开发了自适应的算法

2016 2nd Workshop on Machine Learning in HPC Environments

Communication Quantization for Data-parallel Training of Deep Neural Networks

Nikoli Dryden
University of Illinois at Urbana-Champaign
and Lawrence Livermore National Laboratory
Email: dryden2@illinois.edu

Tim Moon
Stanford University
and Lawrence Livermore National Laboratory
Email: tym1@stanford.edu

Sam Ade Jacobs
Lawrence Livermore National Laboratory
Livermore, CA, USA
Email: jacobs32@llnl.gov

Brian Van Essen
Lawrence Livermore National Laboratory
Livermore, CA, USA
Email: vanessen1@llnl.gov

***Abstract*—We study data-parallel training of deep neural networks on high-performance computing infrastructure. The key problem with scaling data-parallel training is avoiding severe communication/computation imbalance. We explore quantizing gradient updates before communication to reduce bandwidth requirements and compare it against a baseline implementation that uses the MPI allreduce routine. We port two existing quantization approaches, one-bit and threshold, and develop our own adaptive quantization algorithm. The performance of these algorithms is evaluated and compared with `MPI_Allreduce` when training models for the MNIST dataset and on a synthetic benchmark. On an HPC system, `MPI_Allreduce` outperforms the existing quantization approaches. Our adaptive quantization is comparable or superior for large layers without sacrificing accuracy. It is 1.76 times faster than the next best approach for the largest layers in our benchmark and achieves near-linear speedup in data-parallel training.**

扩展数据并行训练的关键问题是避免严重的通信/计算不平衡。

我们在通信前探索量化梯度更新以减少带宽需求，并将其与使用MPI allreduce例程的基线实现进行比较。我们移植了两种现有的量化方法，一比特和阈值，并开发了我们自己的自适应量化算法。

开发我们自适应的量化算法

1. Introduction

Recent work in deep learning has achieved state-of-the-art results on many machine learning applications. This is driven in part by massive growth in datasets and, in some cases, by very large models [1], [2]. Training very large models can be very computationally expensive, and is primarily done with relatively small clusters of commodity GPU machines, often in cloud environments. In contrast, training deep neural networks (DNNs) on dedicated high-performance computing infrastructure—primarily the domain of large scientific simulations—is little-studied. HPC resources provide, in particular, highly optimized network infrastructure, in the form of high bisection bandwidth and low latency, that we can exploit. Most existing DNN training software is instead targeted at small clusters and heterogeneous and desktop environments or toward producing models for mobile or embedded devices.

Two common approaches to parallelizing DNN training are model parallelism and data parallelism (see [3, section 12.1.3]). In the former, the parameters of a single model are distributed and updates to it are parallelized through e.g. parallelized matrix operations. In the latter, the models are duplicated and different subsets of the dataset are processed concurrently, with each model sharing its updates with all the others. The chief difficulty in scaling either method, and especially the data-parallel approach, is severe communication/computation imbalance: without care, more time is spent communicating than doing useful work [4]. Here we focus on overcoming this in the data-parallel regime. For even moderately sized layers, each message sent is quite large (many megabytes or gigabytes), and so this regime is bandwidth-dominated.

Our approach to scaling data-parallel training is twofold: The main thrust is to reduce bandwidth usage while secondly ensuring our communication algorithms are efficient. To reduce bandwidth we *quantize* inter-model communication, mapping 32-bit floating point entries to smaller representations. Quantized data is then reconstructed using a pre-computed dictionary. This trades additional computation and potential accuracy loss for reduced bandwidth. We implement and evaluate several existing quantization approaches, some variants thereof, and a novel “adaptive” quantization algorithm we developed. Our adaptive quantization dynamically determines the data to send and good reconstruction values for it. To optimize communication, we implement a custom version of the allreduce collective communication primitive that is aware of the quantization and has good performance when data volumes are large.

我们扩展数据并行训练的方法有两个方面：主要的推动力是减少带宽的使用，其次确保我们的通信算法是有效的。

然后使用预计算字典重构量化数据。

我们的自适应量化动态地确定数据发送和良好的重构值。

为了优化通信，我们实现了allreduce集合通信原语的自定义版本，该版本能够感知量化，并且在数据量较大时性能良好。

We implement this in LBANN, the Livermore Big Artificial Neural Network toolkit, a new library for training DNNs at scale on HPC resources [1]. It supports model parallelism through distributed matrix operations built on the Elemental library [5], and data parallelism via distributed mini-batches, which we extend to use our quantization algorithms. The underlying communication layer builds on the Message Passing Interface (MPI). Our overall architecture

978-1-5090-3882-4/16 $31.00 © 2016 IEEE
DOI 10.1109/MLHPC.2016.4

is illustrated in Figure 1.

Our major contributions are:

- Implement existing communication quantization algorithms on HPC infrastructure and develop our own adaptive quantization algorithm to address their shortcomings. We also develop our own allreduce implementation.
- Compare existing and new data-parallel communication quantization algorithms with each other and with `MPI_Allreduce`.
- Study the viability of data-parallel training on HPC infrastructure.

Our implementation is part of LBANN, which is available at https://github.com/LLNL/lbann.

2. Related Work

There are several deep learning toolkits that support data-parallel training, but very few of which are targeted to dedicated HPC resources. Commonly-used toolkits such as Caffe [6] and Keras [7] are not designed for distributed computation. A Caffe variant, mpi-caffe [8], does support distributed computation, but does no quantization and has little in-depth evaluation of scalability. TensorFlow [9] is another commonly-used toolkit, which includes extensive support for distributed computation, but is not targeted at HPC resources. It makes use of centralized parameter servers instead of using a fully-distributed approach to aggregating gradient updates. Further, while it supports quantizing data to 16 bits, our quantization is much more extensive. FireCaffe [10] was evaluated on HPC resources (the Titan supercomputer) and makes use of optimized communication routines for data-parallel training; however, we are not aware of it doing any quantization. Similarly, the work of Coates et al. [11] looks at scaling DNN training on commodity off-the-shelf HPC equipment using a GPU cluster with InfiniBand and MPI, but focuses on a purely model-parallel approach and also does not do quantization. Lastly, Chung et al. [12] evaluate DNN scaling on a Blue Gene/Q supercomputer. However, instead of using SGD, they make use of a distributed Hessian-free algorithm. LBANN and the work of Coates et al. are the only toolkits to support layers that are too large for one physical node.

它使用集中的参数服务器，而不是使用完全分布的方法来聚合梯度更新。

There are two existing quantization approaches for data-parallel training that we are aware of: the one-bit quantization of Seide et al. [13] and what we call the threshold quantization of Strom [14]. One-bit quantization was evaluated on 24 InfiniBand-connected servers with dual GPUs. Threshold quantization was evaluated on up to 80 Amazon Web Services Elastic Compute Cloud G2 GPU instances, each with a single GPU.

We have implemented both of these within LBANN for comparison and evaluation purposes and discuss both their algorithms and our results in the sequel. Neither one-bit nor threshold quantization were compared against existing MPI methods such as `MPI_Allreduce`, and both were implemented on GPUs, whereas we have ported them to run on CPUs and evaluated them against `MPI_Allreduce`.

3. Communication Quantization

Here we detail the existing quantization algorithms we implement, one-bit and threshold quantization, some variations, and our adaptive quantization algorithm. We also discuss our communication algorithm.

3.1. One-bit quantization

One-bit quantization was developed by Seide et al. [13] to address communication/computation imbalances when training speech DNNs. To accomplish this, when exchanging gradient updates, each update is quantized to one bit and packed. This results in a significant reduction in data volume: if updates would previously have been sent as 32-bit floating point values, this reduces data volume by a factor of 32. As quantization inevitably introduces error, this approach incorporates *error feedback*: the quantization error is retained and locally added to the corresponding gradient in the next mini-batch prior to quantization. This allows the quantization process to eventually add in the full, true, value of each gradient update, just split across multiple mini-batches in a form of delayed updates.

为了实现这一点，当交换梯度更新时，每个更新都被量化为一个比特并打包。

由于量化不可避免地引入了误差，这种方法引入了误差反馈:量化误差被保留下来，并在量化前的下一个小批量中局部加入相应的梯度。

The actual quantization is quite simple: gradient updates greater than or equal to zero are encoded using the value 1, and those less than zero with a 0. The reconstruction values are chosen to be the means of the non-negative and negative updates, respectively, in order to minimize the quantization error. This is done column-wise over the weight matrix. In each data exchange, the two reconstruction values are transmitted along with their respective quantized column.

实际的量化非常简单:大于或等于0的梯度更新使用值1进行编码，而小于0的更新使用0进行编码。选择重构值作为非负更新和负更新的手段，使量化误差最小化。

在每次数据交换中，两个重构值连同各自的量化列一起传输。

Seide et al. also recommend that AdaGrad [15] be performed during the allreduce step (see below) to both distribute the optimization and to put the gradient updates in a more homogeneous range to assist quantization.

Seide等人还建议在allreduce步骤中执行AdaGrad[15]，以分发优化，并将梯度更新放在更均匀的范围内，以帮助量化。

We have additionally developed a slight modification to this approach in order to reduce the computation required for quantizing large matrices. Instead of exactly computing the means for each column, we randomly sample a fixed number of entries from each column and use these to approximate the mean. While this potentially worsens the reconstruction error, error feedback helps compensate for this, and it works well empirically.

3.2. Threshold quantization

Threshold quantization was developed by Strom [14] to address similar problems as one-bit quantization. Indeed, it is quite similar in approach and relies on error feedback to maintain good accuracy; the key difference is in the quantization and reconstruction. A fixed threshold τ is chosen in advance. Gradient updates greater than τ are encoded with the value 1, and those less than $-\tau$ with a 0. Updates of magnitude less than τ are not sent at all, reducing

量化和重建是关键

更新的大小小于τ不发送,减少发送的数据量。

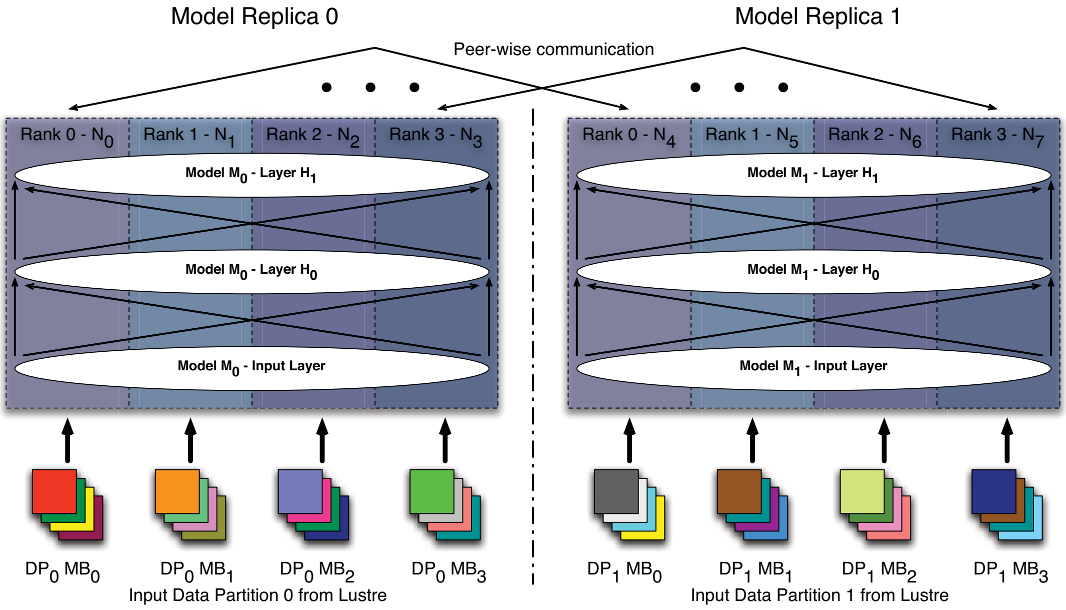

Figure 1. The LBANN model- and data-parallel architecture. This shows two-way data parallelism via model replication and four-way model parallelism with distributed mini-batches in each replica. Within each model, the appropriate parameters of each mini-batch are fed to ranks, and these ranks implement training with distributed matrix operations. Once the mini-batch completes, corresponding ranks in each model communicate their parameter updates using peer-wise collective communication. This communication is quantized to reduce bandwidth requirements.

因此，远程更新是稀疏的，我们同时通信量化值及其相关索引。这是通过发送一个32位的单词来实现的，其中一个位是量化值，另一个位是31位，用于存储更新的索引。

the volume of data sent. Therefore the remote updates are sparse and we communicate both the quantized value and its associated index. This is accomplished by sending a 32-bit word, one bit of which is the quantized value and 31 bits of which are used to store the index of the update. Using only 31 bits to store indices limits the size of models, but due to the model parallelism in our implementation, each process would need to locally store more than 2^{31} parameters for this to be an issue. If need be, the approach can be extended to use larger words.

仅使用31位来存储索引限制了模型的大小，但是由于我们实现中的模型并行性，每个进程需要本地存储超过231个参数才能解决这个问题。

The reconstruction value is simply τ or $-\tau$, respectively, and error feedback is applied as normal. Threshold quantization requires a carefully-chosen threshold to ensure sufficiently small data volumes, but is cheaper to compute than one-bit quantization.

Strom还指出，这些位置非常适合于delta编码[16，第27章]，然后使用Golomb-Rice编码[17]进行无损压缩

Strom also notes that the positions are quite amenable to delta encoding [16, chapter 27] followed by lossless compression using Golomb-Rice coding [17]. However, they note that it introduces a small transmission delay and do not use it in experiments. We have also implemented this and examine the overhead involved in our evaluations.

3.3. Adaptive quantization

We developed adaptive quantization to address deficiencies we observed in one-bit and threshold quantization and to obtain "the best of both worlds" from them. One-bit quantization achieves good reconstruction and low error, but is more computationally expensive than threshold quantization and cannot reduce the data volume by more than a factor of 32. Threshold quantization is fast, but choosing a good τ can be difficult, especially for different models, and using τ as the reconstruction value is sub-optimal. A fixed threshold can also lead to a degenerate situation where error feedback builds up, leading to large values of data being transmitted.

Adaptive quantization uses a fixed *proportion*, π, which indicates the proportion of gradient updates to be sent after processing each mini-batch. The first step is to determine positive and negative thresholds τ^+ and τ^- that satisfy the desired proportion for the current mini-batch. To do this, suppose there are k non-negative updates; we therefore want to send the largest k/π of them.[1] Determining τ^+ can then be done by using a selection algorithm (e.g. quickselect) to find the k/πth largest update. τ^- is determined the in the same manner, finding the smallest (thus, largest in magnitude) k/π updates. Doing this after applying the error feedback ensures that we always send a fixed proportion of the gradient updates, regardless of the state of the updates, while ensuring slowly-growing updates eventually accumulate and are sent.

For the reconstruction values, instead of using the thresholds as done in threshold quantization, we compute the means of the values greater than τ^+ and less than τ^-, analogously to one-bit quantization. Using a selection algorithm that also partially sorts the data speeds up this computation, since less data needs to be examined. As with one-bit quantization, we apply adaptive quantization column-wise, which we find to result in higher accuracies. The two means for reconstruction are sent for each column, and a special

[1] This is an approximation that actually sends a proportion π of each of the non-negative and negative updates and does not account for the relative proportion of these two classes. We do this for simplicity and find it works well.

我们开发了自适应量化，以解决我们在一比特和阈值量化中观察到的缺陷，并从它们中获得 “ 两个世界的最佳 ”。一比特量化可以实现较好的重构和较低的误差，但比阈值量化计算开销更大，并且不能将数据量减少超过32倍。阈值量化快,但选择一个好的τ是很困难的,特别是对于不同的模型,利用τ重建值是次优的。一个固定的阈值也可能导致错误反馈积累的退化情况，导致传输的数据值很大。

自适应量化使用一个固定的比例,π,这表明梯度的比例处理每个mini-batch后发送更新。第一步是确定积极的和消极的阈值τ+和τ−满足所需的当前mini-batch比例。为此，假设有k个非负的更新，我们因此想发送它们中最大的k/π个。确定 τ +可以通过使用一个选择算法(如quickselect)找到第k / π大的更新。

τ －决定以同样的方式,找到最小的(因此,最大震级)k / π 更新。在应用了错误反馈之后，这样做可以确保我们总是发送固定比例的梯度更新，不管更新的状态如何，同时确保缓慢增长的更新最终会积累和发送。

对于重建值,而不是使用阈值进行阈值量化,我们计算出的值大于 τ +和不到 τ －,类似地一比特量化。由于需要检查的数据较少，因此使用选择算法(也对数据进行部分排序)可以加快计算速度。和一比特量化一样，我们采用自适应量化的列，我们发现这样可以得到更高的精度。这两种重构方法分别为每个列发送，并使用一个特殊的分隔符来区分数据流中的列。

separator is used to differentiate between columns in the data stream.

与阈值重构一样，自适应量化也可以适应增量编码和压缩，这是我们实现和评估的。

给定采样数据，我们对其进行选择以逼近阈值，然后使用部分排序的样本来逼近均值。

Just as with threshold reconstruction, adaptive quantization is amenable to delta coding and compression, which we implement and evaluate. As with one-bit quantization, we can also approximate the parameters we need by randomly sampling the data. Given sampled data, we perform selection on it to approximate the thresholds and then use the partially sorted sample to approximate the means. This results in significant runtime improvements, as selection is performed over a much smaller list.

3.4. Allreduce

一种常见的方法，如分布式的方法tensorflow -使用参数服务器聚合来自工作节点的所有更新。

The core collective communication operation required to facilitate data-parallel training is allreduce: the gradient updates from all models are summed and that result is returned to every model. For best performance, care must be taken to ensure the allreduce is efficient, especially in a bandwidth-dominated regime, or scalability will suffer. A common approach—one that is taken by e.g. distributed TensorFlow—is to use parameter servers that aggregate all updates from worker nodes. As this centralizes updates, we are concerned about its viability at large scales and decided not to pursue it. Indeed, when Iandola et al. [10] compared parameter servers and reduction trees, they found that the latter were superior even with only four worker nodes.

Since LBANN's underlying communication layer already uses MPI, another alternative would be to use the built-in `MPI_Allreduce` routine, which has been highly optimized for HPC environments. Unfortunately, for practical reasons, this does not work well with our quantized data. We would need to define custom reduction operations to unquantize the data, perform the sum, and requantize it. While this is not difficult in principle, doing unquantization requires the reconstruction values that we pack into our transmitted data. MPI permits the runtime to call reduction operations on arbitrary chunks of data as performance dictates, which makes it difficult to actually implement the reduction operations. In the case of threshold or adaptive quantization, the output may be larger than the input, a further complication. Lastly, in practice, `MPI_Allreduce` falls back to a simple recursive-doubling algorithm that has a suboptimal $\mathcal{O}(n \log p\beta)$ bandwidth term (β is the transfer time per byte for the network) when given user-defined operations.

To overcome this, we implement our own allreduce operation on top of primitive MPI non-blocking send and receive calls. Our allreduce consists of two steps, a pairwise exchange-based reduce-scatter followed by a ring-based allgather as described in [18], and recommended for large messages. This results in a $\mathcal{O}(\frac{p-1}{p}n\beta)$ bandwidth term, which is superior to recursive-doubling even at small numbers of processors, and ensures that a portion of the communication is nearest-neighbor. While this increases the latency term, we think it less important in our bandwidth-dominated regime.

Our data-parallel communication thus goes as follows: gradient updates are quantized and then split up and scattered. These slices are then unquantized, summed, and the result is quantized again. In the case of threshold quantization, this uses the same τ parameter; adaptive quantization uses the same π. The quantized reductions are then distributed using the allgather, and finally every model unquantizes the results.

4. Evaluation

Here we evaluate one-bit, threshold, and our adaptive quantization in terms of both computational performance and the accuracy of the models they generate. We also examine variants of these involving compression and sampling. Our baseline is to do no quantization and simply use `MPI_Allreduce` on the raw gradient updates.

All tests were run on the Catalyst cluster at Lawrence Livermore National Laboratory [19]. Catalyst consists of 324 nodes, each with two Intel Xeon E5-2695 v2 twelve-core CPUs, 128 GB of RAM, 800 GB of NVRAM, and dual Infiniband QDR network interfaces.

We use the MNIST handwritten digit dataset [20] for some evaluations. MNIST consists of 60,000 28×28 pixel training images containing a single numerical digit and an additional 10,000 test examples. Our test DNN consists of three 4096-neuron fully-connected layers followed by a ten-unit softmax layer for classification. Layers used reLU activations [21]. Training used AdaGrad with an initial learning rate of 0.005. This model was distributed using four-way data-parallelism (four model replicas) and 48-way model-parallelism. The total of 192 processes were distributed 12 to each Catalyst node, using a total of 16 nodes. The underlying BLAS routines automatically make use of the additional cores, and this ratio of processes to cores performs well empirically. Each model replica processed 10 images in each local mini-batch, for a total mini-batch size of 40, and used an identical initialization.[2]

4.1. Model versus data parallelism

A brief but important point to consider is when data parallelism is superior to model parallelism. To test this, we compare the time to complete an epoch with our four-way data-parallel model versus a single large model using 192-way model parallelism and a mini-batch of size 40. This avoids all data-parallel communication, but distributes the layer parameter matrices over many more processes. The single large model completes an epoch in an average of 3561 s; the data-parallel model in 937 s, 3.8 times faster.

Beyond a certain point, the communication overhead of doing distributed matrix operations begins to dominate,

[2] We did not spend much time tuning hyperparameters on this network for maximal accuracy with the different quantization routines. As MNIST is a small dataset, we use only 16 nodes in most of our evaluations; larger datasets would merit larger scales. Future work extending this paper will look at larger datasets and use more highly-tuned neural networks.

at which point gains from model parallelism decrease and eventually become counter-productive. This is where data parallelism can step in to allow continued scalability by doing separate matrix operations on fewer processors to maintain higher efficiency.

4.2. Allreduce benchmark

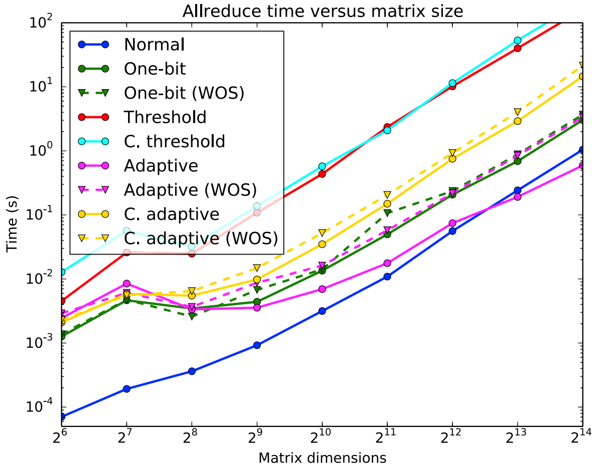

Figure 2. Results of different allreduce implementations on uniformly random matrices with 128 nodes. Each matrix is square and the x-axis gives its height/width. "WOS" variants were run without our sampling optimization and variants prefixed with "C." use compression. Threshold used $\tau = 3.875$ (chosen to leave $1/32$ of the data); adaptive used $\pi = 64$.

We next examine the performance of our allreduce implementations on a synthetic benchmark, using `MPI_Allreduce` (the "normal" variant) as a baseline. We generate square matrices of of size 64×64 to 16384×16384, filled with uniformly random 32-bit floats centered on 0. An allreduce is run for twenty trials, and the average time taken is reported. We use 128 nodes, each with a separate matrix, and two processes on each node. This models the communication patterns present in data-parallel training with 128-way data parallelism and 2-way model parallelism. The matrices should be thought of as a proxy for the gradient updates that are allreduced after each mini-batch. For example, the 16384×16384 case corresponds to a layer with 16384 neurons and roughly 268 million parameters, requiring 1 GiB of memory to store. Note this is a rather large layer compared to our model for MNIST, but this benchmark indicates our scaling trend for the larger models applicable to larger datasets. One-bit quantization typically does a partial AdaGrad step during its allreduce, which none of our other algorithms do. To make the results more fair, we disable this step for the purposes of this evaluation. Our results are plotted in Figure 2.

For small matrix sizes, `MPI_Allreduce` is clearly superior in every case. In these cases, the bandwidth requirements are small enough that the quantization is counter-productive. At larger sizes, the approaches become more differentiated. Threshold quantization performs the worst due to a build-up of the error feedback leading to a significant portion of the data being sent with later allreduces. However, if we examine just the initial iterations before error feedback has built up, its runtime is competitive at larger scales.

Here, compression is actively harmful to performance due to the increased computation required (as noted by [14]); we believe that with further optimization it may prove useful. Sampling is crucial for good performance in our adaptive quantization. Not only does it reduce the amount of data examined, the selection problem is performed on a smaller set, which gives a significant speedup. One-bit quantization receives only a slight benefit from sampling. `MPI_Allreduce` remains competitive until the largest matrix sizes, at which point our adaptive quantization outperforms it due to decreased data volume. For the largest matrices, adaptive quantization is 1.76 times faster than `MPI_Allreduce`.

One-bit and threshold quantization do not appear to be competitive with `MPI_Allreduce` at any scale; as far as we are aware, they have not been compared with MPI prior to this work.

Based on these performance results, we do not consider compression further, nor do we use variants without sampling.

4.3. Mini-batch performance

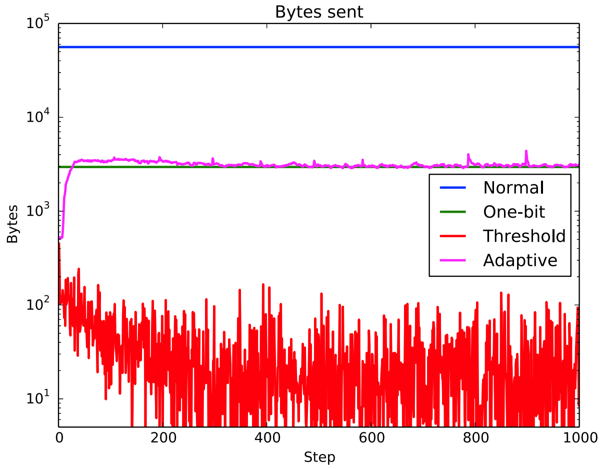

Figure 3. Volume of data sent each mini-batch during allreduces for the third fully-connected layer (other FC layers are similar). The x-axis refers to the particular mini-batch. Normal and one-bit send the expected amount of data, and adaptive quantization closely follows one-bit in data volume. Threshold quantization sends very little data, resulting in poor learning.

Communication is done after each mini-batch completes, so examining metrics for the quantization algorithms at mini-batch granularity provides insight into fine details of their performance and enables us to validate expectations. These were measured during the training of our MNIST model described at the beginning of Section 4.

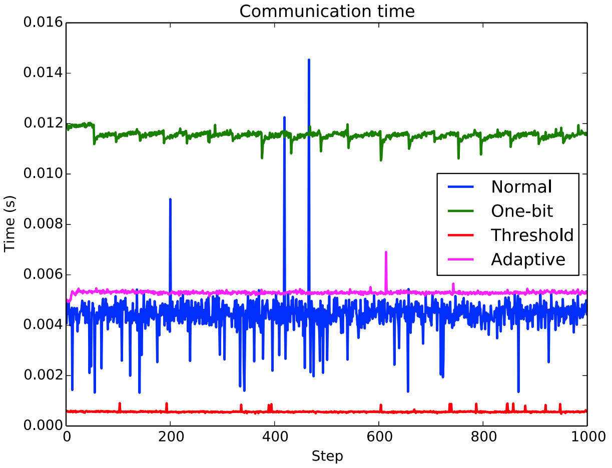

Figure 4. Time spent performing the allreduce and quantization (if any) during each mini-batch for the third fully-connected layer (other FC layers are similar). The x-axis refers to the particular mini-batch. These times are as our allreduce benchmark leads us to expect, except for threshold, which is low due to transmitting almost no data.

We first investigate the amount of data sent in each allreduce, plotted for a representative fully-connected layer in Figure 3. The results for no quantization and one-bit quantization are as expected: either the full update matrix is sent, or the quantity of data is reduced by a factor of 32 (plus a small constant additional amount of meta-data to send the reconstruction values). The adaptive quantization is more interesting. With $\pi = 64$, we should expect a data reduction similar to that of one-bit quantization, but since it uses sampling to approximate the thresholds to exclude updates, it does not exactly achieve a 32-times reduction. Our sampling is a good enough approximation, however, for the data volume to never be too great or too small, as intended. Threshold quantization sends very little data, even with $\tau = 0.001$, and the volume it does send is very erratic. This results in test accuracies that are noticably lower than other methods (see Section 4.4), as each model is essentially only learning from its local mini-batch data.

Figure 4 looks at the time taken to perform the allreduce on the same layer. These results are in line with what our allreduce benchmark (see Section 4.2) would predict, given the size of each process's local matrix, with the exception of threshold quantization. As it sends very little data, threshold quantization achieves the lowest communication overhead by a significant margin. The normal `MPI_Allreduce` is the next fastest, due to the small local matrix sizes making quantization potentially counter-productive. It is also moderately noisy, which we believe to be due to network effects on the shared cluster these tests were run on. Adaptive quantization follows slightly behind `MPI_Allreduce`, and additionally performs quite consistently. One-bit quantization performs noticably slower than adaptive quantization, but note that this is not an entirely fair comparison: a partial AdaGrad step is performed in this allreduce, which the others do not do. This saves some computation time later; however, even excluding the time to do AdaGrad, one-bit quantization is still slower than adaptive quantization. We do not disable this step, because we used this same run to evaluate accuracy in the next section. The slight periodicity present in the one-bit quantization's timing is related to data-structures being reset after each epoch and does not have a significant impact on overall epoch time.

4.4. Accuracy

	Test accuracy (%)
Normal	98.51
One-bit	98.49
Threshold	98.12
Adaptive	**98.53**

TABLE 1. TEST ACCURACY ON MNIST AFTER 20 EPOCHS OF DATA-PARALLEL TRAINING.

It is important that quantization not degrade final test accuracy overly much, or any performance gains will be mooted by poor models. Table 1 reports test accuracies for our network on the MNIST handwritten digit dataset after twenty training epochs.

Normal (unquantized updates) and one-bit and adaptive quantization all achieve comparable accuracies. We do not attribute the small differences between their accuracies to the quantization algorithms but rather to the particulars of the initialization. We expect that all three approaches will be able to get comparable accuracies with appropriately-tuned hyperparameters. The original work on one-bit quantization reported that in some instances, quantization benefitted AdaGrad and resulted in a small accuracy gain, and in others, quantization produced a small accuracy loss. Here, we see neither effect, though this may be due to the different natures of the data and networks as one-bit quantization was originally applied to speech DNNs.

Threshold quantization performs noticably worse than the other variants. It shares very little data between model replicas, and so each model replica essentially learns only from its local subset of data. To test this, we ran our model with no data-parallel communication, so each model learns solely from its local mini-batches. The best model achieves 98.09% accuracy, indicating that threshold quantization offers little gain. The trouble here is due to the difficulty in tuning the τ threshold parameter so that updates are still sent, but there is meaningful data reduction. [14] used $\tau = 4.0$, which results in no updates ever being sent between our models. Our use of $\tau = 0.001$ still leads to little data being transmitted. Indeed, this demonstrates one of the advantages of our adaptive quantization: the threshold is chosen automatically to achieve a given data reduction.

4.5. Data-parallel performance

We examine the scalability of our data-parallelism as we increase the number of model replicas while keeping the per-replica mini-batch size constant. While this is not

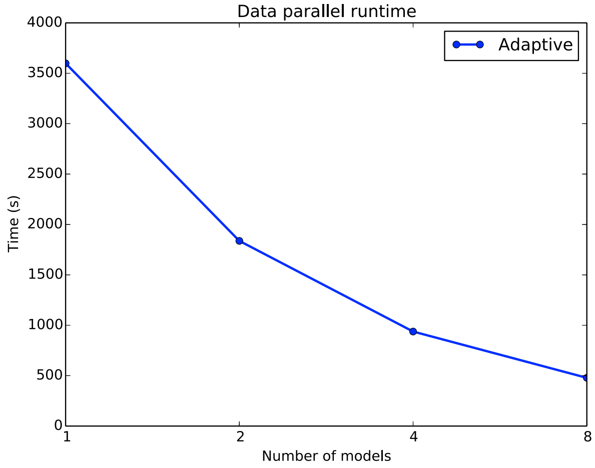

Figure 5. Data-parallel scaling of adaptive quantization with $\pi = 64$ and a model with three 4096-neuron fully-connected layers. We vary the number of model replicas from 1 to 8 with each replica doing a mini-batch of size 10. Replicas use 48 processes across four nodes. The y-axis reports the average time to complete an epoch over the entire training set. Other approaches yield similar scalability.

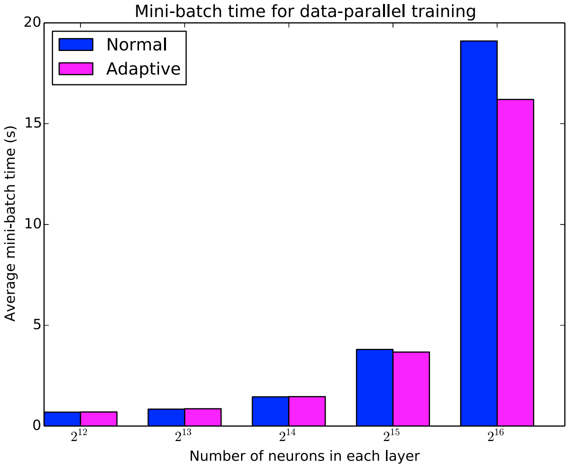

Figure 6. Average mini-batch time for training a model with the fully-connected layers of increasing size using four-way data parallelism and 48-way model parallelism on 16 nodes. Adaptive quantization is 1.18 times faster than `MPI_Allreduce`, despite the large amount of model parallelism.

necessarily the most realistic approach to training a DNN in practice, it provides a good benchmark for understanding the performance of our quantization at larger scales. We train the DNN architecture described in the beginning of Section 4, but vary the number of model replicas. Each model replica still uses 48 processes and four nodes. Data-parallel communication was quantized using $\pi = 64$. The average time taken per epoch is reported in Figure 5. We report only on adaptive quantization as the other approaches scale similarly in this test, just with different constants.

The scaling trend we see here is excellent and we achieve a 7.5-times speedup when using eight model replicas. This further validates the viability of large-scale data-parallel training.

Lastly, we conduct a benchmark similar to our allreduce benchmark in Section 4.2, but with a full DNN. We begin with our original DNN, consisting of three 4096-neuron hidden layers using four-way data parallelism and 48-way model parallelism, and increase the size of each layer, up to 65,536 neurons.[3] The average mini-batch time while training on the MNIST dataset is recorded. This allows us to evaluate our data-parallel communication in a more realistic context, albeit still at smaller scales due to the large quantity of model parallelism lowering the quantity of data each process holds. Our results are in Figure 6. For smaller layers both approaches perform comparably, with `MPI_Allreduce` being slightly faster. For larger layers, adaptive quantization overtakes `MPI_Allreduce`, running 1.18 times faster in the largest case.

This confirms the cross-over point seen in our allreduce benchmark in a more realistic scenario: adaptive quantization begins to outperform `MPI_Allreduce` once the data volume on each process becomes sufficiently large. Note that while this is a large network, prior work with LBANN has looked at autoencoders with 400,000 neurons [1] in a layer. Further, there is a trend toward using very deep models with many layers, which will have similar total data transfer requirements. Our results indicate that our adaptive quantization will also be appropriate for scaling such models.

5. Future Work

There are several avenues for future improvements and investigation. Firstly, on a more practical side, our implementations have room for optimization. Currently all our quantization implementations are single-threaded, but much of the computation is embarrassingly parallel and should achieve good parallel speedup. This will enable better scalability for large layers, and may improve quantization's competitiveness for moderately-sized layers. It is unlikely, however, that quantization will ever be worth while for small layers. Similarly, improvements in compression performance may have implications for the quantization of large layers. Currently, compression is a net loss due to the additional computation required; if this could be reduced sufficiently, the data volume reduction could begin to have an impact.

We plan to extend our quantization to apply beyond fully-connected layers, e.g. to convolutional and pooling layer types once they are fully integrated and tested. We also plan to study larger datasets such as ImageNet, for which convolution has proved extremely successful.

Lastly, we focused almost exclusely on data parallelism here. For best scalability, we need to combine data-parallel

[3] This is not a realistic model for MNIST classification and we use it primarily as an example to demonstrate scalability. Future work will look at larger datasets where such layers are more appropriate.

and model-parallel approaches, and determine what the appropriate tradeoff is between the two.

6. Conclusions

We have shown that data-parallel DNN training scales well on HPC-class systems. We compared one-bit and threshold quantization with built-in MPI operations, and find that `MPI_Allreduce` out-performs them at all scales despite their reduced bandwidth requirements. Our new adaptive quantization algorithm is faster than `MPI_Allreduce` for large layers, and outperforms existing quantization approaches beginning at moderately-sized layers. In evaluations on the MNIST dataset, one-bit and adaptive quantization both achieve similar accuracies to not applying any quantization. Threshold quantization, however, has proven hard to tune in practice.

Taking advantage of HPC resources to train deep neural networks requires exposing extensive parallelism. While further work is required if we hope to scale to the largest supercomputers, quantizing data-parallel communication provides a promising approach to handling large layers.

7. Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-700919). Funding provided by LDRD 16-ERD-039. Experiments were performed at the Livermore Computing facility resources.

References

[1] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "LBANN: Livermore big artificial neural network HPC toolkit," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 2015, p. 5.

[2] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *arXiv preprint arXiv:1003.0358*, 2010.

[3] I. G. Y. Bengio and A. Courville, "Deep learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[4] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "On parallelizability of stochastic gradient descent for speech dnns," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 235–239.

[5] J. Poulson, B. Marker, R. A. Van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 39, no. 2, p. 13, 2013.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[7] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[8] S. Lee, S. Purushwalkam, M. Cogswell, D. J. Crandall, and D. Batra, "Why M heads are better than one: Training a diverse ensemble of deep networks," *arXiv*, 2015. [Online]. Available: http://arxiv.org/abs/1511.06314

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[10] F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer, "FireCaffe: Near-linear acceleration of deep neural network training on compute clusters," *arXiv preprint arXiv:1511.00175*, 2015.

[11] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1337–1345.

[12] I.-H. Chung, T. N. Sainath, B. Ramabhadran, M. Pichen, J. Gunnels, V. Austel, U. Chauhari, and B. Kingsbury, "Parallel deep neural network training for big data on blue gene/q," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 745–753.

[13] 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, pp. 1058–1062.

[14] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *INTERSPEECH*, vol. 7, 2015, p. 10.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[16] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, 1997.

[17] R. Rice and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Transactions on Communication Technology*, vol. 19, no. 6, pp. 889–897, 1971.

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

[19] Lawrence Livermore National Laboratory, "Catalyst," http://computation.llnl.gov/computers/catalyst, 2016.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.