Exploiting Approximate Computing for Deep Learning Acceleration

(Invited Paper)

Chia-Yu Chen, Jungwook Choi, Kailash Gopalakrishnan, Viji Srinivasan, Swagath Venkataramani
IBM T. J. Watson Research Center
Contact Email: choij@us.ibm.com

***Abstract*—Deep Neural Networks (DNNs) have emerged as a powerful and versatile set of techniques to address challenging artificial intelligence (AI) problems. Applications in domains such as image/video processing, natural language processing, speech synthesis and recognition, genomics and many others have embraced deep learning as the foundational technique. DNNs achieve superior accuracy for these applications using very large models which require 100s of MBs of data storage, ExaOps of computation and high bandwidth for data movement. Despite advances in computing systems, training state-of-the-art DNNs on large datasets takes several days/weeks, directly limiting the pace of innovation and adoption.**

In this paper, we discuss how these challenges can be addressed via approximate computing. Based on our earlier studies demonstrating that DNNs are resilient to numerical errors from approximate computing, we present techniques to reduce communication overhead of distributed deep learning training via adaptive residual gradient compression (*AdaComp*), and computation cost for deep learning inference via Prameterized clipping ACTivation (*PACT*) based network quantization. Experimental evaluation demonstrates order of magnitude savings in communication overhead for training and computational cost for inference while not compromising application accuracy.

I. Introduction

Deep Learning (DL) has emerged as the dominant Machine Learning algorithm showing remarkable success in a wide spectrum of challenging application domains ranging from image processing [1], machine translation [2], speech recognition [3] and many others. For applications in each of these domains, Deep Neural Networks (DNNs) achieve superior accuracy by using very large models which require 100s of MBs of data storage, ExaOps of computation and high bandwidth for data movement. Despite advances in computing systems, training state-of-the-art DNNs on large datasets takes several days/weeks, directly limiting the pace of innovation and adoption.

Approximate computing is gaining traction as a computing paradigm for wide range of cognitive applications that aim to extract deep insight from vast quantities of data. There is an exactness implied by traditional computing that is not needed in the processing of most types of these data. Thus, approximate computing aims to relax these constraints on exactness with the goal of obtaining significant gains in computational throughput – while still maintaining an acceptable quality of results. In our prior work [4], we conducted an in-depth study on the effectiveness of approximate computing across a set of applications spanning the domains of DSP, robotics, and deep learning. We focused on popular approximate computing techniques such as *loop perforation*, *reduced precision arithmetic* and *relaxed synchronization*. Our results demonstrate that applications in these domains are amenable to applying multiple approximation techniques simultaneously and achieve compounded benefits from these techniques. In particular, in the case of DNN training, we could skip 50% of iterations of the execution-time dominant program loops, as well as reduce computation bit-precision down to 10-16 bits, and relax synchronization and achieve 50% reduction in the overall execution time.

In this paper, we further exploit approximate computing to improve large scale Deep Learning using: 1) gradient compression to minimize communication overhead of distributed deep learning training and 2) activation quantization to minimize computation cost for deep learning inference.

Highly distributed training of DNNs on future compute platforms (offering 100s of TeraOps/s of computational capacity) is expected to be severely constrained by communication overheads. For example, data-parallel distributed DNN training requires exchange of gradients across the learners, leading to communication-bounded training speed. To overcome this limitation, we introduce a novel gradient compression technique - Adaptive Residual Gradient Compression (**AdaComp** [5]), which uses localized selection of gradient residues and automatically tunes the compression rate depending on local activity to exploit both sparsity and quantization for compression. On a wide spectrum of state of the art Deep Learning models in multiple domains (vision, speech, language), we demonstrate end-to-end compression rates of $\sim 200\times$ for fully-connected and recurrent layers, and $\sim 40\times$ for convolutional layers, without any noticeable degradation in model accuracies.

Turning our attention to DL inference, we observed that the high classification accuracy comes at the expense of significant computation cost (area/energy). To reduce the computation cost, a number of network quantization schemes focusing primarily on quantizing weights have been proposed. Straight-forward extension of these schemes to activation quantization incurs significant accuracy degradation. We introduce a novel quantization scheme for activations during training - that enables neural networks to work well with ultra-low-precision weights and activations without any significant accuracy degradation. This technique, PArameterized Clipping acTivation (**PACT** [6]), uses an activation clipping parameter α that is optimized during training to find the right quantization scale. PACT allows quantizing activations to arbitrary bit precisions, while achieving much better accuracy relative to published state-

978-3-9819263-0-9/DATE18/©2018 EDAA

of-the-art quantization schemes. We show that both weights and activations can be quantized to 4-bits of precision while still achieving accuracy comparable to full precision networks across a range of popular models and datasets. We also show that exploiting these reduced-precision computational units in hardware can enable a super-linear improvement in inferencing performance due to a significant reduction in the area of accelerator compute engines coupled with the ability to retain the quantized model and activation data in on-chip memories.

The rest of the paper is organized as follows. Our prior work on approximate computing for DNNs is summarized in Section II. Section III and IV describe gradient compression (AdaComp) and activation quantization (PACT), respectively along with experimental results. Conclusions are presented in Section V.

II. Approximate Computing for DNNs

Although computationally intensive, DNNs are resilient to numerical errors from approximate computing. Algorithm-level noise-tolerance of deep learning training may be leveraged to relax accuracy constraints in application specific hardware, resulting in systems that achieve better area- and power-efficiency than traditionally designed highly-accurate hardware systems. Our prior work [4] demonstrated that DNNs realize compounded benefits when multiple approximation techniques are applied simultaneously. Below, we present a short summary of the results presented in [4].

We first explored the use of low-precision floating-point arithmetic for DNN training. Decreasing precision of floating-point arithmetic reduces the area, power and delay of compute units, resulting in gains in throughput-per-mm^2 and throughput-per-watt. Representing numbers in low-precision also reduces memory-footprint and communication bandwidth requirement, enabling training of larger models for a given on-chip memory capacity and I/O budget. We experimented with 16-, 12-, and 10-bit representation for floating-point numbers. In the training experiments, only the matrix multiplication (GEMM) computations were performed in reduced precision. Since convolution operations are also cast as GEMM operations, this covers over 90% of all computations. We observed little to no degradation in the training error (cross-entropy) or test error when we reduce the precision to 12bits for CIFAR-10 dataset on MNIST. Since the logic complexity of floating point units (FPUs) is roughly proportional to the square of number of bits, we expect 12-b FPUs to be 7x-8x smaller and more power efficient than 32-b FPUs. Such large gains at the building-block level are critical to the success of hardware accelerators over general-purpose CPUs and GPUs.

In Convolutional Neural Network (CNN), there is significant redundancy in the convolution operation. Thus, we randomly skipped the computation of some of the locations during every convolution operation. For these points, we substituted data from the nearest valid computation. Randomly scrambling the maps for the convolution in each forward and backward convolution ensures that there is little degradation in accuracy over training performed with un-perforated convolution layers.

In order to exploit maximum parallelism, DNN training tasks are typically distributed across multiple accelerator nodes (or learners) – by either partitioning the training minibatch data across multiple accelerator nodes (data parallelism) or by distributing the DNN model across multiple nodes (model parallelism). Although DNN training benefits from using multiple learners in parallel, traditional stochastic gradient descent techniques require the parameter server to wait for an update from each of the learning nodes (Hardsync). Using an in-house DNN framework (Rudra), the results of applying variations of the asynchronous stochastic gradient descent algorithm (Softsync) on model training time and accuracy was presented [7]. This work demonstrated the need to reduce the mini-batch size per learner in proportion to the number of learners in order to maintain model accuracy in large-scale data-parallel DNN training tasks.

To harness the compounded benefits of reduced precision computations, loop perforation and relaxed communication techniques, we explored the impact of applying multiple approximation techniques concurrently on DNN accuracy, as shown in Fig. 1. These experiments showed that it is possible to simultaneously reduce the numerical precision (down to 12-bits) as well as apply loop perforation and relaxed synchronization schemes (for a 50% speedup in execution time) – with incurring little to no degradation additional degradation in the quality of the DNN model.

Motivated by these results demonstrating the resilience of DNNs to a combination of approximation techniques, we explored more opportunities to exploit approximate computing specifically to address challenges in accelerating large scale deep learning tasks. In the next sections we showcase two of the techniques we pursued, namely, gradient compression for minimizing communication overhead of distributed deep learning training ([5]) and activation quantization for minimizing computation cost for deep learning inference ([6]).

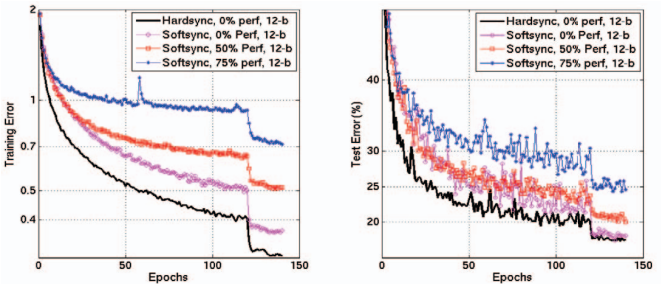

Fig. 1: Training and Test Error for the CIFAR10 with varying degree of perforation and synchronization schemes in the convolution layers. [4]

III. Gradient Compression for Data-Parallel Distributed Training

DNNs achieve superior accuracy through the use of very large and deep models - necessitating up to ExaOps of computation during training (on GPUs) and GB's of model and data storage. In order to improve the training time over single-node systems, distributed algorithms [8], [9], [10] are frequently employed to distribute the training data over multiple CPUs or GPUs - using data parallelism [7], model parallelism [11] and pipeline parallelism [2] techniques. Data parallelism techniques [7] are widely applied to distribute convolutional layers while

model and pipeline parallelism approaches are more effective for fully-connected and recurrent layers of a neural net. All 3 techniques necessitate very high interconnect bandwidth between the GPUs (in order to communicate the necessary parameters) and impose limits on peak system utilization and model training time. More specifically, as the number of learners increases, distribution of the minibatch data under strong scaling conditions has the adverse effect of significantly increasing the demand for communication bandwidth between the learners while proportionally decreasing the FLOPs needed in each learner - creating a severe computation to communication imbalance.

Simultaneously, there has been a renaissance in the computational throughput (TeraOps per second) of DL training accelerators - with accelerator throughputs exceeding 100's of TeraOps/s expected in the next few years [12], [13]. Exploiting hardware architectures based on reduced precision [14], [15], these accelerators promise dramatic reduction in training times in comparison to commercially available GPUs today. For a given DL network and distribution approach, the bandwidth needed for inter-accelerator communication (in GB/s) scales up directly with raw hardware performance as well as the number of learners. In order to guarantee high system performance, radically new compression techniques are therefore needed to minimize the amount of data exchanged between accelerators. Furthermore, the time required for compression needs to be significantly smaller than the computational time required for back-propagation. In this section, we introduce a novel computationally-friendly gradient compression scheme, based on simple local sampling, called AdaComp [5]. We show that this new technique, remarkably, self-adapts its compression rate across mini-batches and layers. We also demonstrate that the new technique results in a very high net compression rate ($\sim$200$\times$ in FC and LSTM layers and $\sim$40$\times$ in convolution layers), with negligible accuracy and convergence rate loss across several network architectures (CNNs, DNNs, LSTMs), data sets (MNIST, CIFAR10, ImageNet, BN50, Shakespeare), optimizers (SGD with momentum, ADAM) and network-parameters (mini-batch size and number of learners).

A. AdaComp: Adaptive Residual Gradient Compression

Given the popularity of distributed training of deep networks, a number of interesting techniques for compressing FC weight gradients have been proposed [16], [17], [18]. All the three techniques above are aimed at reducing weight update traffic in fully-connected layers. One common principle that allows these compression techniques to work without much loss of accuracy is that each learner maintains an accumulated gradient (that we refer to as residual gradients) comprising of the gradients that have not yet been updated centrally.

However, the gradient compression techniques proposed in these papers have several limitations. Seide [16] proposed a one-bit quantization scheme for gradients, which limits achievable compression rate for FC layers by 32x. Strom [17] proposed a thresholding technique for FC layers that can provide much higher compression rates. Only gradient values that exceed a given threshold are quantized to one bit and subsequently propagated. Dryden [18] proposed a technique combining the one-bit quantization and thresholding ideas. Since they propagate a fixed percentage of the gradients, this technique requires sorting of the entire gradient vector which is a computationally expensive task, particularly on a special-purpose accelerator. All the three techniques above are aimed at reducing weight update traffic in deep multi-layer perceptrons composed of fully-connected layers.

To overcome such challenges, we introduce AdaComp that can be applied to both convolution and fully-connected layers of DNNs. The key insight behind AdaComp is that it is critical to consider both input feature activities as well as accumulated residual gradients for maximum compression. This is accomplished very effectively through a local sampling scheme combined with an adjustable (soft) threshold, which automatically handles variations in layers, mini-batches, epochs, optimizers, and distributed system parameters (i.e., number of learners).

As we observe that there is lack of correlation between the activity of the input features and the residual gradients in any layer, we conjecture that it is important to have a small enough sampling window that can effectively capture the right residues across the entire layer. To facilitate this, we divide the entire residue vector for each layer (laid out as $NumOutMaps \times NumInpMaps \times KernelRows \times KernelCols$) uniformly into several bins - where the fixed length bin size, $\mathbf{L_T}$ is a new hyper-parameter. In each bin, we first find the maximum of the absolute value of the residues. In addition to this value, we found that it was also important to send several other residues that are relatively similar in magnitude to this maximum. There are several ways to find such important gradients inside each bin. In this paper, we propose a relatively simple self-adjusting scheme. Recall that in each mini-batch, the residue is computed as the sum of the previous residue and the latest gradient value obtained from back-propagation. If the sum of its previous residue plus the latest gradient multiplied by a scale-factor exceeds the maximum of the bin, we include these additional residues in the set of values to be sent (and centrally updated). Empirically, we studied a range of choices for the scale factor (from 1.5 - 3.0$\times$) and chose 2$\times$ primarily for computational ease (simple additions vs. multiplications). The primary intuition here is that since the residues are empirically much larger than the gradients, this scheme allows us to send a whole list of important residues close to the local maximum. Furthermore, we quantize the compressed residue vector in order to increase the overall compression rate. AdaComp is applied to every layer separately - and each learner sends a scale-factor in addition to the compressed sparse vector.

This approach to "threshold" the selection is self-adjusting in 3 ways. First, it allows some bins to send more gradients than others - as many as are needed to accurately represent each bin. Secondly, since the residues are small in the early epochs, more gradients are automatically transmitted in comparison with later epochs. Third, as will be shown in later sections, in comparison to other schemes, this technique minimizes the chances of model divergence that result from an explosion in the residual gradient values and gradient staleness. Thus, AdaComp adaptively adjusts compression ratios in different mini-batches, epochs, network layers and bins. These characteristics provide automatic tuning of the compression ratio, resulting in robust model convergence. We observe that just one hyper-parameter ($\mathrm{L_T}$) is sufficient to achieve high compression rates without loss of accuracy. Finally, it should also be noted that AdaComp,

TABLE I: Summary of AdaComp performance for various DNNs (CNN, MLP, and LSTM) [5].

Compression hyper-parameters: convolution layer L_T: 50 and fully connected layer L_T: 500							
Model	MNIST-CNN	CIFAR10-CNN	AlexNet	ResNet18	ResNet50	BN50-DNN	LSTM
Dataset	MNIST	CIFAR10	ImageNet	ImageNet	ImageNet	BN50	Shakespeare
Mini-Batch size	100	128	256	256	256	256	10
Epochs	100	140	45	80	75	13	45
Baseline (top-1)	0.88%	18%	42.7%	32.41%	28.91%	59.8%	1.73%
Our method (top-1)	0.85% (8L)	18.4%(128L)	42.9%(8L)	32.87%(4L)	29.15%(4L)	59.8% (8L)	1.75% (8L)
Learner number	1,8	1,8,16,64,128	8	4	4	1,4,8	1,8

unlike [18], does not require any sorting (or approximations to sorting) and is therefore computationally very efficient (O(N)) for high-performance systems.

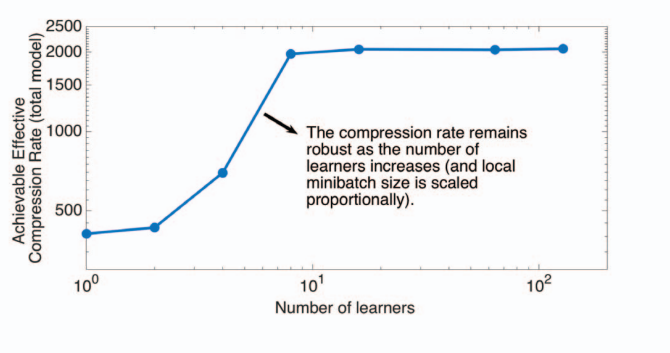

Fig. 2: Impact of number of learners on the compression rate for CIFAR10-CNN [5]. Compared to previous work [18], AdaComp shows higher compression rate for large number of learners - while maintaining test error degradation within the acceptable range (<1%).

B. Experiment Results

To demonstrate the robustness as well as the wide applicability of the proposed AdaComp scheme, we tested it comprehensively on all 3 major kinds of neural networks: CNNs, DNNs, and LSTMs. For CNN, five popular networks for image classification were tested: MNIST-CNN, CIFAR10-CNN, AlexNet, ResNet18, and ResNet50. We also included in our tests two pure DNNs (BN50-DNN for speech and MNIST-DNN (not shown)), and an RNN (LSTM). In all these experiments we used the same hyper-parameters as the baseline (i.e., no compression). The selection of L_T is empirical and is a balance between communication time and model accuracy; the same values are used across all models: L_T is set to 50 for convolutional layers and to 500 for FC and LSTM layers.

The experimental results are summarized in Table I. The proposed AdaComp scheme, on every single network (with different datasets, models and layers - CNNs, DNNs and LSTMs), tested under a wide range of distributed system settings (from 1 to 128 learners), achieved almost identical test errors compared with the non-compressed baseline.

Overall, our experimental results indicate that the AdaComp scheme is remarkably robust across application domains, layer types, learner numbers, and the choice of the optimizer. For the above L_T choices of 50 and 500, the AdaComp algorithm typically selects only up to 5 elements within each bin (through sparsity). For L_T sizes <64, a sparse-indexed representation of 8-bits could be used effectively, while 16-bits of representation would be needed for larger L_T sizes (up to 16K elements) - where 2-bits (out of 8 or 16) would be used to represent the ternarized data values. Therefore, in comparison with traditional 32-bit floating-point representations, the AdaComp scheme achieves an excellent **Effective Compression Rate** of $\mathbf{40}\times$ for convolutional layers and $\mathbf{200}\times$ for fully connected and recurrent layers.

Furthermore, as shown in Fig. 2, we empirically observe that the achievable compression rate for CIFAR10-CNN dramatically scales with the number of learners in the distributed system (while proportionally reducing the mini-batch size per learner). With more learners, each learner sees a smaller local mini-batch size and therefore compression rate is enhanced due to lower feature activity.

IV. Parameterized Clipping Activation for Quantized Neural Networks

Reducing the bit-precision of DNN data structures, namely weights and activations, has gained attention for alleviating DNN's extensive computational costs, thanks to its potential to significantly reduce both storage requirements and computational complexity. In particular, several weight quantization techniques ([19] and [20]) showed significant reduction in the bit-precision of CNN weights with limited accuracy degradation. However, prior work ([21], [22]) has shown that a straightforward extension of weight quantization schemes to activations incurs significant accuracy degradation in large-scale image classification tasks such as ImageNet ([23]). Recently, activation quantization schemes based on greedy layer-wise optimization were proposed ([24], [25], [26]), but achieved limited accuracy improvement.

Quantization of weights is equivalent to discretizing the hypothesis space of the loss function with respect to the weight variables. Therefore, it is indeed possible to compensate weight quantization errors during model training [27], [28]. Traditional activation functions, on the other hand, do not have any trainable parameters, and therefore the errors arising from quantizing activations cannot be directly compensated using back-propagation. Activation quantization becomes even more challenging when ReLU (the most common activation function in CNNs) is used. ReLU allows gradient of activations to propagate through deep layers and therefore achieves

TABLE II: Comparison of top-1 accuracy between DoReFa-Net [22] and PACT [6]. Weights are quantized with DoReFa-Net scheme, whereas activations are quantized with PACT.

Network	FullPrec	DoReFa-Net		PACT	
		2b	4b	2b	4b
CIFAR10-CNN	0.916	0.882	0.905	0.897	0.913
AlexNet	0.551	0.536	0.549	0.550	0.557
ResNet18	0.702	0.626	0.681	0.644	0.692
ResNet50	0.769	N/A	N/A	0.722	0.765

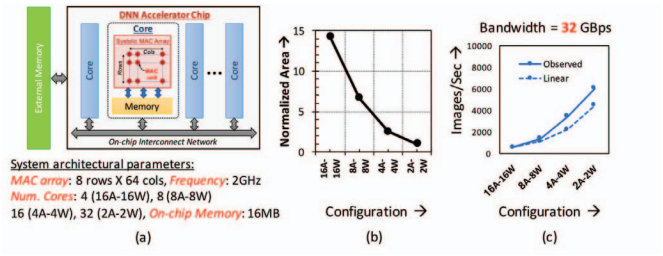

Fig. 3: (a) System architecture and parameters, (b) Variation in MAC area with bit-precision and (c) Speedup at different quantizations for inference using ResNet50 DNN.

superior accuracy relative to other activation functions ([29]). However, as the output of the ReLU function is unbounded, the quantization after ReLU requires a high dynamic range (i.e., more bit-precision).

A. PACT: Parameterized Clipping Activation Function

Building on these insights, we introduce PACT ([6]), a new activation quantization scheme in which the activation function has a parameterized clipping level, α. α is dynamically adjusted via gradient descent-based training with the objective of minimizing the accuracy degradation arising from quantization.

More specifically, in PACT, the conventional ReLU activation function in CNN is replaced with the parameterized clipping function, with the clipping level α limiting the dynamic range of activation to $[0, \alpha]$. The truncated activation output is then linearly quantized to k bits for the dot-product computations. With this new activation function, α is a variable in the loss function, whose value can be optimized during training. For back-propagation, gradient can be computed using the Straight-Through Estimator (STE) [30]. The larger the α, the more the parameterized clipping function resembles ReLU. To avoid large quantization errors due to a wide dynamic range, we include an L2-regularizer for α in the loss function.

B. PACT Performance for Quantized CNNs

In this section, we demonstrate that although PACT targets activation quantization, it does not preclude us from using weight quantization as well. We used PACT to quantize activation of CNNs, and DoReFa-Net scheme [22] to quantize weights. Table II summarizes top-1 accuracy of PACT for the tested CNNs (CIFAR10-CNN, AlexNet, ResNet18, ResNet50). We also show the accuracy of CNNs when both the weight and activation are quantized by DoReFa-Net's scheme. As can be seen, with 4 bit precision for both weight and activation, PACT achieves full-precision accuracy consistently across the networks tested. To the best of our knowledge, this is the lowest bit precision for both weight and activation ever reported, that can achieve near ($\leq 1\%$) full-precision accuracy.

C. System-level Performance Gain

In this section, we demonstrate the gain in system performance as a result of the reduction in bit-precision achieved using PACT. To this end, as shown in Fig. 3(a), we consider a DNN accelerator system comprising of a DNN accelerator chip, comprising of multiple cores, interfaced with an external memory. Each core consists of a 2D-systolic array of fixed-point multiply-and-accumulate (MAC) processing elements on which DNN layers are executed. Each core also contains an on-chip memory, which stores the operands that are fed into the MAC processing array.

To estimate system performance at different bit precisions, we studied different versions of the DNN accelerator each comprising the same amount of on-chip memory, external memory bandwidth, and occupying iso-silicon area. First, using real hardware implementations in a state of the art technology (14 nm CMOS), we accurately estimate the reduction in the MAC area achieved by aggressively scaling bit precision. As shown in Fig. 3(b), we achieve $\sim$14$\times$ improvement in density when the bit-precisions of both activations and weights are uniformly reduced from 16 bits to 2 bits.

Next, to translate the reduction in area to improvement in overall performance, we built a precision-configurable MAC unit, whose bit precision can be modulated. The peak compute capability (FLOPs) of the MAC unit varied such that we achieve iso-area at each precision. Note that the total on-chip memory and external bandwidth remains constant at all precisions. We estimate the overall system performance using DeepMatrix, a detailed performance modelling framework for DNN accelerators [31].

Fig. 3(c) shows the gain in inference performance for the ResNet50. We study the performance improvement using different weight and activation bit precision for the DNN accelerator system with memory bandwidth constrained at 32 GBps. Surprisingly, it exhibits a super-linear growth in performance with quantization. For example, quantizing from 16 to 4 bits (when PACT achieves full precision accuracy, Table II) leads to a 4$\times$ increase in peak FLOPs but a 4.5$\times$ improvement in performance. This is because, the total amount of on-chip memory remains constant, and at very low precision some of the data-structures begin to fit within the memory present in the cores, thereby *avoiding* data transfers from the external memory. Consequently, in bandwidth limited systems, reducing the amount of data transferred from off-chip can provide an additional boost in system performance beyond the increase in peak FLOPs.

V. Conclusion

The superior accuracy of deep learning for a wide spectrum of challenging application domains such as image processing, machine translation and speech recognition comes at the expense of high computational complexity in storage, energy, area, and communication costs which directly limit the pace of

innovation and adoption. In this paper, we discuss how these costs can be mitigated via approximate computing. Exploiting the resilience of DNNs to numerical errors from approximate computing, we demonstrate reduction in communication overhead of distributed deep learning training via adaptive residual gradient compression (AdaComp), and reduction in computation cost for deep learning inference via Prameterized clipping ACTivation (PACT) based network quantization. Experimental results show that order of magnitude savings in communication overhead in training as well as computational costs in inference can be achieved without compromising application accuracy.

ACKNOWLEDGMENT

The authors would like to thank Naigang Wang, Daniel Brand, Ankur Agrawal, Wei Zhang, Pierce I-Jen Chuang and I-Hsin Chung for helpful discussions and supports. This research was supported by IBM Research AI, IBM SoftLayer, and IBM Cognitive Computing Cluster (CCC).

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The microsoft 2016 conversational speech recognition system," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5255–5259.

[4] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 2016, pp. 1–8.

[5] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp : Adaptive residual gradient compression fordata-parallel distributed training," in *AAAI*, To appear, pp. 3–9.

[6] Anonymous, "Pact: Parameterized clipping activation for quantized neural networks," *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=By5ugjyCb

[7] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 171–180.

[8] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system." in *OSDI*, vol. 14, 2014, pp. 571–582.

[9] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[10] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[12] L. Durant, O. Giroux, M. Harris, and N. Stam, "Inside volta: The world's most advanced data center gpu," in *https://devblogs.nvidia.com/parallelforall/inside-volta/*, 2017.

[13] C. Merriman, "Google announces tpu 2.0 with 180 teraflop max out for ai acceleration," in *https://www.theinquirer.net/inquirer/news/3010365/google-announces-tpu-20-with-180-teraflop-max-out-for-ai-acceleration*, 2017.

[14] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1737–1746.

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[16] 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 *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[17] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[18] N. Dryden, S. A. Jacobs, T. Moon, and B. Van Essen, "Communication quantization for data-parallel training of deep neural networks," in *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*. IEEE Press, 2016, pp. 1–8.

[19] F. Li and B. Liu, "Ternary Weight Networks," *CoRR*, vol. abs/1605.04711, 2016.

[20] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," *International Conference on Learning Representations (ICLR)*, 2017.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *CoRR*, vol. abs/1609.07061, 2016.

[22] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *CoRR*, vol. abs/1606.06160, 2016.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[24] E. Park, J. Ahn, and S. Yoo, "Weighted-Entropy-Based Quantization for Deep Neural Networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[25] B. Graham, "Low-Precision Batch-Normalized Activations," *CoRR*, vol. abs/1702.08231, 2017.

[26] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep Learning With Low Precision by Half-Wave Gaussian Quantization," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[27] K. Hwang and W. Sung, "Fixed-point Feedforward Deep Neural Network Design Using Weights +1, 0, and -1," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2014, pp. 1–6.

[28] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

[29] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *27th International Conference on Machine Learning (ICML)*, pp. 807–814, 2010.

[30] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *CoRR*, vol. abs/1308.3432, 2013.

[31] S. Venkataramani, J. Choi, V. Srinivasan, K. Gopalakrishnan, and L. Chang, "POSTER: Design Space Exploration for Performance Optimization of Deep Neural Networks on Shared Memory Accelerators," *Proc. PACT 2017*.