Supplementary Material:

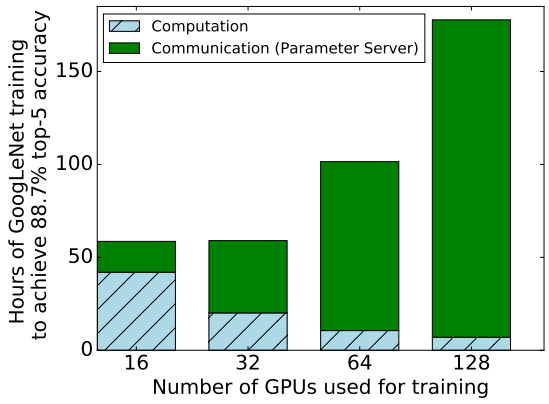
FireCaffe: near-linear acceleration of deep neural network training on compute clusters

Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Kurt Keutzer
DeepScale* and UC Berkeley
{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu

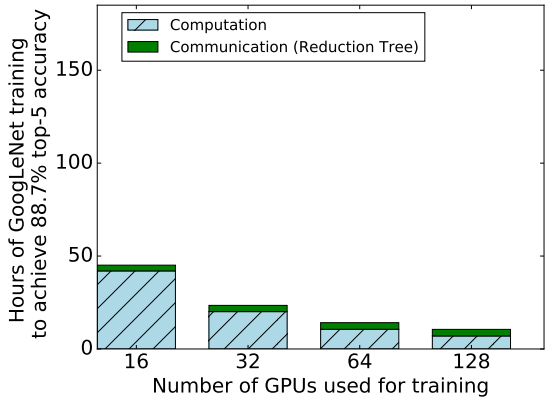
Appendix A: Further analysis of FireCaffe results

One of the important techniques in FireCaffe is the use of allreduce collective operations (e.g. reduction trees) instead of a parameter server for communicating gradient updates in a compute cluster. In Figure 4 of the main paper, we microbenchmarked the communication portion of our distributed DNN training implementation, and we found that the scalability of a parameter server is much worse than the scalability of a reduction tree. At this stage, a reasonable question is: if we used a parameter server in FireCaffe, would this substantially degrade the overall training time? We address this question in Figures 1(a) and 1(b). Observe in Figure 1(a) that, as we scale beyond 16 GPUs, the parameter server overhead dominates the overall execution time. Fortunately, we find in Figure 1(b) that our reduction tree approach scales much more efficiently, yielding a 47x speedup on 128 GPUs, training GoogLeNet [14] in just 10.5 hours.

One of the strategies that we use to expose parallelism is to increase the batch size as much as possible, on a limited budget of epochs, or passes through the dataset. The CVPR reviewers encouraged us to provide convergence plots to show intuition of how changing the batch size impacts convergence. We show convergence plots of NiN [10] training with two different batch sizes (default of 256, and our preferred batch size of 1024) in Figures 2(a) and 2(b).



(a) using a parameter server



(b) using a reduction tree

Figure 1. Contributions of communication and computation to GoogLeNet training time in FireCaffe.

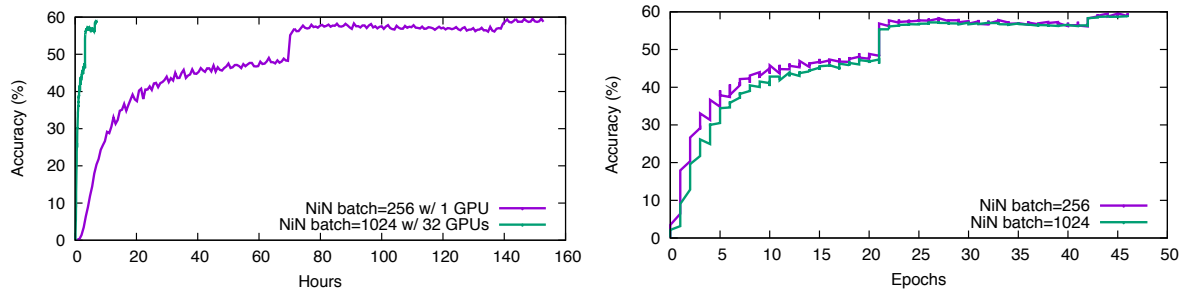
Appendix B: Complementary approaches to accelerate DNN training

We discussed related work throughout the paper, but we now provide a brief survey of additional techniques to accelerate deep neural network training. Several of the following techniques could be used in concert with FireCaffe to further accelerate DNN training.

Accelerating convolution on GPUs

In the DNN architectures discussed in this paper, more than 90% of the floating-point operations in forward and backward propagation reside in convolution layers, so accelerating convolution is key to getting the most out of each GPU. Recently, a number of techniques have been developed to accelerate convolution on GPUs. Unlike CPUs, NVIDIA GPUs have an inverted memory hierarchy, where the register file is larger than the L1 cache. Volkov and

*http://deepscale.ai



(a) time vs accuracy

(b) epochs vs accuracy

Figure 2. Convergence plots for NiN.

Demmel [15] pioneered a *communication-avoiding* strategy to accelerate matrix multiplication on GPUs by staging as much data as possible in registers while maximizing data reuse. Iandola *et al*. [6] extended the communication-avoiding techniques to accelerate 2D convolution; and cuDNN [2] and maxDNN [9] extended the techniques to accelerate 3D convolution. FireCaffe can be coupled with current and future GPU hardware and convolution libraries for further speedups.

Reducing communication among servers

Reducing the quantity of data communicated per batch is a useful way to increase the speed and scalability of DNN training. There is an inherent tradeoff here: as gradients are more aggressively quantized, training speed goes up, but the model's accuracy may go down compared to a non-quantized baseline. While FireCaffe uses 32-bit floating-point values for weight gradients, Jeffrey Dean stated in a recent keynote speech that Google often uses 16-bit floating-point values for communication between servers in DNN training [4]. Along the same lines, Wawrzynek *et al*. used 16-bit weights and 8-bit activations in distributed neural network training [16]. Going one step further, Seide *et al*. used 1-bit gradients for backpropagation, albeit with a drop in the accuracy of the trained model [12]. Finally, a related strategy to reduce communication between servers is to discard (and not communicate) gradients whose numerical values fall below a certain threshold. Amazon presented such a thresholding strategy in a recent paper on scaling up DNN training for speech recognition [13]. However, Amazon's evaluation uses a proprietary dataset, so it is not clear how this type of thresholding impacts the accuracy compared to a well-understood baseline.

So far in this section, we have discussed strategies for compressing or quantizing data to communicate in distributed DNN training. There has also been a series of studies on applying dimensionality reduction to DNNs once they have been trained. Jaderberg *et al*. [7] and Zhang *et al*. [17] both use PCA to compress the weights of DNN models by up to 5x, albeit with a substantial reduction in the model's classification accuracy. Han *et al*. [5] use a combination of pruning, quantization, and Huffman encoding to compress the weights of pretrained models by 35x with no reduction in accuracy. Thus far, these algorithms have only been able to accelerate DNNs at test time.

Appendix C: Frequently Asked Questions about FireCaffe

1. *I use Caffe. Will my DNN models and network definitions work with FireCaffe?*
Yes.

2. *What are some good DNN architectures to use with FireCaffe?*
We offer intuition and analysis of this in Section 5 of the main paper, but here we offer a short summary of what you need to know. As a general rule, DNN architectures with fewer parameters train faster in FireCaffe. We have found that NiN [10] and AlexNet [8] provide similar accuracy on ImageNet and other popular data benchmarks. However, NiN has 8x fewer parameters and therefore incurs 8x less communication cost than AlexNet. If you were planning to use AlexNet, we recommend trying NiN for faster training.

3. *I want to design my own DNN architectures that will scale well nicely in FireCaffe while producing good accuracy on my problem. What design tradeoffs should I consider?*
If you're designing your own DNN architectures, do your best to economize on parameters. Also, reducing the convolution and pooling strides may improve accuracy; this doesn't require additional parameters and doesn't hurt training scalability.

4. *Is it better to use FireCaffe with one server that contains many GPUs, or is it better to distribute the GPUs distributed over many servers?*
FireCaffe is compatible with both of these scenarios. FireCaffe can even run across many servers that each contain

many GPUs. To avoid stragglers during backpropagation, we prefer running FireCaffe on a collection of identical GPUs (e.g. all Titan X or all K80; but preferrably not a mixture of K80 and Titan X). For optimal speed and utilization, FireCaffe prefers a low-latency interconnect such as Infiniband between servers.

5. *How does FireCaffe handle the storage and loading of training data?*
As in Caffe, FireCaffe can ingest LMDB databases [3] of training data. This data format is agnostic to the type of data (e.g. images, audio, text), so long as each training data item consists of a vector (e.g. pixels, audio waveform, text trigram) and a label (e.g. dog or cat). We store the LMDB database of training data on a distributed filesystem that is accessable to all workers. During DNN training, each worker is responsible for loading its own training data from the distributed filesystem.

6. *Does FireCaffe use nondeterministic techniques such as Hogwild [11]?*
No. Given a specific seed for the random number generator, FireCaffe produces repeatable numerical results, just like ordinary Caffe. In fact, for a model with Dropout disabled, FireCaffe will give you the exact same numerics as ordinary Caffe. With Dropout enabled, FireCaffe handles randomization in a slightly different way than Caffe, but the numerics are still deterministic for FireCaffe with a constant number of GPUs.

7. *How did FireCaffe get its name?*
Asanovic and Patterson presented a roadmap for warehouse-scale computing in the year 2020, which they call *FireBox* [1]. The grand vision is that a typical large-scale commercial datacenter will have extremely low latency connections within and across racks, likely using photonic networking hardware. Low-latency network hardware is crucial not only for today's mainstream applications like distributed databases and search engines, but also for emerging applications such as deep neural network training at large scale. We have designed FireCaffe with FireBox-style warehouse-scale computing in mind.

References

[1] K. Asanovic and D. Patterson. FireBox: a hardware building block for 2020 warehouse-scale computers. In *USENIX FAST*, 2014. 3

[2] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: efficient primitives for deep learning. *arXiv:1410.0759*, 2014. 2

[3] H. Chu. MDB: A memory-mapped database and backend for openldap. In *LDAPCon*, 2011. 3

[4] J. Dean. Keynote: Large scale deep learning. In *GPU Technology Conference*, 2015. 2

[5] S. Han, H. Mao, and W. J. Dally. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. *arXiv:1510.00149*, 2015. 2

[6] F. N. Iandola, D. Sheffield, M. Anderson, P. M. Phothilimthana, and K. Keutzer. Communication-minimizing 2d convolution in gpu registers. In *ICIP*, 2013. 2

[7] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv:1405.3866*, 2014. 2

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012. 2

[9] A. Lavin. maxDNN: an efficient convolution kernel for deep learning with maxwell gpus. *arXiv:1501.06633*, 2015. 2

[10] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013. 1, 2

[11] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011. 3

[12] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *INTERSPEECH*, 2014. 2

[13] N. Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *INTERSPEECH*, 2015. 2

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014. 1

[15] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Supercomputing*, 2008. 2

[16] J. Wawrzynek, K. Asanovic, B. Kingsbury, D. Johnson, J. Beck, and N. Morgan. Spert-ii: A vector microprocessor system. *Computer*, 1996. 2

[17] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. *arXiv:1411.4229*, 2014. 2