama3,benson31,moon13,vanessen1}@llnl.gov
Extended Abstract

1 Overview

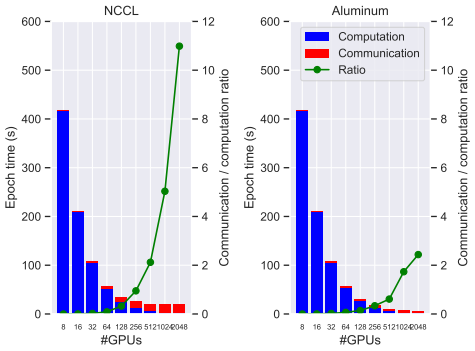
Accelerating the training of deep neural networks has become increasingly important, particularly as model complexity and data sizes grow. Training a modern CNN to convergence can take days to weeks. To alleviate this, clusters of GPUs are employed to reduce training times to more reasonable levels. Taking advantage of such clusters requires that training be scaled, and doing so typically falls into one of two broad categories: strong and weak scaling. In strong scaling, a fixed model and mini-batch size are trained with more GPUs; in weak scaling, a fixed mini-batch size per GPU is used, growing the mini-batch size as more GPUs are added.

Weak scaling to large mini-batches (8k or larger) has recently become viable for datasets such as ImageNet [1, 2]. While this results in a fixed amount of work per GPU to hide communication, at large scale, communication costs still dominate, limiting the profitability of additional GPUs. However, it is not clear that large mini-batch techniques generalize to all datasets or problems [3], restricting them to training with small mini-batches. Further, emerging datasets from HPC and scientific simulations (e.g. [4–6]) are different from ImageNet and similar data: each sample may be very large. Because training requires intermediate activations, large samples result in very high pressure on GPU memory; the case where a single sample is too large to fit in memory is also possible.

To address communication overheads, primarily due to allreduces for accumulating gradient updates in decentralized, synchronous, data-parallel SGD, we first observe that allreduces are typically performed on a per-buffer basis. In modern deep networks, these buffers are as small as 1 KiB (e.g. due to batchnorm parameters), and many are firmly in a *latency*-dominated regime, especially at large scale. Most decentralized DL communication frameworks leverage bandwidth-optimized ring algorithms for allreduces [7–10], and have limited support for latency-optimized algorithms (Rabenseifner's algorithm in Gloo; tensor fusion in Horovod). In contrast, MPI distributions [11, 12] typically feature a large suite of tuned allreduce algorithms [13]. However, leveraging MPI directly requires performing additional synchronization to ensure data computed in GPUs is ready, resulting in a bulk-synchronous computation that limits GPU and network utilization [14]. It also limits CPU/GPU asynchrony, making I/O prefetching more difficult. We address this with our Aluminum communication library [15], which dynamically selects latency- or bandwidth-optimized allreduce algorithms while providing GPU-friendly semantics supporting asynchrony.

To strong scale deep networks, we exploit finer-grained parallelism beyond the sample-parallelism used in standard data-parallel training by additionally decomposing each sample spatially. This enables additional scaling beyond the mini-batch size, and enables large samples to be decomposed so they, along with the corresponding activations, fit in GPU memory. Spatial decomposition is relatively straightforward and follows a similar pattern as stencil computations from scientific computing [16]. Different portions of a sample are partitioned onto different GPUs. As remote data along a partition border will be needed for the convolution to be computed, a halo exchange is used to swap data. This is required on forward prop (data from input x, to compute y) and backward-data (data from dL/dy, to compute dL/dx). Halo exchanges add communication overhead, but standard optimizations to overlap interior computation with halo exchanges can be used to reduce this. Several prior approaches have made similar attempts at parallelization. AlexNet [17] partitioned filters among two GPUs, but only communicated at certain layers; Coates et al. [18] spatially partitioned locally-connected layers;

Preprint. Work in progress.



N	Sample	Spatial (1 sample/node)
8	0.521s	0.164s (3.2x)
16	0.527s	0.158s (3.3x)
32	0.534s	0.168s (3.2x)
64	0.53s	0.184s (2.9x)
128	0.539s	0.219s (2.5x)
256	0.538s	0.272s (2.0x)

Figure 1: Left: Weak scaling results for ResNet-50 using a synthetic benchmark (no I/O, etc.). Bars break down runtime by computation and unoverlapped communication. (Adapted from [15].) Right: Strong scaling results for the mesh model on 1K input data, where N is the global mini-batch size. Spatial uses four GPUs per node to process each sample (1024 GPUs used for $N = 256$).

DistBelief [19] and Project Adam [20] use parameter servers to partition filters. Memory pressure has been addressed via "micro-batching" [21] or "swapping"/"prefetching" [22–24]. However, these can add additional overhead compared to an optimized implementation [25, 26], and do not address the case of a single very large sample.

2 Results

Fig. 1 presents brief results illustrating the performance of our Aluminum library and spatial partitioning. Results were obtained on the Sierra [27] and Lassen [28] supercomputers, which consist of compute nodes with 4 Nvidia V100 GPUs with NVLINK2 and dual-rail InfiniBand EDR.

Two models and datasets are used for evaluation. For weak scaling, we use ResNet-50 [29] on ImageNet-1K [30]. For strong scaling, we consider a synthetic dataset for a 2D mesh-tangling problem. This is a semantic segmentation problem, where for each input pixel we predict whether or not a mesh cell at that location needs to be relaxed. CNNs incorporating global information from multiphysics simulations may be able to detect degenerate meshes. However, as interesting simulations are high-resolution, it has not been possible to train on this data before. We use show results with 1024×1024 input data with 18 channels consisting of state variables and mesh quality metrics from a hydrodynamics simulation. Our model is simply a proof-of-concept stack of convolutions; we leave better models to future work now that training is feasible.

In Fig. 1 (left), weak scaling is compared when using only NCCL to Aluminum, which dynamically selects between NCCL for bandwidth-dominated regimes and latency-optimized allreduces. At small scale, there is sufficient compute to overlap communication, but at large scale communication can inhibit scaling. NCCL uses only ring algorithms, which have latency that scales linearly in the number of processors p; Aluminum supports tree allreduces that scale logarithmically ($\log p$ or $2 \log p$). Aluminum enables CPU/GPU asynchrony by associating streams with communicators and ensuring no operation on a stream blocks any other operation using a progress engine.

Fig. 1 (right) presents strong scaling results for our mesh model. Here we fix a mini-batch size (N) and compare sample parallelism (with one sample per GPU, the most possible) to four-way spatial parallelism, where four GPUs process a single sample. We see significant speedups at every scale by taking advantage of spatial parallelism, as input samples are large enough for this to be a benefit. At the largest scales, we see slightly less speedup due to allreduce overheads, but improvements are still at least 2x. With larger input data (e.g. 2048×2048), sample parallelism is not possible at all due to memory requirements per sample exceeding GPU memory. However, spatial parallelism enables training, while scaling well for larger mini-batches. We have also experimented with spatial decomposition to strong-scale ResNet-50; this is a more difficult task, because layers in ResNet-50 have smaller spatial domains. Nonetheless, we observe speedups of over 2x with 256-sample mini-batches (compared to sample parallelism with 32 samples/GPU), indicating that spatial decomposition can accelerate training when mini-batch sizes cannot grow, even for standard models.

References

[1] P. Goyal *et al.*, "Accurate, large minibatch SGD: training ImageNet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[2] Y. You *et al.*, "ImageNet training in minutes," in *ICPP*, 2018.

[3] N. S. Keskar *et al.*, "On large-batch training for deep learning: Generalization gap and sharp minima," in *ICLR*, 2017.

[4] T. Kurth *et al.*, "Deep learning at 15PF: supervised and semi-supervised classification for scientific data," in *SC*, 2017.

[5] ——, "Exascale deep learning for climate analytics," in *SC*, 2018.

[6] A. Mathuriya *et al.*, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC*, 2018.

[7] Baidu Research, "Baidu allreduce," https://github.com/baidu-research/baidu-allreduce, 2018.

[8] NVIDIA, "NVIDIA collective communications library," https://developer.nvidia.com/nccl, 2018.

[9] Facebook, "Gloo," https://github.com/facebookincubator/gloo, 2018.

[10] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[11] MPI Forum, "MPI," https://www.mpi-forum.org/, 2018.

[12] H. Wang *et al.*, "MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters," *Computer Science-Research and Development*, vol. 26, 2011.

[13] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[14] S. Potluri *et al.*, "GPU-centric communication on NVIDIA GPU clusters with InfiniBand: A case study with OpenSHMEM," in *HiPC*, 2017.

[15] N. Dryden *et al.*, "Aluminum: An asynchronous, GPU-aware communication library optimized for large-scale training of deep neural networks on HPC systems," in *MLHPC*, 2018.

[16] N. Maruyama *et al.*, "Physis: An implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers," in *SC*, 2011.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, 2012.

[18] A. Coates *et al.*, "Deep learning with COTS HPC systems," in *ICML*, 2013.

[19] J. Dean *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[20] T. M. Chilimbi *et al.*, "Project Adam: Building an efficient and scalable deep learning training system." in *OSDI*, 2014.

[21] Y. Oyama *et al.*, "Accelerating deep learning frameworks with micro-batches," *arXiv preprint arXiv:1804.04806*, 2018.

[22] M. Rhu *et al.*, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *MICRO*, 2016.

[23] C. Meng *et al.*, "Training deeper models by GPU memory optimization on TensorFlow," in *ML Systems Workshop @ NIPS*, 2017.

[24] L. Wang *et al.*, "Superneurons: dynamic GPU memory management for training deep neural networks," in *PPoPP*, 2018.

[25] R. Landaverde *et al.*, "An investigation of unified memory access performance in CUDA," in *HPEC*, 2014.

[26] Ł. Jarząbek and P. Czarnul, "Performance evaluation of unified memory and dynamic parallelism for selected parallel cuda applications," *The Journal of Supercomputing*, vol. 73, 2017.

[27] Lawrence Livermore National Laboratory, "Sierra," https://hpc.llnl.gov/hardware/platforms/sierra, 2018.

[28] LLNL, "Lassen," https://hpc.llnl.gov/hardware/platforms/lassen, 2018.

[29] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.

[30] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, 2015.