Parallel Deep Neural Network Training for LVCSR Tasks using Blue Gene/Q

Tara N. Sainath, I-hsin Chung, Bhuvana Ramabhadran, Michael Picheny,
John Gunnels, Brian Kingsbury, George Saon, Vernon Austel, Upendra Chaudhari

IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A
{tsainath, ihchung, bhuvana, picheny, gunnels, bedk, gsaon, austel, uvc}@us.ibm.com

Abstract

While Deep Neural Networks (DNNs) have achieved tremendous success for LVCSR tasks, training these networks is slow. To date, the most common approach to train DNNs is via stochastic gradient descent (SGD), serially on a single GPU machine. Serial training, coupled with the large number of training parameters and speech data set sizes, makes DNN training very slow for LVCSR tasks. While 2nd order, data-parallel methods have also been explored, these methods are not always faster on CPU clusters due to the large communication cost between processors. In this work, we explore using a specialized hardware/software approach, utilizing a Blue Gene/Q (BG/Q) system, which has thousands of processors and excellent interprocessor communication. We explore using the 2nd order Hessian-free (HF) algorithm for DNN training with BG/Q, for both cross-entropy and sequence training of DNNs. Results on three LVCSR tasks indicate that using HF with BG/Q offers up to an 11x speedup, as well as an improved word error rate (WER), compared to SGD on a GPU.

1. Introduction

Deep Neural Networks (DNNs) have become a popular acoustic modeling technique over the last few years [1], showing significant gains over Gaussian Mixture Model/Hidden Markov Model (GMM/HMM) systems on many small and large vocabulary tasks. The development of pre-training algorithms [2] and better forms of random initialization [3], as well as the availability of faster computers, has made it possible to train deeper networks than before, and in practice these deep networks have achieved excellent performance [4, 5, 6].

However, one drawback of DNNs is that training remains very slow, particularly for large vocabulary continuous speech recognition (LVCSR) tasks. This can be attributed to a variety of reasons. First, models for real-world speech tasks are trained on hundreds to thousands of hours of data, which amounts to many millions of training examples. Second, roughly 10-50 million DNN parameters are used for speech tasks [4, 5], which is much larger than the number of parameters used with GMMs on the same tasks. Third, to date the most popular methodology to train DNNs is with stochastic gradient descent (SGD), serially on one machine, typically using GPUs.

Optimization techniques that parallelize training across multiple machines have also been explored for DNN training [7, 8, 9, 10]. However, these methods are not always faster than training DNNs via SGD on GPUs. For example, as shown in [11], parallelization of dense networks can actually be slower than serial SGD training, typically because of communication costs in passing models and gradients as well as the need to run more training iterations than serial SGD. While asynchronous SGD methods have been explored in an attempt to address some of the communication bottlenecks [9], it is unclear if SGD methods are as good as 2nd-order methods when using a sequence-level objective function [12] for DNN training [13]. In short, parallelization methods for DNNs have not become the standard training approach, and training DNNs via SGD on GPUs is still the most popular technique.

In this paper, we explore the use of a specialized hardware/software architecture, which minimizes communication bottlenecks between cores, in order to improve the DNN training speed. Specifically, we utilize the IBM Blue Gene/Q (BG/Q) [14], which is designed to deliver ultra-scale performance with a standard programming environment. It is currently one of most efficiently scalable systems for computationally intensive applications, and has enabled science to address a wide range of complex problems [15, 16, 17]. Given its impressive performance in the area of analytics applications, Blue Gene/Q is a promising platform for parallelizing big-data applications, such as LVCSR tasks, across thousands of cores.

The objective of this paper is to explore speeding up DNN training using 2nd-order optimization, which easily lends itself to parallelization, with BG/Q. We specifically apply BG/Q to the Hessian-free (HF) algorithm [7], which has shown state-of-the-art performance with DNNs on many LVCSR tasks [8]. Since DNN training for speech consists of both cross-entropy (CE) and sequence training (ST) stages [12], we investigate speeding up both stages of training using BG/Q.

To explore the robustness of HF with BG/Q, we investigate the behavior on 3 different LVCSR tasks, namely a 50-hr English Broadcast News (BN) task, a 430-hr BN task, and a 300-hr Switchboard task. For CE training, HF BG/Q offers a 3-5x speedup over SGD with GPUs, with no degradation in word error rate (WER). In addition, for ST, HF BG/Q provides a 2-11x speedup over SGD with GPUs, with a slightly better WER.

The rest of this paper is organized as follows. Section 2 reviews the HF algorithm used for DNN training in this paper, while Section 3 describes the BG/Q architecture. Section 4 describes various improvements made to allow HF to work with BG/Q. A description of the corpora and DNN architecture used for experiments is discussed in Section 5, while results comparing HF with BG/Q to SGD GPU is presented in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2. Parallel Deep Neural Network Training

Before describing how we utilized the BG/Q architecture to implement the Hessian-free (HF) algorithm, we briefly summarize HF for DNN training, as described in [7]. Let $\boldsymbol{\theta}$ denote the network parameters, $\mathcal{L}(\boldsymbol{\theta})$ denote a loss function, $\nabla\mathcal{L}(\boldsymbol{\theta})$ denote the gradient of the loss with respect to the parameters, $\mathbf{d}$ denote a search direction, and $\mathbf{B}(\boldsymbol{\theta})$ denote a matrix characterizing the curvature of the loss around $\boldsymbol{\theta}$ (i.e., a Hessian approx-

imation). The central idea in HF optimization is to iteratively form a quadratic approximation to the loss,

$$\mathcal{L}(\boldsymbol{\theta}+\mathbf{d}) \approx \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\mathbf{B}(\boldsymbol{\theta})\mathbf{d} \qquad (1)$$

and to minimize this approximation using conjugate gradient (CG), which accesses the curvature matrix only implicitly through matrix-vector products of the form $\mathbf{B}(\boldsymbol{\theta})\mathbf{d}$. Such products can be computed efficiently for neural networks [18]. In the HF algorithm, the CG search is truncated, based upon the relative improvement in the approximate loss. The curvature matrix is chosen to be the Gauss-Newton matrix $\mathbf{G}(\boldsymbol{\theta})$ [19], which is positive semi-definite. To avoid breakdown of CG, a positive definite approximation can be enforced by shifting the matrix using an additional damping term: $\mathbf{B}(\boldsymbol{\theta}) = \mathbf{G}(\boldsymbol{\theta})+\lambda\mathbf{I}$, where λ is set via the Levenberg-Marquardt algorithm. Our implementation of HF optimization, which is illustrated as pseudo-code in Algorithm 1, closely follows [7]. Gradients are computed over all the training data. Gauss-Newton matrix-vector products are computed over a sample (about 1-3% of the training data) that is taken each time `CG-Minimize` in Algorithm 1 is called. `CG-Minimize` calls CG multiple times to get a series of conjugate search directions, and then uses a backtracking line search to find the best search direction p_i.

Algorithm 1 Hessian-free optimization (after [7]).

for i=1, 2, ... **do**
  $\mathbf{g_i} \leftarrow \nabla\mathcal{L}(\boldsymbol{\theta_i})$
  Adjust damping factor λ via Levenberg-Marquardt;
  Define $\mathbf{B}(\boldsymbol{\theta_i}) = \mathbf{G}(\boldsymbol{\theta_i})\mathbf{d} + \lambda\mathbf{d}$
  $\mathbf{p}_i \leftarrow$ CG-MINIMIZE$(\mathbf{B}(\boldsymbol{\theta_i}), -\mathbf{g_i})$
  $\boldsymbol{\theta_{i+1}} \leftarrow \boldsymbol{\theta_i} + \mathbf{p}_i$
end for

To perform distributed computation, we use a master/worker architecture in which each worker performs data-parallel computation of gradients and curvature information across a cluster, while the master implements the HF optimization and coordinates the activity of the workers.

3. Architecture

The IBM BG/Q system [14] is the third generation of high-performance computer in the BG series. BG/Q's processor [20] is a system-on-chip multi-core processor with 16 A2 PowerPC 64-bit cores. Each A2 core runs at 1.6 GHz, is 4-way multi-threaded, and is capable of executing instructions in-order in two pipelines. One pipeline executes all integer control and memory access instructions while the other pipeline executes all floating point arithmetic instructions. The floating point unit is a 4-wide SIMD double precision unit known as the QPX. The QPX unit is capable of delivering up to four fused multiply-add (FMA) results per processor clock.

Each core has a 16K-byte private level 1 (L1) cache and 2K-byte prefetching buffer (L1P). All the cores on the same processor share a 32M-byte level 2 (L2) cache. The L2 cache provides versioning support which is used to implement both hardware transactional memory and thread level speculation. The compute nodes are connected in a 5-D torus network with a total network bandwidth of 44 GB/s per node.

The BG system architecture includes an efficient software stack and operating system. The lightweight compute node kernel (CNK) is designed to be efficient and therefore neither virtual memory nor multi-tasking is supported. The file I/O is offloaded to the dedicated I/O nodes via network communication.

4. HF Training for BG/Q

In this section, we describe numerous improvements made to allow the HF algorithm to work well with BG/Q.

4.1. Matrix Multiplication

Optimizing matrix multiplication for an in-order, multi-threaded, dual issue, multicore architecture such as BG/Q is a straightforward, if somewhat intricate, procedure. The goal is to execute a SIMD FMA every cycle on every core involved in the computation. This requires that the matrix multiplication routine is carefully scheduled, threaded, blocked, and synced at the right level of granularity, contains an appropriate instruction mix, and exploits the coordination available between both threads and cores.

While a strategy specific to a narrow range of matrix dimensions may improve performance for those cases, we designed our code to be efficient for a wide range of matrix sizes and alignments and, with the exception of the inner kernel, reasonably generic. The single-threaded, innermost kernel is written in assembly language, exploits what we refer to as co-operative prefetching [21] (the code on each thread is slightly different), has an instruction mix which maximizes the dual-issue abilities of the multithreaded core, avoids all dependency stalls in the steady state, and reduces read bandwidth requirements to 4 bytes/cycle by targeting (8x8) blocks of the register-resident C matrix. The four-way-threaded code for a single core orchestrates the coordinating threads so as to keep them in sync, allowing us to leverage the small (4KB per thread) L1D cache, and cutting the read bandwidth requirements in half (to 2 bytes/cycle). At the node level, while the number of cores working together (i.e. the number of cores in an MPI rank) in a matrix multiplication might vary from one to sixteen, the code was written so as to have the cores in a rank reduce cumulative load on the shared L2 cache, such that this demand increases as the square root of the number of cores involved, as opposed to a linear increase.

This set of techniques, used in concert, allows matrix multiplication to exceed 95% of theoretical peak on a BG/Q node.

4.2. Communication

In order to improve communication performance, we replaced the socket communication, which was used in the past for the HF algorithm when run on Intel CPU machines [8], with MPI [22]. Socket communication requires that the programmer take care of all details of communication which includes channel (communication parties) set up, data packaging and routing. MPI provides much more functionality when managing many nodes in a large scale application and is portable, based on the MPI standard. The BG/Q MPI communication library is heavily optimized, as MPI has wide support in high performance computing. The optimized MPI on BG/Q is implemented on top of PAMI, a high-level active-message interface used to support many programming models.

4.3. Load Balancing

One key factor affecting scalability is load balancing, as distributing data evenly across worker nodes helps the program proceed in a synchronized pace to achieve better performance.

We distributed the data so as to minimize the run-time variation between workers. In speech tasks, the training data is comprised of several spoken utterances from thousands of speakers. These utterances in the training set are not all of the same length, so we preprocess the data by sorting according to utterance length and employ a distribution scheme for the utterances in which each worker receives the same amount of data.

4.4. Caching of Lattices

One of the bottlenecks of ST is the loading of lattices [8]. When HF ST was performed in [8], the lattices were loaded from disk every time an utterance was processed. All lattices corresponding to utterances that a specific worker processed could not be cached ahead of time due the large number of utterances, small number of machines (64), memory requirements (32GB), and size of the lattices. When using BG/Q with thousands of cores, we can now cache the lattices on each machine ahead of time, as each worker sees many fewer utterances.

5. Experimental Details

5.1. Tasks and DNN Archictecture

In this section, we describe the corpora and DNN architecture used for experiments in more detail.

5.1.1. 50 hour Broadcast News

Initial experiments are conducted on a 50 hour English Broadcast News (BN) transcription task [12] and results are reported on 100 speakers in the EARS `dev04f` set[1]. An LVCSR recipe described in [23] is used to create a set of feature-space speaker-adapted (FSA) features, using vocal-tract length normalization (VTLN) and feature-space maximum likelihood linear regression (fMLLR). All DNNs use FSA features as input, with a context of 9 frames around the current frame. Following [5], we use a 5-layer DNN with 1,024 hidden units per layer and a sixth softmax layer with 3,000 output targets. All DNNs are discriminatively pre-trained [4], followed by CE and ST.

5.1.2. 430 Hour Broadcast News

Further experiments are conducted on a larger 430-hr English BN task, and results are reported on the full DARPA EARS `dev04f` set. DNNs use FSA input features, with a context of 9 frames around the current frame. The architecture consists of 5 hidden layers with 1,024 hidden units per layer and a sixth softmax layer with 5,999 output targets.

5.1.3. 300 Hour Switchboard

Finally, we conduct experiments on 300 hours of conversational English telephony data from the Switchboard (SWB) corpus. The DNN uses FSA features as input, with a context of 11 frames around the current frame. Similar to [4], the architecture consists of 6 hidden layers with 2,048 hidden units per layer and a seventh softmax layer with 9,300 output targets.

5.2. Details of Other Algorithms

We compare the HF BG/Q method to mini-batch SGD run on a GPU. The GPU used for these experiments is a single Tesla K10 GPU. Weight updates are made after processing each mini-batch. For CE training, the mini-batch is a random collection of 256 frames, while for ST each mini-batch is a randomly chosen utterance. Following [5], during CE and ST, after one pass through the data, loss is measured on a held-out set and the learning rate is reduced by a factor of 2 if the held-out loss has not improved sufficiently over the previous iteration. Training stops after the WER has converged on a held-out set.

We also compare HF BG/Q to HF running on a cluster of Intel CPUs. Each CPU processor is an 8 core Intel Xeon X5570 running at 2.93GHz CPU. Matrix/vector operations for DNN training are multi-threaded using Intel MKL-BLAS. All experiments use 64 machines. The HF CPU implementation uses the same parameters as the HF BG/Q implementation, and is described in more detail in Section 2. Training stops once the WER has converged on a held-out set. We have found the HF algorithm to be sensitive to the choice of initial weights during CE training, and therefore run a few iterations of SGD before switching over to HF. All HF CE timing numbers reflect the timing with warm-start SGD+BG/Q HF. However, because ST starts from CE-trained weights, which are in a decent space, we don't need to do this warm start SGD.

6. Results

6.1. Tuning HF for BG/Q

In BG/Q, parallelization of DNN training occurs at multiple levels, namely the number of workers which perform data-parallel computations, as well as the number of threads/cores used to parallelize the matrix multiplication. Given a BG/Q rack, which has up to 1,024 nodes, 16 cores/node and 4 threads/core, we need to find the optimal balance of these parameters for training DNNs. First, on a small 50-hour BN task, we explore training time with different configurations of workers (i.e., MPI ranks), ranks/node and threads/rank.

A plot of the time taken for different configurations is shown in Figure 1a. BG/Q has a maximum capacity of 64 threads/node. Any product of rank/node and threads/rank which does not exceed 64 gives an acceptable configuration. The plot shows that if we use 1,024 ranks, timing continues to improve as we utilize more threads/node, up to a maximum of 64. This is no surprise as increased threads/node allow for more efficient parallelization of the matrix multiplication.

If the number of ranks is increased past 1,024, thereby increasing the number of workers, we must decrease the number of threads. Thus, there is a tradeoff between parallelization done at the data-level (thus affecting ranks) and parallelization at the thread level (thus affecting threads/rank), that we must optimize. The plot shows that further speedups can be achieved by using more ranks, either through a configuration of 2,048-2-32, or 4,096-4-16, and both have similar timings.

Next, we tuned BG/Q for a larger 430-hr task. For this task, we used 2 BG/Q racks, which has 2,048 nodes now. By using 2 racks, Figure 1b shows an additional speedup of 22% over just 1 rack, with the optimal configuration at 8192-4-16.

6.2. Timing Performance

In this section, we compare HF with BG/Q to HF run on a CPU cluster and SGD run on a GPU, both for CE and ST.

6.2.1. 50 Hour Broadcast News

Table 1 shows results on the 50-hr BN task. For the HF CPU experiments, 64 machines were exclusively reserved for HF training to get reliable training time estimates. Notice that for CE

[1] This set has one speaker removed

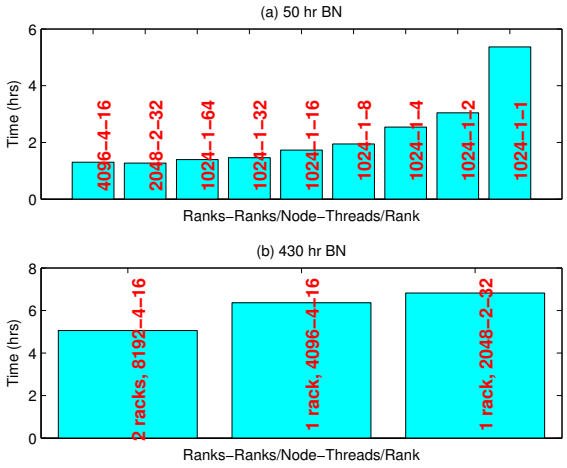

Figure 1: Execution time for different configurations

training, the WER of all three algorithms is the same. SGD using GPUs is faster then HF with CPUs, both for CE and ST, highlighting the communication bottleneck issues with parallel CPU optimization methods. However, by utilizing a specialized architecture like BG/Q with HF, we can achieve a 5x speedup over the GPU for CE.

For ST, BG/Q offers a 1.7x speedup over GPUs, though the WER of SGD GPU is worse than that of HF. As ST utilizes sequential lattice information, this requires that mini-batches are now utterances, rather than frames. For CE, we have seen that we take a hit in WER when mini-batches are chosen to be utterances, rather than a random collection of frames [24]. However, since HF uses large batches of data for gradient and CG computations, it not negatively impacted by utterance randomization. Increasing the batch size for GPUs to alleviate this frame vs. utterance issue would increase training time, and was also shown in [13] not to help. Thus, HF with BG/Q provides the fastest training method and best WER for CE and ST.

Hardware	Cross-Entropy		Sequence Training	
	Time (hrs)	WER	Time (hrs)	WER
HF CPU	9	17.8	18.7	14.7
SGD GPU	7	17.8	6.6	16.2
HF BG/Q	**1.3**	17.8	**3.9**	14.7
Speedups	**5**		**1.7**	

Table 1: Training Time, 50 hr Broadcast News

6.2.2. *430 Hour Broadcast News*

Next, we explore how BG/Q HF scales to a larger 430 hr BN task, now run with 2 racks. Since SGD GPU is faster the HF CPU for CE, and both achieve similar WER, we only report numbers for ST with HF CPU. Due to the larger data set, it was not possible to exclusively reserve machines for the HF CPU experiments, so we report WER numbers only.

Table 2 indicates that HF BG/Q provides a 3x speedup over SGD CE and 11.6x speedup over SGD ST, while matching the WER of the HF CPU. Speedup improvements for CE with BG/Q are less than ST because of the warm-start SGD required for HF CE. Again notice the degradation in WER with SGD GPU for ST. Also notice that we get better speedups for ST with 430-hr BN (11.6x) compared to 50-hr BN (1.7x). This is due not only to using 2 racks for 430-hr BN, but also because the SGD GPU algorithm required more iterations to converge for 430-hr BN as opposed to 50-hr BN.

Hardware	Cross-Entropy		Sequence Training	
	Time (hrs)	WER	Time (hrs)	WER
HF CPU	-	-	n/a	15.1
SGD GPU	77.9	16.5	42.1	15.8
HF BG/Q	**21.7**	16.5	**3.6**	15.1
Speedups	**3**	-	**11.6**	-

Table 2: Training Time, 430 hr Broadcast News

6.2.3. *300 Hour Switchboard*

Finally, to explore the robustness of BG/Q HF across tasks, we explore speedups on the 300-hr SWB task. Again only WER (and not timing results) are reported for HF CPU. Trends are similar to 430-hr BN, where we see speedups of 4x and 10.3x over SGD GPU for CE and ST, while while matching the accuracy of the HF CPU implementation.

Hardware	Cross-Entropy		Sequence Training	
	Time (hrs)	WER	Time (hrs)	WER
HF CPU	-	-	n/a	12.4
SGD GPU	121.5	14.1	47.6	12.7
HF BG/Q	**28.0**	14.1	**4.6**	12.4
Speedups	**4**	-	**10.3**	-

Table 3: Training Time, 300 hr Switchboard

6.3. Further Speedups

Various speedups have been explored for the HF algorithm, including low-rank matrix factorization [25], pre-conditioning for conjugate gradient (CG) and sampling the data used for the gradient and CG steps [26]. In this section, we explore what speedups these ideas have provided on top of the BG/Q architecture. Table 4 show that we get an additional 1.2x and 1.8x speedup on CE and ST respectively, showing how algorithmic speedups show benefits even on top of a specialized BG/Q architecture. Overall, algorithmic speedups + BG/Q provide a 4.2x speedup for CE and 21x for ST compared to SGD on a GPU in Table 2. The small degradation in WER during CE with the speedup ideas is erased during ST [25], as the table also indicates. Larger improvements in speedup can be seen with ST compared to CE, as the low-rank factorization has been shown to constrain the space of search directions and reduce the number of HF iterations during ST [25].

Hardware	Cross-Entropy		Sequence Training	
	Time (hrs)	WER	Time (hrs)	WER
HF BG/Q	21.7	16.5	3.6	15.1
+[25]+[26]	18.4	16.6	2.0	15.1
Speedups	**1.2**	-	**1.8**	-

Table 4: Training Time with Speedups, 430 hr BN

7. Conclusions

With thousands of cores, and minimal communication bottlenecks, BG/Q is an ideal architecture for massively parallel tasks, such as HF training of DNNs. In this paper, we found that across three different LVCSR tasks, HF with BG/Q was 2-11x faster compared to SGD on a GPU. To our knowledge, this is one of the fastest architectures to train DNNs to date.

8. References

[1] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] G. E. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, 2010, pp. 249–256.

[4] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," in *Proc. Interspeech*, 2011.

[5] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A. Mohamed, "Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition," in *Proc. ASRU*, 2011.

[6] N. Jaitly, P. Nguyen, A. W. Senior, and V. Vanhoucke, "Application Of Pretrained Deep Neural Networks To Large Vocabulary Speech Recognition," in *Proc. Interspeech*, 2012.

[7] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. Intl. Conf. on Machine Learning (ICML)*, 2010.

[8] B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization," in *Proc. Interspeech*, 2012.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," in *NIPS*, 2012.

[10] O. Vinyals and D. Povey, "Krylov subspace descent for deep learning," in *Proc. NIPS Workshop on Optimization and Hierarchical Learning*, 2011.

[11] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Pronchnow, and A. Ng, "On Optimization Methods for Deep Learning," in *Proc. ICML*, 2011.

[12] B. Kingsbury, "Lattice-Based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Modeling," in *Proc. ICASSP*, 2009.

[13] G. Saon and H. Soltau, "A Comparison of Two Optimization Techniques for Sequence Training of Deep Neural Networks," in *to appear in Proc. ICASSP*, 2014.

[14] M. Gschwind, "Blue Gene/Q: Design for Sustained Multipetaflop Computing," in *Proceedings of the 26th ACM international conference on Supercomputing*, ser. ICS '12. New York, NY, USA: ACM, 2012.

[15] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A Scalable Simulator for an Architecture for Cognitive Computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[16] J. Doi, "Peta-scale Lattice Quantum Chromodynamics on a Blue Gene/Q Supercomputer," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[17] P. Huang, H. Avron, T. Sainath, V. Sindhwani, and B. Ramabhadran, "Kernel Methods Match Deep Neural Networks on TIMIT: Scalable Learning in High-Dimensional Random Fourier Spaces," in *to appear in Proc. ICASSP*, 2014.

[18] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[19] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Computation*, vol. 14, pp. 1723–1738, 2004.

[20] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Chist, and C. Kim, "The IBM Blue Gene/Q Compute Chip," *IEEE Micro*, vol. 32, no. 2, pp. 48–60, 2012.

[21] J. Gunnels, "Making good enough...better: Addressing the multiple objectives of high-performance parallel software with a mixed global-local worldview," 2012.

[22] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[23] H. Soltau, G. Saon, and B. Kingsbury, "The IBM Attila Speech Recognition Toolkit," in *Proc. SLT*, 2010.

[24] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Improvements in Using Deep Belief Networks for Large Vocabulary Continuous Speech Recognition," IBM T.J. Watson Research Center, Tech. Rep., 2010.

[25] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-Rank Matrix Factorization for Deep Neural Network Training with High-Dimensional Output Targets," in *Proc. ICASSP*, 2013.

[26] T. N. Sainath, L. Horesh, B. Kingsbury, A. Aravkin, and B. Ramabhadran, "Accelerating Hessian-Free Optimization for Deep Neural Networks by Implicit Preconditioning and Sampling," in *Proc. ASRU*, 2013.