

Parallel Deep Neural Network Training for Big Data on Blue Gene/Q

I-Hsin Chung, *Senior Member, IEEE*, Tara N. Sainath, Bhuvana Ramabhadran, *Fellow, IEEE*, Michael Picheny, *Fellow, IEEE*, John Gunnels, *Senior Member, IEEE*, Vernon Austel, Upendra Chauhari, and Brian Kingsbury, *Senior Member, IEEE*

Abstract—Deep Neural Networks (DNNs) have recently been shown to significantly outperform existing machine learning techniques in several pattern recognition tasks. DNNs are the state-of-the-art models used in image recognition, object detection, classification and tracking, and speech and language processing applications. The biggest drawback to DNNs has been the enormous cost in computation and time taken to train the parameters of the networks—often a tenfold increase relative to conventional technologies. Such training time costs can be mitigated by the application of parallel computing algorithms and architectures. However, these algorithms often run into difficulties because of the cost of inter-processor communication bottlenecks. In this paper, we describe how to enable Parallel Deep Neural Network Training on the IBM Blue Gene/Q (BG/Q) computer system. Specifically, we explore DNN training using the data-parallel Hessian-free 2nd order optimization algorithm. Such an algorithm is particularly well-suited to parallelization across a large set of loosely coupled processors. BG/Q, with its excellent inter-processor communication characteristics, is an ideal match for this type of algorithm. The paper discusses how issues regarding programming model and data-dependent imbalances are addressed. Results on large-scale speech tasks show that the performance on BG/Q scales linearly up to 4,096 processes with no loss in accuracy. This allows us to train neural networks using billions of training examples in a few hours.

Index Terms—Big data, speech recognition, high performance computing

1 INTRODUCTION

DEEP Neural Networks (DNNs) have become a popular in the machine learning community over the last few years [1], showing significant gains over state-of-the-art machine learning methods used in both speech and image processing. The development of pre-training algorithms [2] and better forms of random initialization [3], as well as the availability of improved hardware, has made it possible to train deeper networks than before, and in practice these deep networks have achieved excellent performance in speech recognition applications [4], [5], [6].

However, one drawback of DNNs is that training remains very slow, particularly in applications involving a lot of data. Given the increased amount of data currently available, particularly from multimedia sources, the need for algorithms to run on large-scale tasks is critical. Slow DNN training can be attributed to a variety of causes. First, models for big-data tasks are often trained with millions to billions of training examples, as using increased amounts of training data often improves DNN performance [7]. Second, large data sets often use DNNs with a large number of parameters (i.e., can be 10-50 million DNN parameters in many speech tasks) [6], [8]. Third, to date the most popular

methodology to train DNNs is the first-order stochastic gradient descent (SGD) optimization technique, which is a serial algorithm executed on a multi-core CPU. While second order optimization techniques, which are much easier to parallelize across machines, have been explored for DNN training, these methods are not always faster than training DNNs via SGD. However, these methods yield one of the best performing models for several training criteria. For example, as shown in [9], 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 compared to serial SGD. Therefore, parallelization methods for DNNs have not enjoyed much success, and training DNNs via SGD is still the most popular technique.

IBM Blue Gene/Q is designed to deliver ultra-scale performance with a standard programming environment. It is currently one of most efficiently scalable systems for computationally intensive science applications. It has enabled science to address a wide range of complex problems. With its good performance in analytics applications, Blue Gene/Q is a good platform for big data applications.

Speech recognition is one important big-data application, which we will use as an example in this paper, in presenting our algorithm to speed up DNN training. Speech recognition is an important, real-world big-data application, as models are large (roughly 10-50 million parameters) and models are typically trained on millions to billions of training examples [6]. Furthermore, the second order Hessian Free training algorithm [10] has been shown to produce state-of-the-art performance (measured by word-error-rate)

- The authors are with IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. E-mail: {ihchung, tsainath, bhuvana, picheny, gunnells, austel, uvc, bedk}@us.ibm.com.

Manuscript received 18 Jan. 2016; revised 27 July 2016; accepted 21 Oct. 2016. Date of publication 8 Nov. 2016; date of current version 17 May 2017.

Recommended for acceptance by U.V. Catalyurek.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.

Digital Object Identifier no. 10.1109/TPDS.2016.2626289

on vision and speech tasks [5], making it an ideal algorithm to explore on a Blue Gene/Q architecture.

The objective of this paper is to explore speeding up second order optimization of DNN training using a Blue Gene/Q computer system. We specifically apply BG/Q to the Hessian-free (HF) algorithm [10], which has shown state-of-the-art performance with DNNs on many large vocabulary continuous speech recognition (LVCSR) tasks [5]. Since DNN training for speech consists of both cross-entropy (CE) and sequence training (ST) stages [11], 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 three 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 3-5 \times speedup over SGD with GPUs, with no degradation in word error rate (WER). In addition, for ST, HF BG/Q provides a 2-11 \times speedup over SGD with GPUs, with a slightly better WER.

With this Parallel DNN training application enablement and performance analysis, it also provides input for future generation system/architecture design.

The rest of this paper is organized as follows. Section 2 shows the related work. Section 3 gives an overview of the Blue Gene/Q system. Section 4 introduces the Parallel Deep Neural Network Training. Section 5 describes the challenges to enable and tune the application on the Blue Gene/Q system. Section 6 presents the performance results. The discussion and conclusion are presented in Sections 7 and 8.

2 RELATED WORK

2.1 DNN Training Methods

Stochastic Gradient Descent remains one of the most popular approaches for training DNNs. SGD methods are simple to implement and are generally faster for large data sets when compared to second order methods [9].

While parallel SGD methods have been successfully explored for convex problems [12], for non-convex problems such as DNNs it is very difficult to parallelize SGD across machines. With SGD the gradient is computed over a small collection of frames (known as a mini-batch), which is typically on the order of 100-1,000 for speech tasks [13]. Splitting this gradient computation onto a few parallel machines, coupled with the large number of network parameters used in speech tasks, results in large communications costs in passing the gradient vectors from worker machines back to the master. Thus, it is generally cheaper to compute the gradient serially on one machine using the standard SGD technique [14]. It is important to note that recently [15] explored a distributed asynchronous SGD method to improve DNN training speed.

Second-order batch methods, including conjugate gradient (CG) or limited-memory BFGS (L-BFGS), generally compute the gradient over all of the data rather than a mini-batch, and therefore are much easier to parallelize [16]. However, as shown in [9], parallelization of dense networks can actually be slower than serial SGD training, again because of communication costs in passing models and gradients as well as the need to run more training iterations compared to serial SGD.

2.2 Improved Matrix Libraries

Math library involving matrix calculation plays an important role in the high performance computing. Prior work

such as [17] involves DGEMM on Blue Gene/Q. General optimization of mathematical routines are needed in applications such as [18], [19]. The matrix calculation tuning for Parallel Deep Neural Network Training is focused on SGEMM which is not as heavily utilized in conventional high performance computing applications.

3 ARCHITECTURE

The IBM Blue Gene/Q system [20] is the third generation of high-performance computer in the Blue Gene series. Blue Gene/Q's processor [21] is a system-on-chip multi-core processor with 16 A2 PowerPC 64 bit cores. Each A2 core runs at 1.6 GHz, is four-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 four-wide SIMD double precision unit known as the QPX. The QPX unit is capable of delivering up to four fused multiply-add results per processor clock.

Each core has 16 K-byte private level 1 (L1) cache and 2 K-byte prefetching buffer (L1P). All the cores on the same processor share a 32 M-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.

Built on the Blue Gene hardware design is 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 PARALLEL DEEP NEURAL NETWORK TRAINING

The challenge in performing distributed optimization is to find an algorithm that uses large data batches that can be split across compute nodes without incurring excessive overhead while simultaneously achieving performance competitive with stochastic gradient descent. One class of algorithms for this problem uses second-order optimization, with large batches for the gradient and much smaller batches for stochastic estimation of the curvature [10], [22], [23]. A distributed implementation of one such algorithm has already been applied to learning an exponential model with a convex objective function for a speech recognition task [22].

The current study uses Hessian-free optimization [10] because it is specifically designed for the training of deep neural networks, which is a non-convex problem. Let θ denote the network parameters, $\mathcal{L}(\theta)$ denote a loss function, $\nabla\mathcal{L}(\theta)$ denote the gradient of the loss with respect to the parameters, \mathbf{d} denote a search direction, and $\mathbf{B}(\theta)$ denote a matrix characterizing the curvature of the loss around θ . The central idea in Hessian-free optimization is to iteratively form a quadratic approximation to the loss,

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

and to minimize this approximation using conjugate gradient, which accesses the curvature matrix only through matrix-vector products $\mathbf{B}(\theta)\mathbf{d}$ that can be computed efficiently for neural networks [24]. If $\mathbf{B}(\theta)$ were the Hessian and conjugate gradient were run to convergence, this would be a matrix-free Newton algorithm. In the Hessian-free algorithm, the conjugate gradient search is truncated, based on the relative improvement in approximate loss, and the curvature matrix is the Gauss-Newton matrix [25], which unlike the Hessian is guaranteed positive semidefinite, with additional damping: $\mathbf{G}(\theta) + \lambda\mathbf{I}$.

Our implementation of Hessian-free optimization, which is illustrated as pseudo code in Algorithm 1, closely follows that of [10], except that it currently does not use a preconditioner [26]. Gradients are computed over all the training data. Gauss-Newton matrix-vector products are computed over a sample (about 1 to 3 percent of the training data) that is taken each time CG-Minimize is called. The loss, $\mathcal{L}(\theta)$, is computed over a held-out set. CG-Minimize($q_\theta(\mathbf{d}), \mathbf{d}_0$) uses conjugate gradient to minimize $q_\theta(\mathbf{d})$, starting with search direction \mathbf{d}_0 . Similar to [10], the number of CG iterations is stopped once the relative per-iteration progress made in minimizing the CG objective function falls below a certain tolerance. The CG-Minimize function returns a series of steps $\{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N\}$ that are then used in a backtracking procedure. The parameter update, $\theta \leftarrow \theta + \alpha\mathbf{d}_i$, is based on an Armijo rule backtracking line search. $\beta < 1.0$ is a momentum term.

Algorithm 1. Hessian-Free Optimization (After [10])

```

initialize  $\theta; \mathbf{d}_0 \leftarrow \mathbf{0}; \lambda \leftarrow \lambda_0; \mathcal{L}_{\text{prev}} \leftarrow \mathcal{L}(\theta)$ 
while not converged do
   $\mathbf{g} \leftarrow \nabla \mathcal{L}(\theta)$ 
  Let  $q_\theta(\mathbf{d}) = \nabla \mathcal{L}(\theta)^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T (\mathbf{G}(\theta) + \lambda \mathbf{I}) \mathbf{d}$ 
   $\{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N\} \leftarrow \text{CG} - \text{MINIMIZE}(q_\theta(\mathbf{d}), \mathbf{d}_0)$ 
   $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}(\theta + \mathbf{d}_N)$ 
  for  $i \leftarrow N - 1, N - 2, \dots, 1$  do ▷ backtracking
     $\mathcal{L}_{\text{curr}} \leftarrow \mathcal{L}(\theta + \mathbf{d}_i)$ 
    if  $\mathcal{L}_{\text{prev}} \geq \mathcal{L}_{\text{best}} \wedge \mathcal{L}_{\text{curr}} \geq \mathcal{L}_{\text{best}}$  then
       $i \leftarrow i + 1$ 
      break
     $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}_{\text{curr}}$ 
  if  $\mathcal{L}_{\text{prev}} < \mathcal{L}_{\text{best}}$  then
     $\lambda \leftarrow \frac{3}{2}\lambda; \mathbf{d}_0 \leftarrow \mathbf{0}$ 
    continue
   $\rho = (\mathcal{L}_{\text{prev}} - \mathcal{L}_{\text{best}}) / q_\theta(\mathbf{d}_N)$ 
  if  $\rho < 0.25$  then
     $\lambda \leftarrow \frac{2}{3}\lambda$ 
  else if  $\rho > 0.75$  then
     $\lambda \leftarrow \frac{3}{2}\lambda$ 
   $\theta \leftarrow \theta + \alpha\mathbf{d}_i; \mathbf{d}_0 \leftarrow \beta\mathbf{d}_N; \mathcal{L}_{\text{prev}} \leftarrow \mathcal{L}_{\text{best}}$ 

```

To perform distributed computation, we use a master/worker architecture in which worker processes distributed over a compute cluster perform data-parallel computation of gradients and curvature matrix-vector products and the master implements the Hessian-free optimization and coordinates the activity of the workers. All communication between the master and workers is via MPI. The master/worker architecture used in this paper is a simple one-layer architecture, with one master and many workers.

5 APPLICATION ENABLEMENT AND PERFORMANCE TUNING

In this section we describe our efforts to enable and tune the Parallel DNN Training application on the Blue Gene/Q computer system.

5.1 Matrix Multiplication on Blue Gene/Q

In order to optimize the performance of the matrix multiplication routine on Blue Gene/Q as well as to make it performance portable, the architectural levels are viewed in something of a telescoping fashion: node, core, and thread. As is typical on multicore architectures, decisions made at each level affect the decisions made at other levels as a result of feature interaction, so the division is not clean and optimizing locally tends to lead to implementations that are not globally optimal.

5.1.1 The Blue Gene/Q Architecture

As described in Section 3, a Blue Gene/Q node consists of 16 PowerPC A2 compute cores operating at 1.6 GHz and, each, in turn, made up of four true hardware threads, fully in-order and single issue. Since each core can execute four wide SIMD FMA instructions, the floating point peak of a core is $8 \times 1.6 = 12.8$ GFLOPS, thus the theoretical peak operating speed of a node is 204.8 GFLOPS. When considering matrix-multiplication, this theoretical peak is (almost) achievable in practice for two primary reasons. First, because the caches and the memory itself are capable of supporting the bandwidth requirements of matrix multiplication. Second, because not every instruction in matrix multiplication is an FMA, it is important that multiple instructions, namely a floating point load along with an FMA, be executable in any given cycle. On Blue Gene/Q this is possible; while any single thread can only execute one instruction per cycle, two threads on the same core can execute (two) instructions at the same time as long as only one is performing a floating point arithmetic operation and the other is executing a different type of operation, such as a load or store.

5.1.2 A View from the Thread Level

At the lowest level of software construction, there is the inner kernel for matrix multiplication. This code is written in assembly and is not threaded (the scaffolding for threading is encoded at a higher level of computational granularity). There are two overriding goals when writing code for the inner kernel. First, cover the latency to the appropriate level(s) in the memory hierarchy which equates to separating operand load from operand use by the appropriate number of cycles. Second, reduce the bandwidth to a level to which it can be fed by the cache(s) in which the operands will reside. For Blue Gene/Q, the inner (register-block) kernel consists of an 8×8 C matrix updated by a sequence of outer products. The A and B matrices are reformatted in such a way so as to allow strictly stride-one access to both matrices, thus the L1P prefetch engine is used, reducing the latency that must be covered.

Given that the code is written in assembly language, the scheduling of the instructions as well as the register layout can be made precise and latency coverage (approximately

20 cycles from the L1P prefetch unit) virtually guaranteed. Using higher-level languages such as C or Fortran leads to code that is easier to maintain, but may be subject to lack of SIMD instruction generation, suboptimal scheduling, and less aggressive register allocation by the compiler. Thus, changes in the compiler can lead to substantial changes in the generated code and the witnessed performance of computationally-intensive routines. It should be noted that the use of intrinsics, which have a one-to-one mapping to assembly level instructions, has been successfully employed in this domain on Blue Gene/Q [27].

5.1.3 Combining Threads: A View from the Core Level

While multiple instruction issue requires the use of only two hardware threads per core, for matrix multiplication, we used four hardware threads in all instances. This allows the system to make more effective use of dual-issue capabilities. It also allows for us to get great leverage from the small (4 KB per thread) L1D cache via a novel software feature that we refer to as implicitly synchronized threads [28], [29].

Given the size of the L1D cache, the conventional practice of storing a slice of the A or B matrix in the cache is not viable unless the matrices have a very small K dimension. Using blocking on that matrix dimension results in multiple reads and writes of the C matrix, erasing potential performance gains. However, with precise control of task layout, data sharing can be improved in theory. Implicit synchronization allows us to realize this benefit in practice by minimizing thread skew (i.e., ensuring that all four threads execute at the same level of instruction throughput).

To illustrate, if we view the assignment of blocks of the resultant C matrix to be done in 2×2 sets to each core, we can see that both the A and B operand are shared across two threads. While this somewhat mitigates the cache splitting effect, the real benefit is somewhat different. With the threads cooperating to prefetch operand streams, the cache can be used as a staging area for the operands, as well as serving as the device through which the threads are coordinated. To see this, assume that the code is written (timed) so as to rely upon the property that every cache line needed is in the L1D cache before it is loaded in a register so as to be used in a computation. Further consider that for each thread, every other cache line of each operand is to be prefetched by a different (partner) thread. Thus, threads that advance past other threads are slowed down by the fact that half of their cache lines are “missing” and trailing threads can catch up. While it would seem that this scheme could break down should any thread get so far ahead of its compatriots that the operands that it prefetched are no longer in the L1D cache, higher levels of cache can be utilized in the same manner, greatly decreasing the odds of this occurring and extremely infrequent explicit synchronizations (via OpenMP and the special synchronization features of Blue Gene/Q) can be issued to eliminate this possibility as well as to orchestrate not just threads, but cores.

This shared prefetching, and the mutual dependency that it implies, enforces the implicit synchronization of threads primarily through the use of the L1D cache in this instance, but the method can be used at any level of the memory

hierarchy and is not restricted to Blue Gene/Q systems. Furthermore, synchronizing and the cooperative use of shared operands, also lowers bandwidth requirements, as, with the correct task layout, a 2×2 grid, the four individual 8×8 outer product computations, each requiring 8 bytes/cycle (in double precision arithmetic) were they to run at 100 percent of peak, become one 16×16 outer product that requires only half that bandwidth, via a reduction in the surface to volume ratio.

5.1.4 Cooperation Between Cores: The Task and Node Levels

While the code for matrix multiplication is written so that it is correct regardless of the number of threads used per core or the number of cores used per node, it is tuned for the case where four threads per core are used and works best when, at the task level, the number of cores per rank is a perfect square. The square value allows for a (conceptually) square task layout, reducing the bandwidth from the (shared) L2 cache as much as is possible. This allows not only power savings, but the preloading of “to be used” operands with minimal impact on processing speed.

Two large levers that we have at the task level are the aforementioned explicit synchronization, made efficient through Blue Gene/Q’s highly optimized OpenMP library, and a more hands-on control of allocated memory. Both of these techniques are used to gain close to peak performance at the expense of considerable care in coding and maintenance and are sometimes not done in more widely used libraries for those, and other, reasons. Explicit synchronization requires no real explanation; this is simply syncing all of the threads on a rank so that the cooperative properties mentioned above are established as a precondition. We tend to do this as new cache blocks are loaded, but the granularity of these synchronizations involves a trade-off pitting synchronization overhead against property guarantees. We manage memory by essentially keeping track of what we have allocated so that we can reallocate out of that memory instead of repeatedly freeing and allocating when new memory is required. This tends to enhance performance only slightly for large matrix operations, but it greatly reduces timing jitter which is invaluable when tuning a library or an application. While we constructed a rather involved method for controlling this allocated memory (the complexity stems from what has to be done when another application requests memory and cannot get it because our library is holding it), we did not use that method for the experiments seen here.

5.1.5 Bringing It Together

There are several advantages to considering the system holistically, rather than constructing the code bottom-up (thread to node) or top-down. We have already mentioned the shared prefetching and the ability to lower the bandwidth requirements at both the node and core level by laying out the tasks to “slice up” the matrices by square “cookie cutters.” Both of these properties require us to slice between levels of the system hierarchy. Knowledge of task layout also allows the implementer to have the threads cooperate for long- and short-term data prefetching. Long-

term prefetching is used to pull in the next cache or register block of a matrix operand into the cache and is typically amortized over a large number of operations. As the C operand is not re-used in Level 3 BLAS operations like DGEMM, the intricate long-term prefetching of C matrix blocks proved important for the last 5 percent of the performance gained. Short-term prefetching is the technique used in the inner kernel to hide latency. While latency hiding is greatly improved via the use of multiple threads on a core, cooperative prefetching has all of the benefits mentioned above and improves latency tolerance by allowing us to utilize more registers, per thread, for prefetching. This improves performance when new streams are started (the latency is higher when the L1P unit is not engaged) and when pulling new operand blocks from the L2 cache or main memory.

The tuning done specifically for this application included optimizations to maintain performance as the number of ranks per node is changed, handling small matrices and matrices with dimensions that do not lend themselves to full SIMDization without losing significant efficiency, and changes to the inner kernel to greatly improve the performance of calculations done using single-precision arithmetic.

5.2 Communication

In order to scale up the application, we abandoned the socket communication, the method used on a 64-node Intel/Linux cluster, and rewrote the application to utilize MPI communication. Socket communication requires the programmer take care of all details of communication which includes channels (communication parties) set up, data packaging and routing. MPI communication provides much more functionality when managing many nodes (of the order of thousands) in a large scale application and is portable, based on the MPI standard.

The Blue Gene/Q MPI communication library is heavily optimized, as MPI has wide support in high performance computing. The MPI-2.2 standard is supported on BG/Q with the exception of a few features that are incompatible with the compute node kernel (e.g., those that require process forking). The optimized MPI on BG/Q is implemented on top of PAMI, a high-level active-message interface used to support many programming models.

The application previously used files to communicate among processes in order to synchronize the weights that are used in the DNN. This weight-synchronization step was converted to rely upon MPI, performance was improved by using the broadcast (MPI_Bcast) mechanism to take the advantage of the optimized MPI collectives available on the system.

5.3 Load Balance

One key factor affecting scalability is load balancing. At the processor/core level, efficiently overlapping computation and communication helps to improve the performance. At the application level, distributing data evenly across compute 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 recognition tasks, the

training data comprises of several spoken utterances from thousands of speakers. These utterances in the training set are not all of the same length, so we preprocessed the data by sorting and computed the number of utterances per worker such that they all receive an equal amount of data. This improves the load balancing and the effect is more apparent when the training data is scaled to larger sizes (millions of samples).

6 EVALUATION

In this section, we first use up to two racks (2,048 nodes) of Blue Gene/Q and study the effect of varying MPI/OpenMP configuration on application performance. We then compare HF with BG/Q to HF run on a CPU cluster and SGD run on a GPU, both for CE and ST. We also analyze the application to understand the manner in which the computation and (MPI) communication cycles are spent.

6.1 Training HF for BlueGene/Q

We start with a small sized (50 hour) input training data using one rack of Blue Gene/Q. 50 hrs of audio data amounts to roughly 18 million training samples. A plot of the time taken for each configuration is shown in Fig. 1a. As one might imagine, scaling up by increasing the number of OpenMP threads to fully utilize the cores improves the performance. As there are 16 cores on a node, one must use at least 16 threads to utilize all cores. This can be done by using 1 rank per node and 16 threads (1,024-1-16) per rank or in any other combination that leads to the product of ranks per node and threads per rank being equal to 16. In our study, we target 64 threads per node. The potential configurations are as expected. As for the Rank(s)/Node and OpenMP ratio, the performance of 2,048-2-32 is slightly better than 4,096-4-16 which is better than 1,024-1-64. Using more threads per core helps to hide the time gaps (e.g., stall cycles) for the hardware execution components.

Fig. 1b illustrates how these runs scale with a larger input training data set comprising of 400 hours. An additional 22 percent speedup is obtained when the configuration is scaled to 8,192-4-16 (two Blue Gene racks). A DNN on 400 hours can be trained using this configuration in 6.3 hours.

6.2 Timing Performance

6.2.1 50 Hour Broadcast News

The experiments are conducted on a 50 hour English Broadcast News transcription task [26] and results are reported on 100 speakers in the EARS dev04f set.¹ An LVCSR recipe described in [30] 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 nine frames around the current frame. Following [8], we use a five-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 [6], followed by CE and ST.

1. This set has one speaker removed.

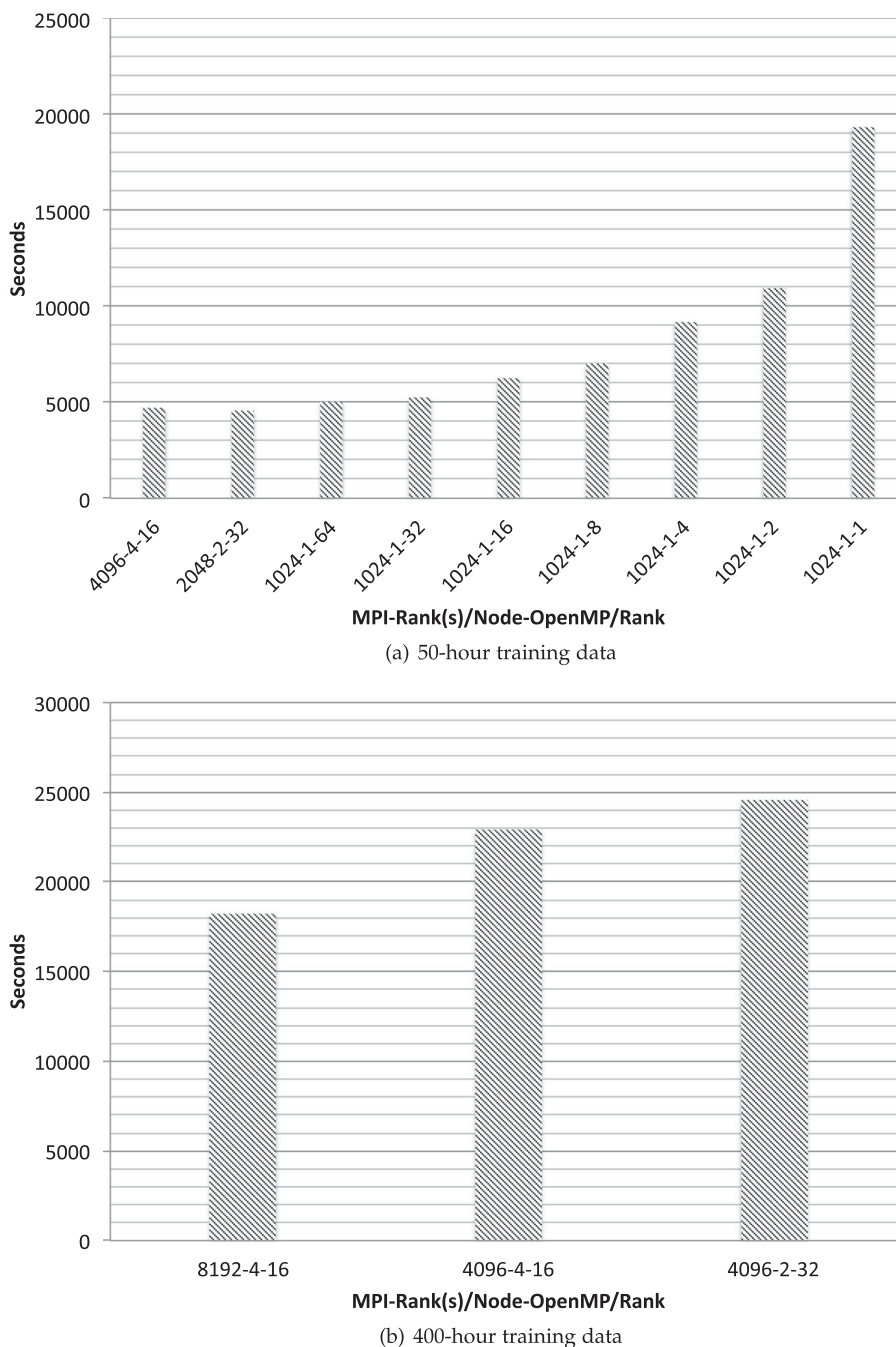


Fig. 1. Execution time for different configurations.

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 training, the WER of all three

TABLE 1
Training Time, 50 hr Broadcast News

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	

algorithms is the same. SGD using GPUs is faster than 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 $5\times$ speedup over the GPU for CE. For ST, BG/Q offers a $1.7\times$ 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 [31]. However, since HF uses large batches of data for gradient and CG computations, it not negatively impacted by

TABLE 2
Training Time, 430 hr Broadcast News

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	14.7
Speedups	3		11.6	

utterance randomization. Increasing the batch size for GPUs to alleviate this frame versus utterance issue would increase training time, and was also shown in [32] not to help. Thus, HF with BG/Q provides the fastest training method and best WER for CE and ST.

6.2.2 430 Hour Broadcast News

The 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 nine frames around the current frame. The architecture consists of five hidden layers with 1,024 hidden units per layer and a sixth softmax layer with 5,999 output targets.

We explore how BG/Q HF scales to a larger 430 hr BN task, now run with two racks. Since SGD GPU is faster than 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 3 \times speedup over SGD CE and 11.6 \times 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 (11.6 \times) compared to 50 hr BN (1.7 \times). This is due not only to using two 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.

6.2.3 300 Hour Switchboard

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 [6], the architecture consists of six hidden layers with 2,048 hidden units per layer and a seventh softmax layer with 9,300 output targets.

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 4 \times and 10.3 \times over SGD GPU for CE and ST, while matching the accuracy of the HF CPU implementation in Table 3.

6.3 Further Speedups

Various speedups have been explored for the HF algorithm, including low-rank matrix factorization [33], pre-

TABLE 3
Training Time, 300 hr Switchboard

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	15.1
Speedups	4		10.3	

conditioning for the conjugate gradient and sampling the data used for the gradient and CG steps [34]. In this section, we explore what speedups these ideas have provided on top of the BG/Q architecture. Table 4 shows that we get an additional 1.2 \times and 1.8 \times 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 4.2 \times speedup for CE and 21 \times 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 [33], 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 [33].

6.4 Cycles Breakdown

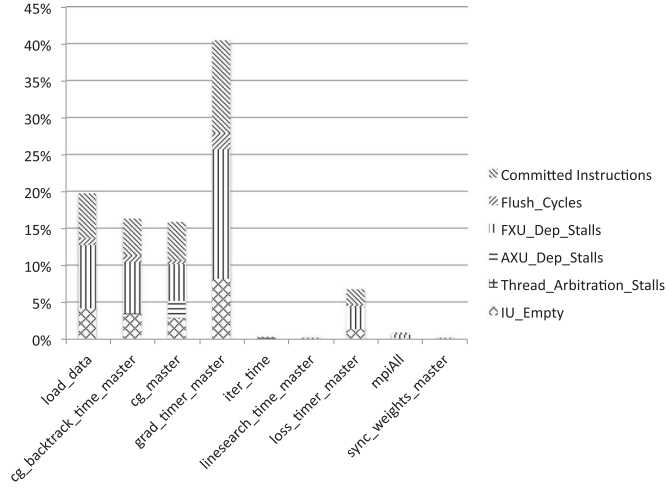
Figs. 2 and 3 plot the cycles spent in each of the key function calls of the algorithm described in Section 4.

We further analyze the cycles consumed by the master and the worker processes. Figs. 2 and 3 show the breakdown in cycles. The IU_Empty are the cycles where Instruction Unit is empty (e.g., Icache and Ierat Misses). AXU/FUX_Dep_Stalls are stall cycles due to dependency in Auxiliary/Fixed point units. The Committed Instructions are the cycles that the core is being “productive” doing the work. Figs. 4 and 5 are the cycles spent in collective and point-to-point MPI communication.

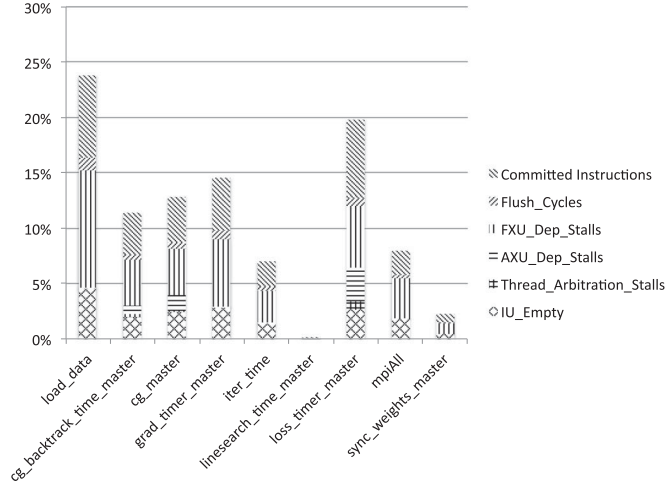
As the number of MPI ranks increases, shown in Fig. 2, the master process needs to spend more time distributing the data (load_data) using point-to-point MPI calls and synchronizing the weights (sync_weights_master) using collective MPI calls (Fig. 4c). One can observe that for almost all function calls, as the MPI ranks increase, the computation time decreases (such as gradient_loss), while for other functions such as worker_curvature_product, the computation time can vary. This can be attributed to the fact that the algorithm randomly selects a small percentage of the data for this part of the computation which could contribute to the variance seen in Fig. 3.

TABLE 4
Speedup Comparison

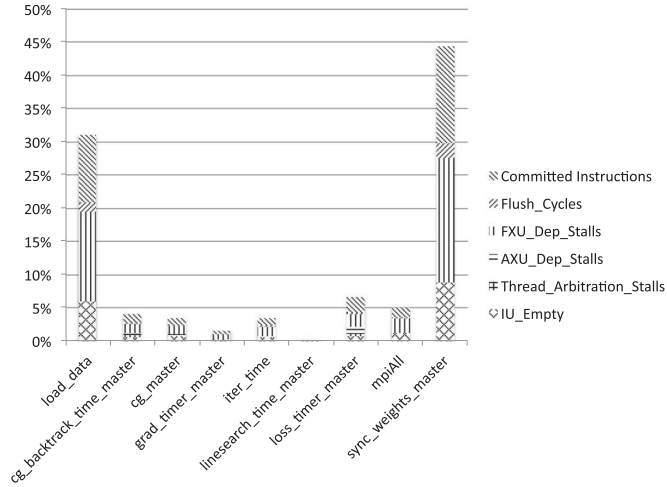
Hardware	Cross-Entropy		Sequence Training	
	Time (hrs)	WER	Time (hrs)	WER
HF BG/Q	21.7	16.5	3.6	15.1
+ [33] + [34]	18.4	16.6	2.0	12.7
Speedups	1.2		1.8	



(a) 1024 MPI ranks, 1 rank/node, 64 OpenMP/rank



(b) 2048 MPI ranks, 2 ranks/node, 32 OpenMP/rank

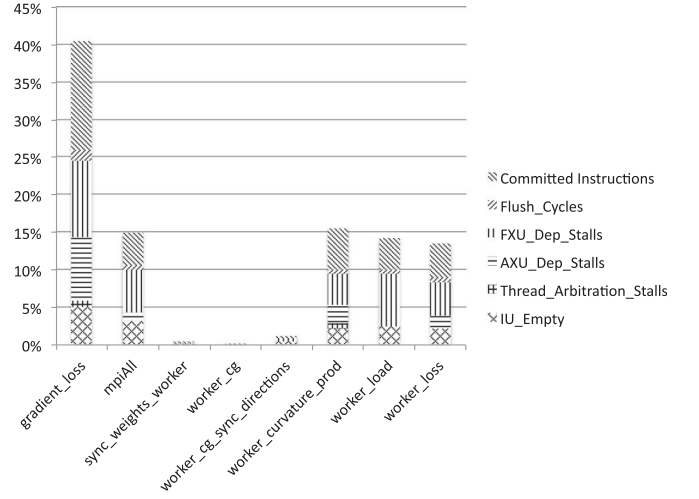


(c) 4096 MPI ranks, 4 ranks/node, 16 OpenMP/rank

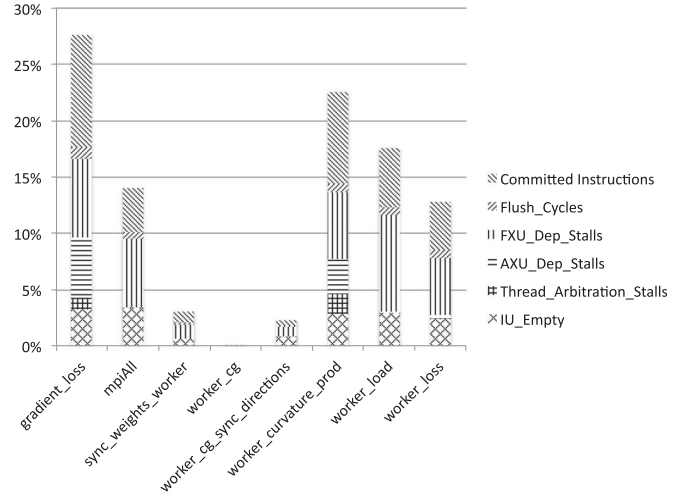
Fig. 2. Master process cycles break down.

7 DISCUSSION

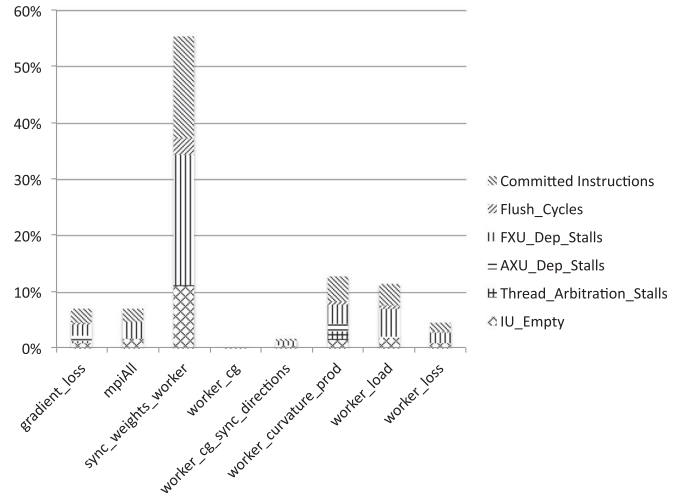
To enable the application on a large scale, we investigate and improve the parallelism in different levels. At the core level, the multi-threading helps to overlap the empty cycles in the hardware components. The tuned matrix library



(a) 1024 MPI ranks, 1 rank/node, 64 OpenMP/rank



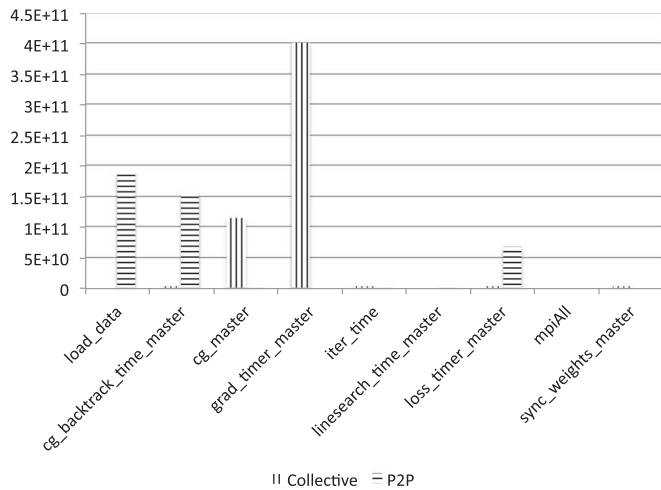
(b) 2048 MPI ranks, 2 ranks/node, 32 OpenMP/rank



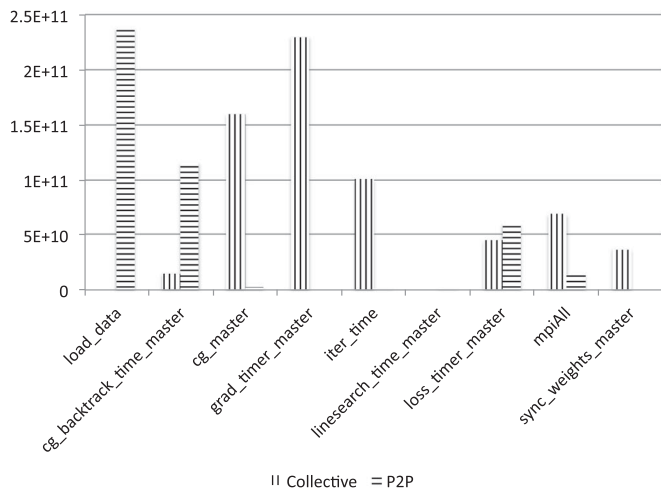
(c) 4096 MPI ranks, 4 ranks/node, 16 OpenMP/rank

Fig. 3. Worker process cycles break down.

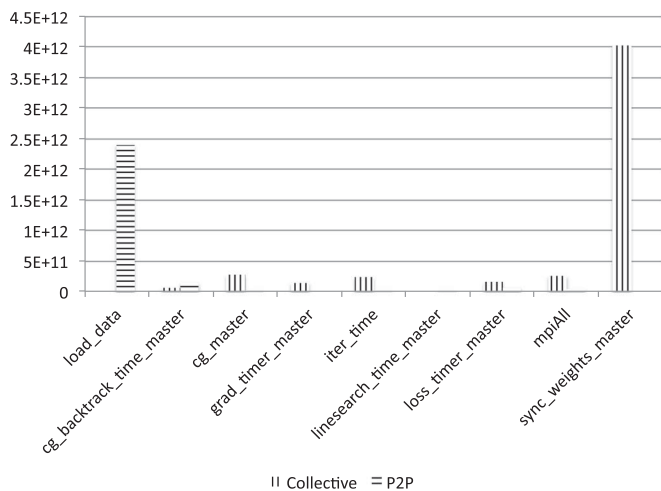
helps to utilize the SIMD unit to have more floating point instructions executed in a single cycle. At the node level, load balancing across MPI ranks synchronize the processes to improve the parallelization efficiency.



(a) 1024 MPI ranks, 1 rank/node, 64 OpenMP/rank



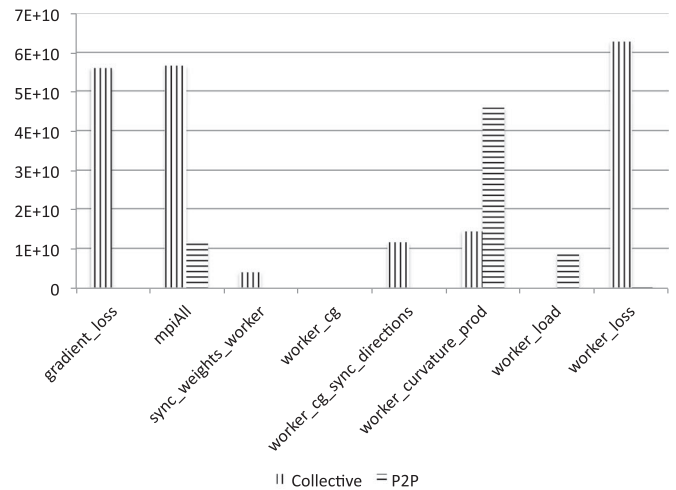
(b) 2048 MPI ranks, 2 ranks/node, 32 OpenMP/rank



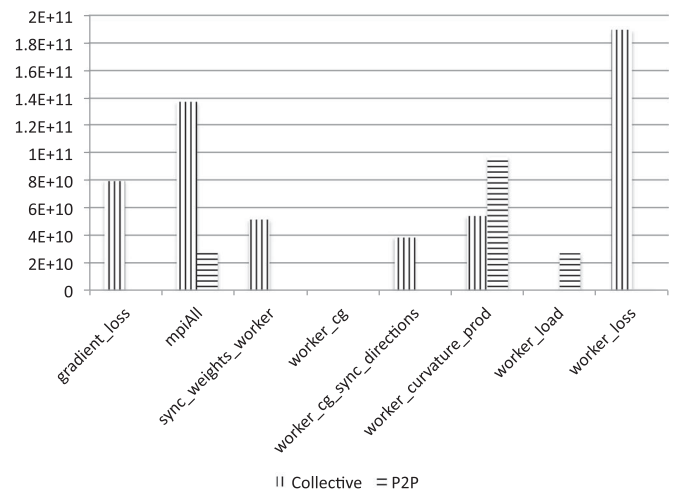
(c) 4096 MPI ranks, 4 ranks/node, 16 OpenMP/rank

Fig. 4. Master MPI communication time.

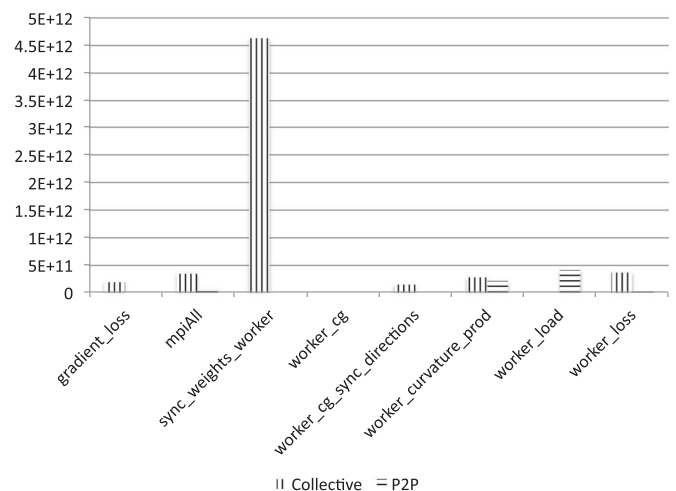
Analysis on our initial implementation indicated that the distribution of load (data) across the workers is critical in this application. Speech utterances are not of the same length, and therefore we needed to ensure that all workers processed the right set of utterances to adhere to the randomness needed by



(a) 1024 MPI ranks, 1 rank/node, 64 OpenMP/rank



(b) 2048 MPI ranks, 2 ranks/node, 32 OpenMP/rank



(c) 4096 MPI ranks, 4 ranks/node, 16 OpenMP/rank

Fig. 5. Worker MPI communication time.

the algorithm as well as to ensure no load imbalances. This also made sure that the master did not wait on just one or two workers that had more data to work with.

It is key to point out that a Linux cluster that can be built with the same number of cores as used in Blue Gene will

TABLE 5
Scaling up Performance

Training data	Intel Xeon 96 processes (hrs)	BG/Q 4096 MPI (hrs)	Speed Up	Frequency Adjustment
50-hour Cross-Entropy	9	1.3	6.9×	12.6×
50-hour Sequence	18.7	4.19	4.5×	8.2×

suffer from several communication bottlenecks (collisions), this is one of the main advantages of Blue Gene. A specialized network is needed to scale to that order of magnitude in order to take advantage of thousands of cores for this application.

Table 5 summarizes the overall speed-ups obtained with the Blue Gene architecture. Allowing for the differences in core processor speed between an Intel-CPU based infrastructure and the Blue Gene computer system, we find that speed-ups of up to a factor of twelve can be obtained for two different algorithms, namely one that uses cross-entropy as the objective function and another that uses a discriminative criterion as the objective function to train the DNN. Both of these algorithms are extensively applied in speech applications, and this speed-up is significant, especially when a typical training set operates on several thousands of hours of data. Each of these networks converges to their optimal weights, after 20 to 40 iterations through the entire data set. As such, in order to turn around several experiments, performance is paramount. The last column in the figure adjusts for the clock frequency differences between the Intel (2.9 GHz) and Blue Gene processors (1.6 GHz).

8 CONCLUSION

In this paper, we describe our efforts to enable Parallel Deep Neural Network Training on the IBM Blue Gene/Q (BG/Q) computer system. Specifically, we explore DNN training using the Hessian-free 2nd order optimization algorithm. We found that across three different LVCSR tasks, HF with BG/Q was 2-11× faster compared to SGD on a GPU. We observe that we can achieve speed-ups that scale linearly up to 4,096 processes. Beyond that, although we see a significant speed up, the speed improvements are sub-linear. We have demonstrated that with two racks of Blue Gene, we can train a deep network with over 100 M parameters in 6 hours. This would take approximately a month to train on the Linux cluster of Intel-CPU's. Scaling to large data sets increases the communication costs (data exchanges) between the workers such that they dominate the overall training time. This requirement renders training with Intel-CPU's inefficient when dealing with large data sets. We have also observed on the larger training set (400 hours) that the BG/Q was able to achieve a good speedup, of the same order seen with the 50 hour dataset. We have demonstrated these gains in speed over two types of input, and by scaling the data to billions of training samples. In summary, these speed-ups obtained with BG/Q, make the training of large networks more feasible, and open up new avenues

for research in compute-intensive applications. To date, the Hessian-free, second order optimization method with BG/Q provides the fastest training method and best WER for both, cross-entropy and sequence training objective functions on deep networks. Scalability of Blue Gene/Q stems from the lack of interference (e.g., noise, such as, OS jitter) from running computer processes. The system is essentially free of interference, which has been verified directly through measurements [35], [36]. The Blue Gene/Q is also a leader in energy efficiency compared to the 30 different systems studied [37] in the same time era. These factors further render the use of Blue Gene/Q more apt for large-scale, big data applications.

The Blue Gene/Q has many more hardware features such as hardware transactional memory, thread level speculation and list prefetch engine. In the future, we plan to investigate the possibility of utilizing those hardware features to further improve the performance such that the application can handle an even larger input training data set in a reasonable amount of time.

ACKNOWLEDGMENTS

We would like to express our appreciation for generous support and guidance from Fred Mintzer, John Magerlein, James Sexton and Michael Rosenfield.

REFERENCES

- [1] G. Hinton, et al., "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
- [2] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554, 2006.
- [3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
- [4] N. Jaitly, P. Nguyen, A. W. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2012.
- [5] 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. 13th Annu. Conf. Int. Speech Commun. Assoc.*, 2012, pp. 10–13.
- [6] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2011, pp. 437–440.
- [7] D. Yu, L. Deng, and G. E. Dahl, "Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2010.
- [8] 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. IEEE Workshop Automat. Speech Recognit. Understanding*, 2011, pp. 30–35.
- [9] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Pronchnow, and A. Ng, "On optimization methods for deep learning," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 265–272.
- [10] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 735–742.
- [11] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Apr. 2009, pp. 3761–3764.
- [12] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Proc. Advances Neural Inf. Process. Syst.*, 2010, pp. 2595–2603.
- [13] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," Mach. Learn. Group, Univ. Toronto, Toronto, Canada, Tech. Rep. 2010–003, 2010.

- [14] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Improving training time of deep belief networks through hybrid pre-training and larger batch sizes," in *Proc. NIPS Workshop Log-Linear Models*, 2012.
- [15] J. Dean, et al., "Large scale distributed deep networks," in *Advances Neural Inf. Process. Syst.*, 2012, pp. 1232–1240.
- [16] C. T. Chu, et al., "Map-reduce for machine learning on multicore," in *Proc. 19th Int. Conf. Advances Neural Inf. Process. Syst.*, 2007, pp. 281–288.
- [17] F. G. V. Zee, et al., "Implementing level-3 BLAS with BLIS: Early experience," Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, Tech. Rep. TR-13-03, 2013.
- [18] A. A. Mirin, et al., "Toward real-time modeling of human heart ventricles at cellular resolution: Amulation of drug-induced arrhythmias," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2012, Art. no. 2.
- [19] B. Carnes, et al., "Science at LLNL with IBM blue gene/Q," *IBM J. Res. Develop.*, vol. 57, no. 1/2, 2013, Art. no. 11.
- [20] M. Gschwind, "Blue compute/Q: Design for sustained multi-petaflop computing," in *Proc. 26th ACM Int. Conf. Supercomputing*, 2012, pp. 245–246. [Online]. Available: <http://doi.acm.org/10.1145/2304576.2304609>
- [21] R. A. Haring, et al., "The IBM blue gene/Q compute chip," *IEEE Micro*, vol. 32, no. 2, pp. 48–60, Mar. 2012.
- [22] R. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal, "On the use of stochastic Hessian information in unconstrained optimization," *SIAM J. Optimization*, vol. 21, no. 3, pp. 977–995, 2011.
- [23] O. Vinyals and D. Povey, "Krylov subspace descent for deep learning," in *Proc. NIPS Workshop Optimization Hierarchical Learn.*, 2011.
- [24] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Comput.*, vol. 6, no. 1, pp. 147–160, 1994.
- [25] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Comput.*, vol. 14, pp. 1723–1738, 2004.
- [26] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2009, pp. 3761–3764.
- [27] T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. van Zee, "Opportunities for parallelism in matrix multiplication. FLAME Working Note #71," Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, Tech. Rep. TR-13-20, 2013.
- [28] A. E. Eichenberger and J. A. Gunnels, "Shared prefetching to reduce execution skew in multi-threaded systems," Jul. 2013. [Online]. Available: <http://www.osti.gov/scitech/servlets/purl/1087835>
- [29] J. Gunnels, "Making good enough... better: Addressing the multiple objectives of high-performance parallel software with a mixed global-local worldview," Slides of a talk given at ICERM, Synchronization-reducing and Communication-reducing Algorithms and Programming Models for Large-scale Simulations, 2012. [Online]. Available: http://icerm.brown.edu/materials/Slides/tw-12-1/Making_Good_Enough...Better-Addressing_the_Multiple_Objectives_of_High-Performance_Parallel_Software_with_a_Mixed_Global-Local_Worldview_%5D_John_A_Gunnels_IBM.pdf
- [30] H. Soltan, G. Saon, and B. Kingsbury, "The IBM Attila speech recognition toolkit," in *Proc. IEEE Spoken Language Technol. Workshop*, Dec. 2010, pp. 97–102.
- [31] T. Sainath, K. Kingsbury, and B. Ramabhadran, "Improvements in using deep belief networks for large vocabular continuous speech recognition," IBM Thomas J. Watson Res. Center, Yorktown Heights, NY, USA, 2010.
- [32] G. Saon and H. Soltan, "A comparison of two optimization techniques for sequence discriminative training of deep neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2014, pp. 5567–5571.
- [33] T. 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. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2013, pp. 6655–6659.
- [34] T. 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. IEEE Workshop Automat. Speech Recognit. Understanding*, Dec. 2013, pp. 303–308.
- [35] K. Davis, et al., "A performance and scalability analysis of the BlueGene/L architecture," in *Proc. IEEE/ACM Conf. Supercomputing*, 2004, pp. 41–41.
- [36] K. Singh, E. Ipek, S. McKee, B. de Supinski, M. Schulz, and R. Caruana, "Predicting parallel application performance via machine learning approaches," *Concurrency Comput.: Practice Experience*, vol. 19, no. 17, pp. 1–7, 2006.
- [37] [Online]. Available: <http://green500.org/>



I-Hsin Chung received the PhD degree in computer science from the University of Maryland, College Park, in 2004, prior to joining IBM Research. He is a research scientist with IBM Research. His research focus has been in the system architecture area including high performance and data centric computing. His experience includes performance analysis, modeling and tooling on IBM platforms such as Linux/AIX on POWER systems, and world-renowned Blue Gene series HPC systems. He is currently working

on the co-design for next generation data center and high performance computing systems by better understanding of application workload characteristics and data movement requirements. His novel works are applied in IBM HPC systems and are used widely. He has more than 50 published and creative works in top journals and conferences including the *IEEE Transactions on Parallel and Distributed Systems*, the *ACM Transactions on Mathematical Software*, the *Journal of Parallel and Distributed Computing*, SC, and IPDPS. He is a senior member of the IEEE.



Tara N. Sainath received the PhD degree in electrical engineering and computer science from MIT, in 2009. The main focus of the PhD degree work was in acoustic modeling for noise robust speech recognition. After the PhD degree, she spent 5 years with the Speech and Language Algorithms Group, IBM Thomas J. Watson Research Center, before joining Google Research. She has co-organized a special session on sparse representations with Interspeech 2010, Japan. She has also organized a special session

on deep learning with ICML 2013, Atlanta. In addition, she is a staff reporter of the *IEEE Speech and Language Processing Technical Committee (SLTC)* Newsletter. Her research interests include mainly in acoustic modeling, including deep neural networks, sparse representations, and adaptation methods.



Bhuvana Ramabhadran is a principal research staff member, and manager with IBM Watson. She leads a team of researchers in the Speech Technologies Research Group and coordinate research activities across IBM's World-Wide Research Labs in the areas of speech recognition, synthesis, spoken term detection, and machine learning. She serves as an adjunct professor with Columbia University, where she co-teaches a course on automatic speech recognition. She is the elected chair of the Speech and

Language Technical Committee (SLTC) of the IEEE Signal Processing Society (2015-2017) and has served as an area chair of ICASSP (2011-2016), Interspeech (2012, 2014-2016), served on the editorial board of the *IEEE Transactions on Audio, Speech, and Language Processing* (2011-2015), and organized ASRU 2011. She has served as the principal investigator on two major international projects: the NSF-sponsored Multilingual Access to Large Spoken Archives (MALACH) project, and the EU-sponsored TCSTAR project, and the lead with IBM for the Spoken Term Detection evaluation in 2006. She is currently responsible for acoustic and language modeling research for both, commercial and government projects ranging from voice search and transcription tasks to spoken term detection in multiple languages and expressive synthesis for IBM Watson. She is a fellow of the IEEE.



Michael Picheny received the doctorate degree from MIT. He is the senior manager of Watson multimodal with the Watson Group, IBM Thomas J. Watson Research Center. He has worked in the speech recognition area since 1981, joining IBM. He has been heavily involved in the development of almost all of IBM's recognition systems, ranging from the world's first real-time large vocabulary discrete system through IBM's product lines for telephony and embedded systems. He has published numerous papers in both journals and conferences on almost all aspects of speech recognition. He has received several awards from IBM for his work, including a corporate award, three outstanding Technical Achievement Awards and two Research Division Awards. He is the co-holder of more than 30 patents and was named a master's inventor by IBM, in 1995 and again in 2000. He served as an associate editor of the *IEEE Transactions on Acoustics, Speech, and Signal Processing* from 1986-1989, was the chairman of the Speech Technical Committee of the IEEE Signal Processing Society from 2002-2004. He served as an adjunct professor in the Electrical Engineering Department, Columbia University, in 2009 and 2012 and co-taught a course in speech recognition. He was a member of the board of International Speech Communication Association (ISCA) from 2005-2013 and named an ISCA fellow in 2014. He was the co-general chair of the IEEE ASRU 2011 Workshop, Hawaii. He is a fellow of the IEEE.



John Gunnels received the PhD degree in computer science from the University of Texas at Austin, in 2001. He joined IBM Research, in 2001 and is currently a distinguished research staff member, program director, and manager in the Data Centric Solutions Department, IBM Thomas J. Watson Research Center. His research interests include machine learning, performance analysis, code verification, high-performance library specification and development, large-scale scientific computing, and the effective use of hardware accelerators. He is an eight-time Gordon Bell Award finalist and a three-time Gordon Bell Award winner. He is a senior member of the IEEE.



Vernon Austel has worked with IBM Thomas J. Watson Research Center, Yorktown Heights, New York, since 1994. He has worked on compilers, user interfaces, simplex solvers, traffic schedulers, and intrinsic library functions. He currently works with the Optimization Research Group on customer engagements.



Upendra Chauhari received the SB, SM, and PhD degrees from the Massachusetts Institute of Technology, in 1991, 1993, and 1997, respectively, where the main focus of his research was in the field of statistical signal processing and non-orthogonal signal expansions. He became a post doctoral associate with MIT after finishing the PhD degree and continued work on signal expansions, particularly for speech. While at MIT, he also worked on computational complexity, multi-access communication theory, and information retrieval. Later, he moved to the Human Language Technologies Group, IBM Thomas J. Watson Research Center where he is a research staff member. His work centers on statistical modeling and signal processing, with applications in speaker modeling and recognition. He is currently with Bank of America Merrill Lynch.



Brian Kingsbury (M'97, SM'09) received the BS degree in electrical engineering from Michigan State University and the PhD degree in computer science from the University of California, Berkeley. He is a research scientist with IBM Watson Group. His research interests include deep learning, large-vocabulary speech transcription, and keyword search. He is currently co-PI and technical lead for LORELEI, an IBM-led consortium participating in the IARPA Babel program. He has contributed to IBM's entries in numerous competitive evaluations of speech technology, including Switchboard, SPINE, EARS, Spoken Term Detection, and GALE. He has served as a member of the Speech and Language Technical Committee of the *IEEE Signal Processing Society* (2009-2011), as an ICASSP speech area chair (2010-2012), an associate editor of the *IEEE Transactions on Audio, Speech, and Language Processing* (2012-2016), and as a program chair of the International Conference on Representation Learning (2014-2016). He is a senior member of the IEEE.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.