Partition Pruning: Parallelization-Aware Pruning for Deep Neural Networks

Sina Shahhosseini, Ahmad Albaqsami, Masoomeh Jasemi++, and Nader Bagherzadeh

University of California,Irvine
{sshahhos,aalbaqsa,mjasemi,nader}@uci.com
++Jasemi@ce.sharif.edu

Abstract. Parameters of recent neural networks require a huge amount of memory. These parameters are used by neural networks to perform machine learning tasks when processing inputs. . To speed up inference, we develop Partition Pruning, an innovative scheme to reduce the parameters used while taking into consideration parallelization. We evaluated the performance and energy consumption of parallel inference of partitioned models, which showed a 7.72x speed up of performance and a 2.73x reduction in the energy used for computing pruned layers of TinyVGG16 in comparison to running the unpruned model on a single accelerator. In addition, our method showed a limited reduction some numbers in accuracy while partitioning fully connected layers.

Keywords: Parallelization · Deep Neural Network · Pruning · Partitioning · Hardware Accelerator.

1 Introduction

Neural networks have become ubiquitous in applications that include computer vision, speech recognition, and natural language processing. The demand for processing neural network applications on edge devices, including smart phones, drones, and autonomous vehicles, is increasing [1]. Meanwhile, the size of neural network models has been drastically increased over time, reaching beyond the Peta scale [1]. In 1998, a handwritten digits classifier had about 1 M parameters [2], but in 2012, an image classifier for the ImageNet [3] dataset had more than 60 M parameters. In addition, Neural Talk, which automatically creates proper captions for ImageNet dataset has more 230 M parameters [4]. The top 5 error accuracy has been reduced by 30% each year, suggesting why this trend drastically increases the number of layers, parameters, and operations [1].

Large deep neural networks (DNNs) models consume a significant amount of energy because they are required to be stored in DRAMs or on-chip SRAMs, and thus are fetched every time they are processed. From 2012 to 2015, the energy efficiency of DRAMs increased due to CMOS scaling based on Moore's Law. As of 2015, CMOS scaling no longer provided substantial improvements in either energy efficiency or memory density. Because SRAM is realized using CMOS transistors, its energy efficiency is typically bounded by Moore's Law [18] [19]. Therefore, the energy efficiency of the memory cannot keep up with the increasing size of the neural networks. This leads to consuming more energy to accomplish the same processing tasks. Therefore,

innovations in architectural design, algorithms development, and circuit technique are required [5].

Both memory footprint and computational complexity lead to the need for sparsity and/or reducing the number of parameters in a neural network. For example, AlexNet requires 234 MB of memory space for storing parameters and requires 635 million arithmetic operations for feed-forward processing. AlexNet's convolutional layers are locally connected, but they are followed by fully connected layers that make up 95% of the connections in the AlexNet network [6]. Fully connected layers are over-parameterized and tend to overfit the training data. At the algorithm level, pruning methods were proposed before deep learning became popular. Based on the assumption that many parameters are unnecessary, pruning methods remove these parameters, resulting in expanding sparsity of layers [7].

Previous research has sought to reduce the number of parameters. Dropping out random connections was proposed by [11]. The Optimal Brain Damage [12] and Optimal Brain Surgeon [13] reduced the number of connections according to the loss function. Singular value decomposition (SVD) decreased the number of weights [14]. Another approach , adopted by the GoogleNet model [15], exploits the convolutional layers rather than the fully connected layers. This resulted in sparse layers that provided three benefits [16]. First, sparse layers required less storage for space for parameters. Second, it omitted computation of the removed edges, which reduced power consumption and latency. Third, it required less memory bandwidth to transfer parameters from the DRAM.

In this paper based on the insight that smart pruning can reduce the number of off-chip accesses, we propose a new scheme to better partition and prune the inputs to each layer. This way, we partition a large matrix into small matrices and distribute them to multiple computational units. The proposed partitioning algorithm has three objectives: first enhancing the parallelism among accelerators, second reducing the number of off-chip accesses, and third maintaining the accuracy as high as the baseline. In the first step, we formulate the problem and then enforce some constraints. We define our constraint in such a way that the three mentioned objectives are satisfied. .The experimental results show that the proposed scheme can increase the speed up by 7.72x and energy efficiency by 2.73x, respectively.

The rest of this paper is organized as follows. Section. 2 provide an overview of the problem. Section 3 describes the proposed partition pruning algorithm followed by Multi-core organization in Section. 4. Experimental setup and evaluation methodology is presented in Section. 5. We discuss the result in Section. 6. And finally, we conclude the paper in Section. 7.

2 Overview

Figure 1 illustrates a high-level diagram of the proposed framework. First, a neural network model is trained. Section V discusses the baseline accuracy for different neural network models that were used to evaluate the framework. Then, fully connected layers of each model were pruned using the Partition Pruning approach. Section III explains how the partitioning algorithm was applied to these layers. Then, inference was

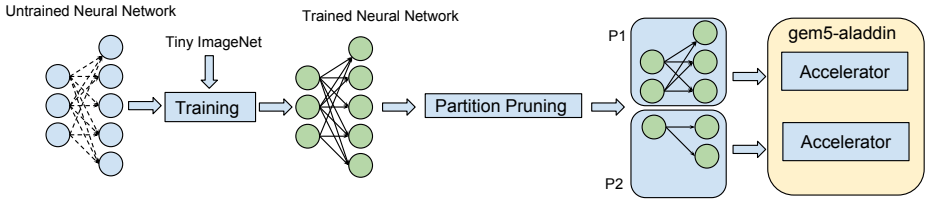

Fig. 1. Overview of the procedure used. Note that Partition Pruning is applied to a trained neural network since it is dependent on the weights of the fully connected layer(s). The illustration shows only one fully connected layer.

performed on multiple processing cores. Section IV explains multi-core architecture, which provides the ability to run parallel matrix multiplication. Section VI evaluates our framework in terms of performance and accuracy.

3 Partition Pruning

3.1 System Model

Our framework targets neural networks that have some or all of their nodes fully connected to the subsequent nodes. The *set* of starting nodes, $\boldsymbol{N_{initial}}$ is *fully* connected to the subsequent nodes $\boldsymbol{N_{final}}$, i.e. *fully-connected layers*. A link, which is a parameter, is a connection represented by $\boldsymbol{L_{ij}}$, where $\boldsymbol{i}$ is the starting node number and, $\boldsymbol{j}$ is the connected node number within a layer. The link's value (i.e the parameter's weight) is represented by $\boldsymbol{w_{i,j}}$. $\boldsymbol{L_{i,j} = 0}$ if the link is pruned, and if not, $\boldsymbol{L_{i,j} = 1}$. Note that $\boldsymbol{w_{i,j}}$ may contain any value. The set of weights, $\boldsymbol{W_i}$, consists of links, $\boldsymbol{L_i}$, that connect between the set of Nodes, $\boldsymbol{N_i}$, and $\boldsymbol{N_j}$. Figure 2a shows an example of a fully connected layer of size $\boldsymbol{6 \times 8}$. Figure 2b shows the matrix representation of the fully connected layer. While Figure 2c indicates the weight matrix of the fully connected layer. The *connectedness number*, $\boldsymbol{C}$, is simply;

$$\boldsymbol{C} = \sum_{\boldsymbol{i=1}}^{|\boldsymbol{N_{initial}}|} \sum_{\boldsymbol{j=1}}^{|\boldsymbol{N_{final}}|} \boldsymbol{L_{i,j}} \tag{1}$$

A fully connected layer is annotated as $\boldsymbol{C_{full}}$ and thus;

$$\boldsymbol{C_{full}} = |\boldsymbol{N_{initial}}| \times |\boldsymbol{N_{final}}| \tag{2}$$

Therefore, the *connectedness ratio*, $\boldsymbol{R}$, is:

$$\boldsymbol{R} = \frac{\boldsymbol{C}}{\boldsymbol{C_{full}}} \tag{3}$$

Figure 3 shows an example of a 2-partition pruning of the fully connected layer from Figure 2. Figure 4 visually illustrates the partitions of Fig 3 and the reduction of number of weights due to that partitioning. Given that there are $|\boldsymbol{P}|$ partitions, where a

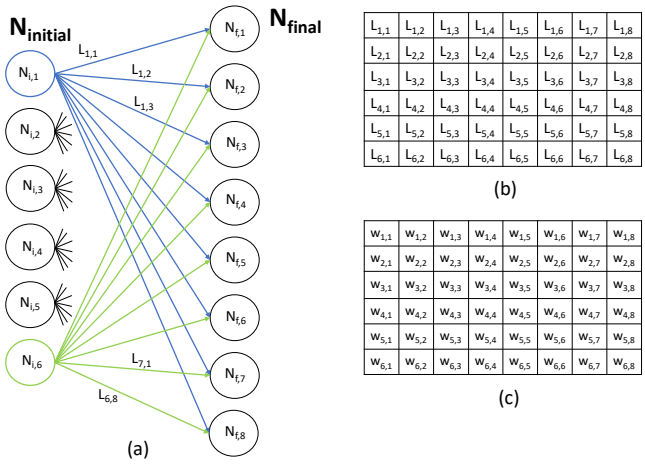

Fig. 2. Example of a model representation of a fully connected layer. b) shows the connection's representation in matrix form. Note that in the above case,$\boldsymbol{C = C_{full} = 6 \times 8 = 48}$ and $\boldsymbol{R = 1}$.

$\boldsymbol{P_x \in P}$, then any given $\boldsymbol{N_{initial,j} \in P_x}$ will not be in any other partition. The same goes for nodes in $\boldsymbol{N_{final,i}}$. More formally,

$$\{\boldsymbol{P_i, P_j \in P | i \neq j, P_i \cap P_j = \emptyset}\} \tag{4}$$

Equation 4 is the constraint of the groupings of nodes in $\boldsymbol{N_{initial}}$ and $\boldsymbol{N_{final}}$. That is, once a particular node is in a particular partition, it cannot be a member of another partition. Another way of stating this is:

$$\boldsymbol{N_i \in P_n} \text{ Then } \boldsymbol{N_i \notin P_m, \forall m \neq n} \tag{5}$$

Note that there is an upper, $\left\lceil \frac{|\boldsymbol{N_{initial}}|}{|\boldsymbol{P}|} \right\rceil$, and lower, $\left\lfloor \frac{|\boldsymbol{N_{initial}}|}{|\boldsymbol{P}|} \right\rfloor$, bound to the number of $\boldsymbol{N_{initial,i}}$ nodes that are members of a partition $\boldsymbol{P_n}$. The same is true for $\boldsymbol{N_{final,i}}$ nodes. In addition, the number of partitions that contain the upper limit is $|\boldsymbol{N_{initial}}| \mod |\boldsymbol{P}|$, while the number that contain the lower limit is $|\boldsymbol{P}| - (|\boldsymbol{N_{initial}}| \mod |\boldsymbol{P}|)$.As an example, if $|\boldsymbol{N_{initial}}| = \boldsymbol{22}$ and $|\boldsymbol{P}| = \boldsymbol{5}$ (i.e number of partitions), then an example of partition sizes for $\boldsymbol{N_{initial}}$, ignoring $\boldsymbol{N_{final}}$, would be

$$(|\boldsymbol{P_1}|, |\boldsymbol{P_2}|, |\boldsymbol{P_3}|, |\boldsymbol{P_4}|, |\boldsymbol{P_5}|) = \boldsymbol{(4, 5, 4, 4, 5)}$$

Therefore, the example suggests that there are three partitions of size 4 and two partitions of size 5. This bound description also applies to $\boldsymbol{N_{final}}$.

3.2 Partition Pruning Overview

The objective of Partition Pruning is two-fold: pruning with the objective of having balanced partitions, and pruning with the objective of having the least absolute weight-

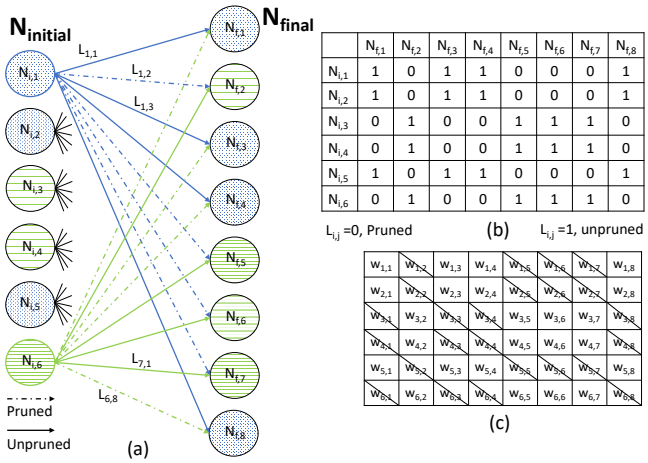

Fig. 3. a) indicating what links are to be pruned from the fully connected layer b) shows the connection's representation, with 0s representing the absence of a link. Note that in the above case,$\boldsymbol{C = C_{full} = 12}$ and $\boldsymbol{R = 0.5}$.

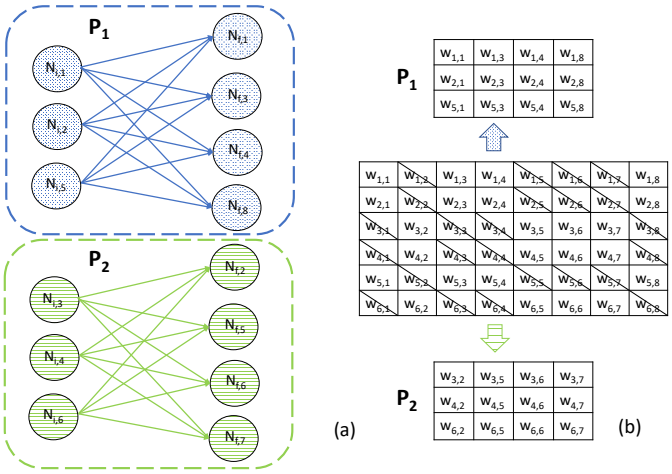

Fig. 4. a) the resulting partitions shows full independence. b) shows the resulting reduction of parameters due to the 2-partition targeted pruning.

loss. The second objective guarantees a smaller loss of accuracy, while the first allows for maximum parallelism. Note that the number of parameters pruned is directly related to the number of partitions desired. The connectedness ratio, in relation to the number of partitions is $\boldsymbol{R_{|P|} = \frac{1}{|P|}}$. Thus, for a given $\boldsymbol{|P|}$, Partition Pruning will find the

following:

$$\begin{aligned} \min_{x} \quad & |C_{full} \sum |w_{i,j}| - \sum x_{i,j}|w_{i,j}|| \\ \text{subject to} \quad & x_{i,j} = 0 \text{ or } 1 \\ & \sum x_{i,j} = R_{|P|}C_{full} \\ & \{P_m, P_n \in P | n \neq m, P_m \cap P_n = \emptyset\} \end{aligned}$$

From the objective function, we determine which $1-R_{|P|}C_{full}$ parameters are pruned for a particular fully connected layer while minimizing the cumulative weight-loss.

3.3 Input/Output

The input to the Partition Pruning algorithm is a matrix representation, $W_{fc,i}$, of the targeted fully connected layer, i. This is exemplified in Figure 2c. Note that the fully connected layer is assumed and asserted to be trained. That is, the parameters have the correct values for the targeted neural network's base accuracy. In a fully connected layer, every element of the matrix $L_{fc,i}$ is 1 (see Equation 2). After Partition Pruning, the output will be $L_{part,i}$ and the sum of all its elements would be RC_{full}. This is exemplified in Figure 3b.

3.4 Methodology

This section the methodology of selecting the links to prune, taking into consideration the partitioning. The example of $|N_{initial}| = 7$, $|N_{final}| = 10$, and $|P| = 3$, will be used to describe the process. Figure 1 shows an overview of the methodology and where Partition Pruning resides.

Start: Selection of $N_{initial,i}$, and $N_{final,j_1,j_2..}$:
In the first stage, a row in the matrix is *randomly* selected. That is, a random $N_{initial,i}$ is selected for processing. Note that currently $|P_n| = 0$ for all n, because no pair of nodes, has joined a partition. After choosing an $N_{initial,i}$, a set of N_{final} nodes is chosen, and in this case, the set size is $\left\lceil \frac{|N_{final}|}{|P|} \right\rceil$. The node $N_{initial,i}$, and the nodes $N_{fianl,j_1,j_2..}$ are chosen to be part of the first partition, P_1. Those selected will have their $L_{i,j} = 1$, while those not selected will have their $L_{i,j'} = 0$. Note that the links selected have the *highest magnitudes* (refer to Figure 5a as an example). Figure 5b illustrates an example of the change in values and a pictorial representation of the first partition.

Non-Start: Selection:
Moving forward, another $N_{initial,i}$ node is selected at random. The highest, non-partition members, $w_{i,j}$s are sorted from the highest to the lowest magnitude, as was done previously. The sum of the highest upper bound (or a lower bound if all upper bound partitions are fulfilled) are compared with the sum of the magnitude of partition-member weights/links that still have capacity (as per the upper and lower bounds of the number of nodes of type $N_{initial}$).

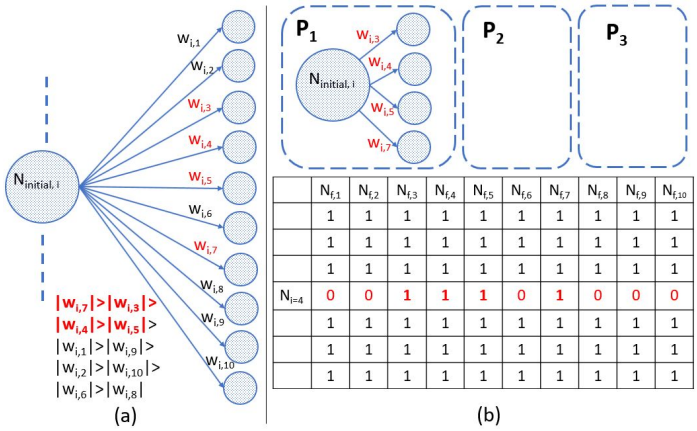

Fig. 5. Random selection of $N_{initial,i}$, where $i = 4$ in this example. The top four weights, in terms of magnitude, are $w_{i,7}, w_{i,3}, w_{i,4}$, and $w_{i,5}$ in descending order. Note that its top *four* because of the upper bound, $\lceil |N_{final}|/|P| \rceil = \lceil 10/3 \rceil = 4$ b) P_1, after partitioning, contains four nodes (the limit) from N_{final}, and one node from $N_{initial}$.The L matrix is updated for row i=4

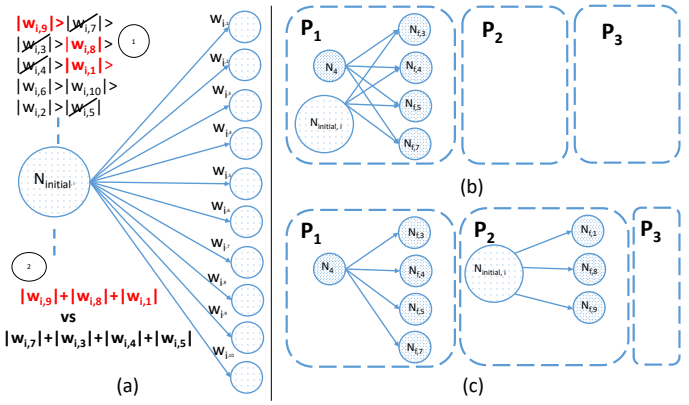

Fig. 6. second random selection of $N_{initial,i}$ (where $i \neq 4$). The top three weights (1), in terms of magnitude and are none partition members, are $w_{i,9}, w_{i,8}, w_{i,1}$, in descending order. Note that its top *three* due to the the capacity for N_{final} node type is $(P_1, P_2, P_3) = (4, 3, 3)$ b) shows the situation in case of $|w_{i,7}| + |w_{i,3}| + |w_{i,4}| + |w_{i,5}| > |w_{i,9}| + |w_{i,8}| + |w_{i,1}|$. c) is the case scenario.

End and Try Again:

This process is repeated until every partition P_m, is at capacity in terms of $N_{initial}$ nodes and N_{final} nodes. Note that the partitioning is dependent on which row, i.e $N_{initial,i}$ was selected at each iteration. Once the process is completed, the weight-loss is recorded.

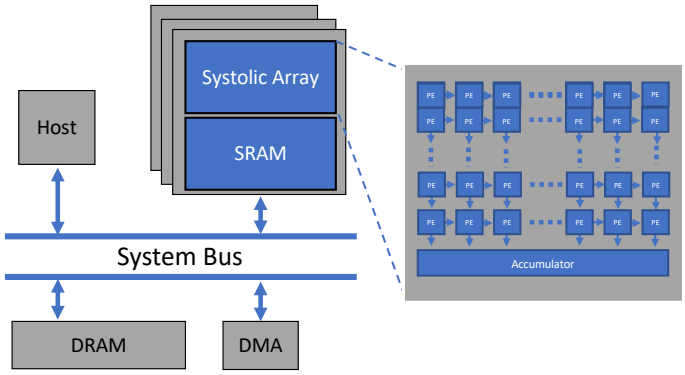

Fig. 7. Architectural template for generated accelerators.

4 Multi-Core Organization

Figure 7 shows the architecture of an System on Chip (SoC) that consists of general purpose cores, memory controllers, a DMA engine, and matrix multiplication accelerators all of which are connected through the system bus. To understand how the system level affects the accelerators' behavior, simulation infrastructures that can model these heterogeneous systems are needed. gem5-Aladdin system simulator is used to evaluate the proposed architecture. This tool is an integration of a gem5 system simulator with an Aladdin accelerator simulator. It is a pre-RTL simulation infrastructure that models multiple accelerators and interactions with central processing units (CPUs) in an SoC that consists of Processing Elements (PEs), fixed-function accelerators, memory controllers, and interfaces. This simulator can model the accelerators' performance, area, and power [27][28]. Multiple matrix multiplication units are connected to the bus. In the gem5-Aladdin system, the accelerators can invoke the DMA engine already present in the Gem5. The DMA is used to transfer bulk data without the CPU's intervention. The internal SRAM stores the weights, input features, and the outputs of the matrix multiplication. Each accelerator uses a 32 x 32 Systolic Array (SA). The SA architecture is a specialized form of parallel computing in which tightly coupled processing elements are connected to a small number of their nearest neighbors in a mesh-like topology. This architecture has a very low amount of global data transfer and can achieve a high clock frequency. However, SA architecture suffers from scalability issues due to the shape being fixed.

In an SA, the horizontal systolic movements are for implementing data broadcasts, and the vertical ones are for implementing accumulations.

5 Experimental Setup

Fully connected layers are pruned by using Partition Pruning for three networks that use a TinyImageNet [23] dataset. which consists of 100,000 training images, 10,000 validation images, and 10,000 testing images that have dimensions of 64x64x3, and

Table 1. System Configuration Parameters

Parameter	Value
Host Clock Frequency	1 GHz
Accelerator Clock Frequency	200 MHz
Technology Width	40 nm
DRAM	DDR3-1600-8x8
Number of CPU	1
Systolic Array Size	32x32
Data Type	FP-32
Data Transfer	DMA

Table 2. Baseline Top-5 and Top-1 accuracy for VGG16, AlexNET

Network Name	Top-5 Accuracy	Top-1 Accuracy
TinyVGG16	76.96%	52.41%
TinyAlexNet	72.06%	46.73%

that classify 200 labels. These images are taken from the ImageNet [3] dataset, cropped into squares, and resized to 64x64. For each network, the fully connected layers are partitioned to 2, 3, 4, and 5 partitions, resulting in the pruning of 50%, 66%, 75%, and 80%, of the fully connected links, respectively.

Initially, the neural networks are trained and evaluated on a TinyImageNet dataset, as shown in Table 2. Convolutional neural networks represent the state-of-the-art in image classification. AlexNet [24] and VGG16 [2] are well-known deep convolutional neural networks that have previously won ImageNet competitions. TinyVGG16 and TinyAlexNet use a 56x56x3 input image instead of 228x228x3, as do the original VGG16 and AlexNet. Each network has three fully connected layers at the end its structure. Partition Pruning prunes the first two of these three fully connected layers. The omission of pruning the last fully connected layer is due to the fact that every link is required for classification. If pruned, the classification accuracy would be affected the considerably and detrimental to the performance of the Neural Network model. Table 2 shows the benchmarks' baseline performances. After training the networks, Partition Pruning is applied to two, of the three, fully connected layers. Google's TensorFlow [25] version 1.7 was used to model the benchmarks. Partition Pruning was implemented in Python 2.7 and was given the NumPy matrices from the first two fully connected layers of the benchmarks. Then, the weights were updated in the TensorFlow model files using the resulting output filters. Note that, as mentioned earlier, gem5-Aladdin is used to evaluate the performance.

6 Results

Table 2 shows the initial baseline accuracies, without pruning, of the TensorFlow implementations of the neural network benchmarks. Figure 8 shows the resulting accuracy losses of the Partition Pruning algorithm for TinyVGG16 and TinyAlexNet. Note that

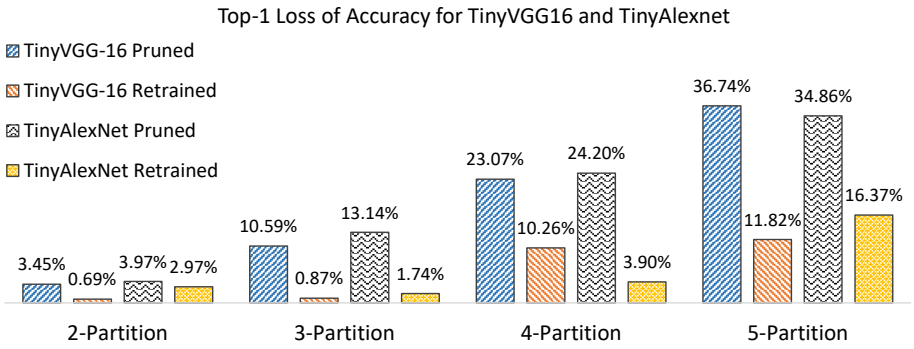

Fig. 8. Top-1 loss of Accuracy for VGG16 and Alexnet. Note that the number of links pruned is equal within the partition group. In retraining, only the non-pruned links are retrained.

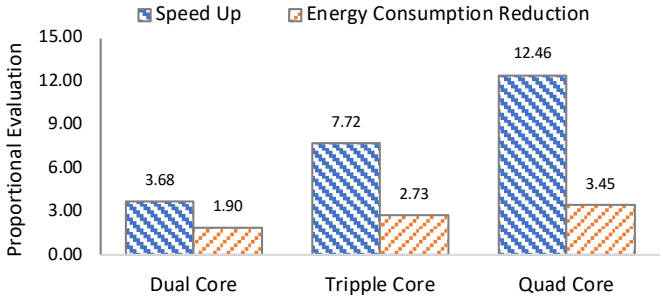

Fig. 9. Power and performance of multi-accelerators are shown. Results are evaluated proportional to result of unpruned benchmark on single core accelerator.

results for retraining are also shown. Accuracy loss increases when the number of partitions is increased, given that more parameters are pruned. After pruning, retraining the models reduces the loss of accuracy. For example, in 3-Partition, retraining reduces accuracy loss in TinyVGG16 from 10.59% to 0.87%. As Figure 7 shows, running inference of partitioned TinyVGG16 layers on different accelerators speeds performance and reduces energy consumption. These results are in comparison to running inference of the unpruned layers on signle accelerator. For example, running this benchmark on a triple-core accelerator executes 7.72x faster while consuming 2.73x less energy. This is because pruning reduces the size of the benchmarks by a factor correlated to the partition number (for example, by a factor of 2x for two partitions). In addition, running inference in parallel on multiple accelerators speeds the execution time. Therefore, the performance speed and the energy consumed by processing partitioned models were both improved by reducing the size of the models and using multiple hardware resources. Running the same benchmarks on multiple accelerators does not increase speed as expected. For example, running two identical workloads on two accelerators can increase speed 1.8x, and on three accelerator, 2.5x. This happens because all accelerators are connected to the same bus with one DMA, which leads to bus congestion. It is expected that using multiple large SAs, for example 256 x 256, would cause bandwidth

bottlenecks and sizeable bus congestion. Although using a small SA does not provide high throughput processing, it leads to low power design because of the number of processing elements used in each accelerator.

7 Conclusions

This paper presented Partition Pruning, an approach that prunes fully connected layers of neural network models with the aim of partitioning for parallelization in order to improve speed and energy. The idea behind Partition Pruning approach is to target low overall weight loss to reduce the impact on accuracy. The approach shows that by partitioning fully dense layers of TinyVGG16 to 3-Partition and executing the model on multiple accelerators, a speed increase of 7.72x and an energy reduction of 2.73x can be obtained. Future work will evaluate a system that has multiple high-bandwidth memories and neural network accelerators. In addition, more optimizations will be applied to the accelerators to minimize power consumption and increase throughput.

References

1. Xu, X. et al. Scaling for edge inference of deep neural networks. Nature Electronics 1, 216 (2018) in press.
2. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR, vol. abs/1409.1556, 2014.
3. A. Krizhevsky I. Sutskever G. E. Hinton "Imagenet classification with deep convolutional neural networks" NIPS 2012
4. Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
5. Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. ArXiv e-prints.
6. S. Han, H. Mao, and W. Dally. Deep compression: Compressing DNNs with pruning, trained quantization and huffman coding. arXiv:1510.00149v3, 2015a
7. R. Reed, Pruning algorithms-a survey, Neural Networks, IEEE Transactions on, vol. 4, no. 5, pp. 740747, 1993.
8. Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature
9. Jouppi, N. P. et al. In-datacenter performance analysis of a Tensor Processing Unit. Proc. 44th Annu. Int. Symp. Comp. Architecture Vol. 17 112 (2017)
10. Schmidhuber, Jrgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.
11. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, vol. 15, no. 1, pp. 19291958, 2014
12. Hanson, Stephen Jose and Pratt, Lorien Y. Comparing biases for minimal network construction with back-propagation. In Advances in neural information processing systems, pp. 177185, 1989
13. LeCun, Yann, Denker, John S, Solla, Sara A, Howard, Richard E, and Jackel, Lawrence D. Optimal brain damage. In NIPs, volume 89, 1989

14. Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In NIPS, pages 12691277, 2014
15. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. arXiv preprint arXiv:1409.4842, 2014
16. Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, et al, "Minerva: Enabling Low-Power Highly-Accurate Deep Neural Network Accelerators", Computer Architecture (ISCA) 2016 ACM/IEEE 43rd Annual International Symposium on, pp. 267-278, 2016
17. Ahmad Albaqsami, Maryam S. Hosseini,and Nader Bagherzadeh, "HTF-MPR: A Heterogeneous TensorFlow Mapper Targeting Performance using Genetic Algorithms and Gradient Boosting Regressors". 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)(pp. 331-336).IEEE.
18. Shahosseini, Sina, et al. "Dependability evaluation of siso control-theoretic power managers for processor architectures." 2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC). IEEE, 2017.
19. Shahhosseini, Sina, et al. "On the feasibility of SISO control-theoretic DVFS for power capping in CMPs." Microprocessors and Microsystems 63 (2018): 249-258.
20. M. G. Augasta and T. Kathirvalavakumar, Pruning algorithms of neural networksa comparative study, Central European Journal of Computer Science, vol. 3, no. 3, pp. 105115, 2013
21. Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems. 2015.
22. B. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (2) (1970) 291307
23. Hansen, Lucas. "Tiny imagenet challenge submission." CS 231N (2015).
24. Iandola, Forrest N., et al. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size." arXiv preprint arXiv:1602.07360 (2016).
25. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, and et. al, Tensorflow: A system for large-scale machine learning, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265283.
26. Reagen, Brandon, et al. "Minerva: Enabling low-power, highly-accurate deep neural network accelerators." ACM SIGARCH Computer Architecture News. Vol. 44. No. 3. IEEE Press, 2016.
27. Shao, Yakun Sophia, et al. "Co-designing accelerators and soc interfaces using gem5-aladdin." Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016.
28. Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, Aladdin: A pre-rtl, power performance accelerator simulator enabling large design space exploration of customized architectures, in Proceeding of the 41st
29. Pezeshkpour, Pouya, Liyan Chen, and Sameer Singh. "Embedding multimodal relational data." (2017).