Balanced Column-Wise Block Pruning for Maximizing GPU Parallelism

Cheonjun Park[1], Mincheol Park[1,2], Hyun Jae Oh[3], Minkyu Kim[1], Myung Kuk Yoon[4], Suhyun Kim[2], and Won Woo Ro[1]*

[1]Yonsei University
[2]Korea Institute of Science and Technology
[3]Samsung Electronics
[4]Ewha Womans University
{cheonjun.park, mincheol.park, minkyu.kim, wro}@yonsei.ac.kr, hyunjae.oh@samsung.com, myungkuk.yoon@ewha.ac.kr, dr.suhyun.kim@gmail.com

Abstract

Pruning has been an effective solution to reduce the number of computations and the memory requirement in deep learning. The pruning unit plays an important role in exploiting the GPU resources efficiently. The filter is proposed as a simple pruning unit of structured pruning. However, since the filter is quite large as pruning unit, the accuracy drop is considerable with a high pruning ratio. GPU rearranges the weight and input tensors into tiles (blocks) for efficient computation. To fully utilize GPU resources, this tile structure should be considered, which is the goal of block pruning. However, previous block pruning prunes both row vectors and column vectors. Pruning of row vectors in a tile corresponds to filter pruning, and it also interferes with column-wise block pruning of the following layer. In contrast, column vectors are much smaller than row vectors and can achieve lower accuracy drop. Additionally, if the pruning ratio for each tile is different, GPU utilization can be limited by imbalanced workloads by irregular-sized blocks. The same pruning ratio for the weight tiles processed in parallel enables the actual inference process to fully utilize the resources without idle time. This paper proposes balanced column-wise block pruning, named BCBP, to satisfy two conditions: the column-wise minimal size of the pruning unit and balanced workloads. We demonstrate that BCBP is superior to previous pruning methods through comprehensive experiments.

Introduction

To achieve higher accuracy in deep neural networks (DNNs), recent studies proposed the use of deeper networks with larger datasets (Simonyan and Zisserman 2015). Better accuracy in DNNs is achieved at the cost of higher computational complexity; therefore, more hardware resources are required (Sze et al. 2017). Recent graphic processing unit (GPU) architectures have successfully provided a massive amount of computational parallelism and can be leveraged in DNNs (NVIDIA 2022). However, the computation load still needs to be reduced, and pruning can be an effective solution to reduce the amount of computation as well as the memory capacity requirement (Han et al. 2015).

Various pruning techniques (Han et al. 2015; He, Zhang, and Sun 2017; Li et al. 2017; Wen et al. 2016) have been developed to reduce the number of nonzero parameters and computational overhead. However, unstructured pruning solely focuses on reducing weights (Han et al. 2015), which incurs memory access irregularity and degrades memory access efficiency (Kung et al. 2019; Hill et al. 2017; Yu et al. 2017). In contrast, structured pruning (He, Zhang, and Sun 2017) reduces weights maintaining memory access regularity by removing closely located weights together. Their highly regular patterns can leverage existing general matrix multiplication (GEMM) kernels so that they can utilize the GPU's computational resources effectively. However, due to the large size of the pruning unit, the accuracy is seriously reduced as the pruning ratio increases.

The pruning unit size should be selected considering two main factors: to minimize the loss of representation power, and to consider the actual feedforward process inside the device. To satisfy the above conditions, we thoroughly delved into GPU optimization methods when computing DNNs, such as lowering (im2col) (Chellapilla, Puri, and Simard 2006; Chetlur et al. 2014) and tiling (Kirk and Wen-Mei 2016). First, while computing convolutions on GPUs, the input and weight tensors are flattened and transformed into a matrix by lowering (im2col) their values. Second, as shown in Figure 1, tiling divides the matrices into small equal-sized matrices (block) and distributes them to streaming multiprocessors (SMs), which are independent processing units

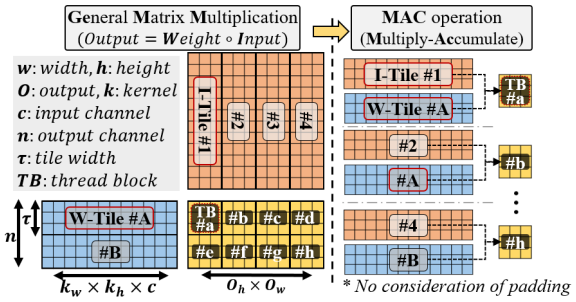


Figure 1: Tiling method for computing DNN on GPU which is provided by NVIDIA cuBLAS (Nvidia 2008). W-Tile denotes weight tile. I-Tile denotes input tile.

*Corresponding author.

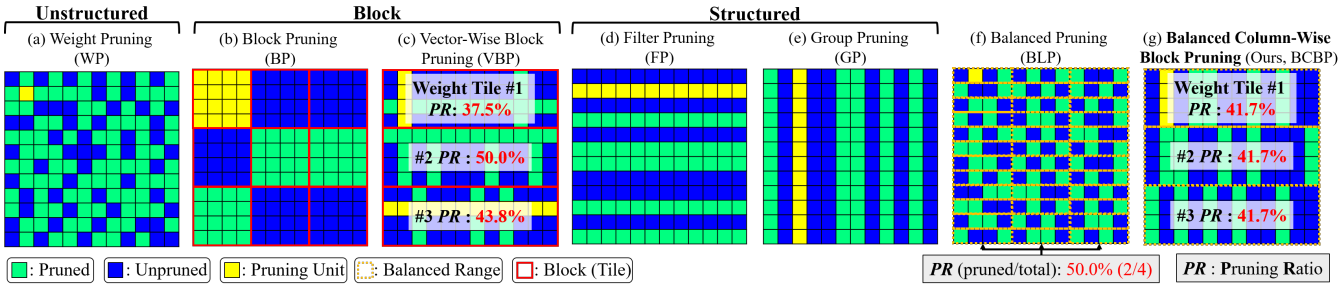


Figure 2: The illustration of different types of pruning in GEMM-view.

of GPUs. Because the separated matrices easily fit into the shared memory of SMs, tiling can reduce global memory access by maximizing data reuse in shared memory and register files (Kirk and Wen-Mei 2016). Two kinds of optimizing the GEMM kernels present a lesson for the optimal pruning unit size (Guo et al. 2020). Considering tiling, vector-wise pruning for each block (tile) reduces the pruning unit size to tile width. This tiling-based pruning method (Guo et al. 2020) is a kind of block pruning. However, considering pruning unit size alone cannot maximize GPU utilization, workload balancing among the SMs is critically required.

Our proposed workload balancing aims to ensure no idle time over the SMs when deploying the pruned model. For this purpose, we revisit the tiling-based approach in the GPU. Vector-wise block pruning (VBP) (Guo et al. 2020) proposed a hybrid technique known as cross-wise that performs row-wise and column-wise pruning for each block simultaneously. VBP carefully considered two GPU optimizations (im2col and tiling).

However, such a hybrid method (cross-wise) can be challenging due to the row-wise approach. For instance, pruning a row affects the corresponding tiled-matrix column in the following layer; thereby, deciding a column to be pruned can be complicated and lean on the row-wise pruned results. Moreover, this method does not consider the same sparsity over the blocks. This causes a load unbalancing problem. To address these problems, first, we use only column-wise pruning for each block, which is a simple but effective method. With this approach, a vector-wise shape is maintained, and it can effectively utilize the computing resource of GPUs. Second, a new metric called workload imbalance among tiles (*WIT*) is proposed to verify the workload imbalance between tiles. As *WIT* increases, the workload imbalance problem becomes more serious. Thus, the pruning ratio of each tile must be adjusted to reduce the *WIT*. Additionally, we propose a pruning ratio recalibration technique that considers pruning sensitivity for each block to reduce information loss due to the same pruning ratio per block.

This paper proposes the balanced column-wise block pruning, BCBP, satisfying three key factors at the same time: optimal size of pruning unit, weight shape regularity, and balanced workloads. Our method performs layer-wise pruning before acceleration based on the API.

This method has three advantages:

- Our BCBP method can preserve more representation power than structured pruning because the pruning unit size is smaller than structured pruning.
- The row-wise vector pruning in blocks results in the removal of entire columns in the following layer, which conflicts with the column-wise pruning of the next layer and degrades the representation power. Therefore, we propose using only column-wise pruning in blocks to minimize accuracy drop.
- BCBP has the same pruning ratio per tile. The same pruning ratio on all tiles prevents SMs in the GPU from being idle and harmonizes the execution time.

Related Work

Unstructured Pruning (Weight Pruning) Weight pruning (WP) removes superfluous weights to reduce the parameters and the computations of DNNs (Figure 2-(a)). WP (Han et al. 2015) uses a threshold to measure the important connections. However, while achieving a significantly high pruning ratio, this cannot avoid retraining the original model to recover the accuracy drop. In order to achieve a more favorable model compression, Deep Compression (Han, Mao, and Dally 2016) uses weight quantization and Huffman encoding, as well as WP. However, these techniques require specialized basic linear algebra subprograms (BLAS) such as cuSPARSE (Naumov et al. 2010), or a dedicated accelerator (Han et al. 2016; Parashar et al. 2017) for swift sparse tensor computation. In the absence of such support units, data irregularity arising from the sparse weights is fatal on GPUs utilization. To achieve a high pruning ratio, additional indices are necessary to indicate removed weight individually. These induce indirect access patterns. WarpPool (Kloosterman et al. 2015) reports that such patterns experience a memory divergence that requires more than one memory request to fetch the data from memory, resulting in a low throughput of the GPU. A high pruning ratio can be expected and counterbalanced when applying WP over a tile. However, multiple indices cause unnecessary complexity owing to the memory divergence. Therefore, we use relatively larger pruning granularity to reduce the risk of rapid processing time.

Structured Pruning Structured pruning typically removes rows (filter pruning) or columns (group pruning) in lowered weight matrices (Figure 2-(b) and (c)). Performing filter pruning (FP) leaves behind a subset of filters via salient

criteria, for example, $l1$-norm, over a layer (Li et al. 2017) or on the whole model (Yu et al. 2018). In addition, owing to the large size of the pruning unit, layer-wise methods encounter cumulative reconstruction errors (Kim et al. 2020). A layer collapse cannot be avoided when FP is performed on the whole model (Tanaka et al. 2020). Sparsity regularization is introduced on features (He, Zhang, and Sun 2017) or BatchNorm variables (Liu et al. 2017) to remove the corresponding filters during training. OTO (Chen et al. 2021) attempts to generate a structured pruned model without a pre-training step by determining non-critical filters at the network initialization phase. However, the performance of the method depends on data adaptation. Group pruning involves pruning the weights located in the same position among all the filters of a certain layer. This model can maintain its learned representation as it has a smaller pruning unit than FP. However, it can damage the representation ability of the model with a high pruning ratio as it breaks the assumption that filters are independent (Li et al. 2017). Our work focuses on using group pruning; however, we separate such coupled weights by exploiting tiling. Our method leverages tiling by constraining the smaller pruning unit from the tile width to prevent the removal of important weights, thereby providing an opportunity to achieve a high pruning ratio.

Block Pruning Block pruning consists of two kinds of approaches: block pruning (BP) and vector-wise block pruning (VBP). One of the BPs (Narang, Undersander, and Diamos 2017; Vooturi, Mudigere, and Avancha 2018), Block Sparse (Narang, Undersander, and Diamos 2017), proposes a technique for removing the entire block, supported by a dedicated GPU kernel (Gray, Radford, and Kingma 2017) for speedup on real hardware (Figure 2-(d)). Second, VBP (Cai et al. 2021; Elsen et al. 2020; Guo et al. 2020; Li et al. 2021; Lin et al. 2022) is a method of removing a row or column vector of a block (Figure 2-(e)). $1 \times N$ pruning (Lin et al. 2022) can achieve speedups on CPUs via a parallelized block-wise vectorized operation. Tile Sparsity (Guo et al. 2020) has improved inference throughput on GPUs. This method utilizes the "tiling" of the GPU kernel operations when it prunes neural networks. As illustrated in Figure 2-(e), Tile Sparsity (Guo et al. 2020) presented both rows and columns pruning (cross-wise pruning) in each tile-shaped matrix. However, it is difficult to acquire the same pruning ratio over all the tiles in vector-wise pruning, and unequal sparsity can cause imbalanced computation workloads in GPUs (Gale et al. 2020). Our study considers tiling; however, it ensures that the same pruning ratio is obtained in each block (tile). Therefore, our pruning method enables all workloads to be computationally balanced on GPUs. The results are demonstrated in the experiments.

Balanced Pruning Balanced pruning (BLP) (Kung, McDanel, and Zhang 2019; Mishra et al. 2021; Yao et al. 2019; Zhou et al. 2021) splits the row of a lowered weight matrix into multiple equal-sized blocks and removes the same number of weight elements for each block (Figure 2-(f)). Therefore, whole blocks can have balanced computations because each block is pruned with the same pruning ratio. Since the GPU executes an instruction with a warp (32-threads), if some pruned blocks are assigned in a warp, computation workloads among such blocks can be sufficiently synchronous. However, it is still required to process the massive indices on pruned matrices because the size of the pruned unit is typically small, as shown in Figure 2-(f). Prior works have co-designed customized API and a specialized accelerator in GPUs, such as the Sparse Tensor Core, to support the computation efficiently (Mishra et al. 2021). However, the A100 (Mishra et al. 2021) has a limit that can only accelerate on a 50% pruning ratio (2:4 sparsity pattern). As illustrated in Figure 2-(g), the proposed method splits equal-sized tiles of a lowered weight matrix. Our approach allows the computation of large granular blocks to be balanced and gains more freedom to find unnecessary weights. In this case, we remove the less critical grouped weights by applying group pruning with a tile-sized pruning unit. Therefore, our method requires smaller indices than balanced pruning and achieves the same pruning ratio among the tiles.

Balanced Column-Wise Block Pruning

We formally introduce how BCBP is performed on GPUs, following the order in Figure 3.

Prerequisites Given n filters of l^{th} convolution layer as $\mathcal{F}^{(l)} \in \mathbb{R}^{n \times c \times k_h \times k_w}$, where n, c, k_h, and k_w are the number of filters, the number of channels, height, and width of the filters, respectively. The lowered matrix of $\mathcal{F}^{(l)}$ can be denoted as $\mathbf{W}^{(l)} \in \mathbb{R}^{n \times m}$, where n and m are the number of filters and a whole dimension of a filter. Moreover, n can be a pruning unit of column-wise manner. m is the product of k_w, k_h, and c whose order is a lowering order of a filter.

Vector-Wise Tile Pre-Pruning

For simplicity, we introduce the vector-wise tile pre-pruning steps on l^{th} layer. Considering tiling of a lowered matrix, $\mathbf{W}^{(l)}$ is divided into multiple equal-sized tiles. It is denoted as $\mathbf{W}^{(l)} = \left\{ \widetilde{\mathbf{W}}_i^{(l)} \in \mathbb{R}^{\tau \times m}, i \in [1, T] \right\}$, where τ is tile width and T is the number of tiles. If n cannot be absolutely divided into T, this step of BCBP is disregarded and our pruning is converted as the group pruning.

In a certain i^{th} tile, $\widetilde{\mathbf{W}}_i^{(l)}$ for a partial set of grouped weights are denoted as $\widetilde{\mathbf{W}}_i^{(l)} = \left\{ \mathbf{w}_{i,k}^{(l)} \in \mathbb{R}^{\tau}, k \in [1, m] \right\}$. Note that each grouped weight, $\mathbf{w}_{i,k}^{(l)}$, is a column vector of a $\widetilde{\mathbf{W}}_i^{(l)}$. Our pruning criteria, denoted as $C^{(l)}(\cdot)$, is used to get the importance scores of the grouped weights in $\widetilde{\mathbf{W}}_i^{(l)}$. Then, we have scores over a tile as follows:

$$score_i^{(l)} = C^{(l)}(\widetilde{\mathbf{W}}_i^{(l)}). \tag{1}$$

where $score_i^{(l)} \in \mathbb{R}^m$ is the score vector of all grouped weights in l^{th} layer and i^{th} tile. Getting scores of total grouped weights in $\mathbf{W}^{(l)}$ can be formulated as:

$$C^{(l)}(\mathbf{W}^{(l)}) = \left\{ C^{(l)}(\widetilde{\mathbf{W}}_i^{(l)}) = \left\| \mathbf{w}_{i,k}^{(l)} \right\|_g, \forall (i,k) \right\}, \\ i \in [1, T],\ k \in [1, m]. \tag{2}$$

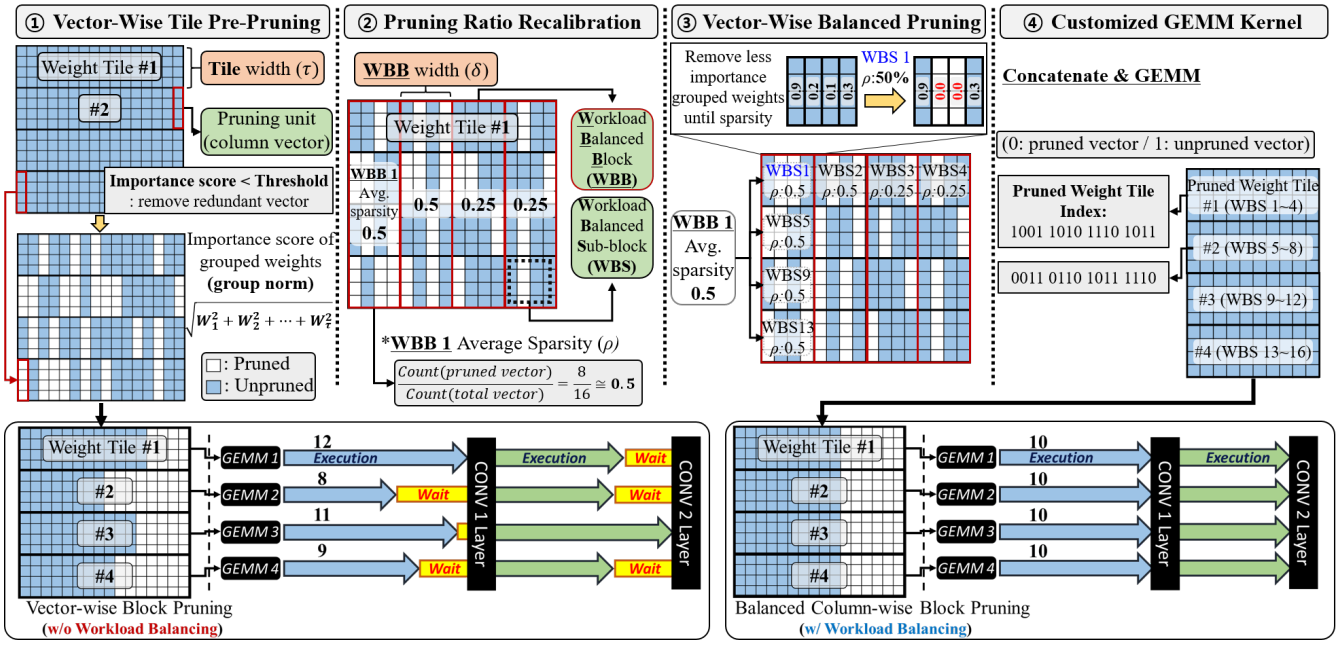


Figure 3: Overview of BCBP. BCBP proceeds in 3 steps: vector-wise tile pre-pruning, pruning ratio recalibration, and vector-wise balanced pruning. The GEMM operation with the unpruned weight parameters of BCBP and input values is accelerated using customized API.

where $\|\cdot\|_g$ is the group norm (Wen et al. 2016). The $\|\cdot\|_g$ what we inherit can be formulated as $\left\|\mathbf{w}_{i,k}^{(l)}\right\|_g = \sqrt{\sum_{x=1}^{|\mathbf{w}_{i,k}^{(l)}|}\left((w_{i,k}^{(l)})_x\right)^2}$, where $|\mathbf{w}_{i,k}^{(l)}|$ is the number of weights in $\mathbf{w}_{i,k}^{(l)}$, which is the same as τ in our work.

The vector-wise tile pre-pruning is conducted based on $C^{(l)}(\mathbf{W}^{(l)})$:

$$rid^{(l)} = \text{Topk}(C^{(l)}(\mathbf{W}^{(l)}), N^{(l)}). \tag{3}$$

where $N^{(l)}$ is the number of the grouped weights to be removed and Topk$(\cdot)$ returns the indexes of the k most unimportant grouped weights, using the scores. The indexes are stored in $rid^{(l)}$.

Pruning Ratio Recalibration

Pruning $\mathbf{W}^{(l)}$ based on $rid^{(l)}$ can occur imbalanced computation workloads in GPU as shown in Figure 3-①. To solve the problem, we consider one hyper-parameter, workload balanced block (WBB) width, to reset the pruning ratio inside a WBB. While adopting WBB width, a lowered matrix can be considered as $\mathbf{W}^{(l)} = \left\{\widetilde{\mathbf{W}}_j^{(l)} \in \mathbb{R}^{n\times\delta}, j \in [1, B]\right\}$, where δ is a WBB width, and B is the number of WBBs.

In a certain j^{th} WBB, $\widetilde{\mathbf{W}}_j^{(l)}$ for a partial set of weights are denoted as $\widetilde{\mathbf{W}}_j^{(l)} = \left\{\mathbf{w}_{j,k}^{(l)} \in \mathbb{R}^{\tau}, k \in [1, \delta T]\right\}$. To get an adjusting pruning ratio, we divide $rid^{(l)}$ into multiple subsets, $rid_j^{(l)}, j \in [1, B]$, corresponding to each WBB. By holding $rid^{(l)}$, this analysis can maintain the precomputed unimportant grouped weights of the layer. Then, the new pruning ratio for the set of WBB can be formulated as:

$$\rho_j^{(l)} = \frac{Count(rid_j^{(l)})}{Count(\widetilde{\mathbf{W}}_j^{(l)})}, j \in [1, B]. \tag{4}$$

In this paper, we use δ based on a spatial dimension of the convolution filter such as multiple of $k_h k_w$. We explain in detail performing processes of recalibration that minimize the accuracy loss while keeping the same pruning ratio between tiles in the Discussion Section.

Vector-Wise Balanced Pruning

To equalize the pruning ratio of tiles inside workload balanced blocks (WBB), we split WBB into equal-sized sub-blocks with tile width, defined as workload balanced sub-blocks (WBS). The lowered matrix with WBS can be denoted as $\mathbf{W}^{(l)} = \left\{\widetilde{\mathbf{W}}_{i,j}^{(l)} \in \mathbb{R}^{\tau\times\delta}, i \in [1, T], j \in [1, B]\right\}$, where τ is the tile width, δ is the WBB width, T is the number of tiles and B is the number of WBBs. A certain $(i, j)^{th}$ WBS, $\widetilde{\mathbf{W}}_{i,j}^{(l)}$, for a partial set of grouped weights are denoted as $\widetilde{\mathbf{W}}_{i,j}^{(l)} = \left\{\mathbf{w}_{i,j,k}^{(l)} \in \mathbb{R}^{\tau}, k \in [1, \delta]\right\}$. Note that each grouped weight, $\mathbf{w}_{i,j,k}^{(l)}$, is a column vector of a $\widetilde{\mathbf{W}}_{i,j}^{(l)}$.

For all WBSs in the same WBB, corresponding $\rho_j^{(l)}$ is identically applied; thereby, vector-wise balanced pruning is conducted as follows:

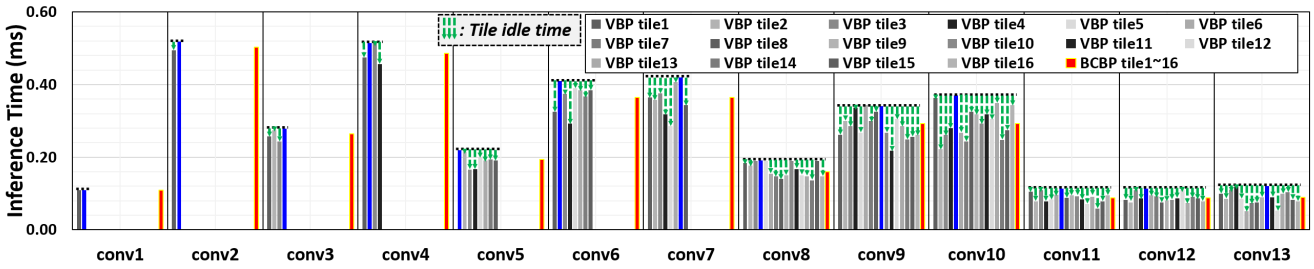


Figure 4: Measurement of inference time for a whole tiled matrix of VGG-16 on ImageNet with a 1.5% accuracy drop. BCBP uses τ of 32, which is the same as the VBP (Guo et al. 2020) and δ of 18. In addition, all layers have the same pruning ratio (PR) of 60% except for the first convolutional (conv) layer. The gray-scale bars are inference times of VBP tiles. The blue bars are the maximum inference time of VBP tile at each layer. The red bars are the inference time of BCBP tiles. All BCBP tiles have the same inference time at each layer.

$$rid_{i,j}^{(l)} = \text{Topk}(score_{i,j}^{(l)}, N_{i,j}^{(l)}),$$
$$\mathcal{K}_{i,j}^{(l)} = \text{Prune}(\widetilde{\mathbf{W}}_{i,j}^{(l)}, rid_{i,j}^{(l)}),$$
$$i \in [1, T],\ j \in [1, B]. \tag{5}$$

where $N_{i,j}^{(l)} = \delta \times \rho_j$ is the number of the grouped weights to be removed on the WBS and $score_{i,j}^{(l)}$ is newly acquired by computing $score_{i,j}^{(l)} = C^{(l)}(\widetilde{\mathbf{W}}_{i,j}^{(l)})$.

With $rid_{i,j}^{(l)}$, Prune$(\cdot)$ removes unimportant grouped vectors in a WBS, and remains the other ones. After this pruning method, $\mathcal{K}_{i,j}^{(l)}$ represents the remained grouped weights on the WBS.

Acceleration After all BCBP steps, two kinds of unpruned weights in i^{th} tile are obtained, such as the remained weights, $\mathcal{K}_{i,:}^{(l)}$, and the indexes of the removed weights, $rid_{i,:}^{(l)}$. In order to accelerate the GEMM of an input and the unpruned weight values, we exploit the customized GPU kernel based on the API (Guo et al. 2020), which supports the skipping function in GPU; thereby, GPU can recognize the pruned location and skip the load operation of the same positioned input values in advance.

Experiments

Datasets and Models We evaluate the performance of BCBP using IWSLT English-Vietnamese (Luong and Manning 2016), SQuAD (Rajpurkar, Jia, and Liang 2018), and ImageNet dataset (Deng et al. 2009). ImageNet contains 1.28M training images and 50K test images. For the validation of image classification, we assess our method with extensive convolutional deep neural network model: VGG-16 (Simonyan and Zisserman 2015), ResNet-18 (He et al. 2016), and ResNet-50. The additional verification of the following DNN models are shown in Appendix [1]: SqueezeNet (Iandola et al. 2016), GoogleNet (Szegedy et al. 2015), WRN-50 (Zagoruyko and Komodakis 2016), NMT (Luong, Brevdo, and Zhao 2017), and BERT-base (Devlin et al. 2018).

[1]https://github.com/cheonjun-park/appendix1.

Baseline Settings We evaluate BCBP using NVIDIA RTX 2080 TI GPUs. In our experiments, we use the pre-trained CNN models from PyTorch framework (Paszke et al. 2019). We perform fine-tuning with only 60 epochs after conducting pruning methods on ImageNet. For settings of the fine-tuning, we use SGD optimizer with the weight decay, 1×10^{-4} and the momentum as 0.9. We set a batch size of 256 and a learning rate of 0.0001.

Analysis of Workload Balancing

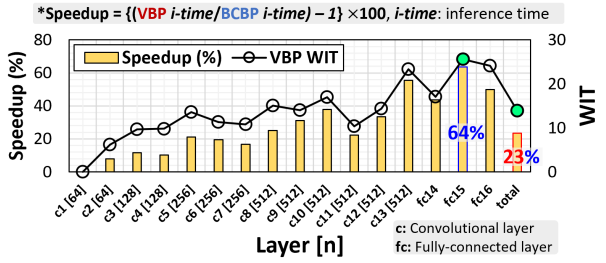
To analyze the computational imbalance problem, we propose a metric, *WIT*. In this section, various experiments are conducted to understand the correlation between our proposed *WIT* and the imbalance in actual workload. As the high *WIT* value indicates in workload imbalance between tiles, well-balanced pruning based on tiling can be the favored approach to improving the GPU computational efficiency.

Workload Imbalance among Tiles (*WIT*) To measure the workload imbalanced among tiles, we propose *WIT*. As shown in Figure 4, we empirically show that this metric can explain that unequal pruning ratios over the tiled weights cause unsynchronized SMs due to the variation of the execution time of each tile. Other tiles of VBP wait for the tile with the minimum pruning ratio to be completed. While, in BCBP, all tiles in a layer have the same inference time. Following the experimental results, the average of the imbalanced ratio is inductively represented by

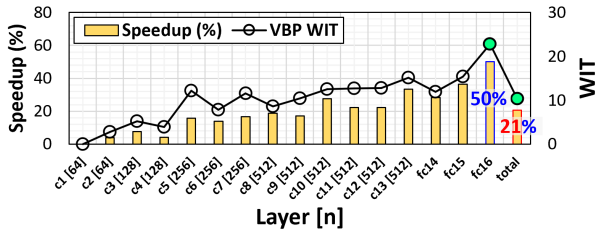
$$WIT = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} |\mathcal{PR}(x) - \mathcal{PR}(x^*)|,$$
$$x^* = \underset{x \in \mathcal{X}}{\text{argmin}}\, \mathcal{PR}(x). \tag{6}$$

where $\mathcal{X}$ represents the set of the tiled weights' index. $\mathcal{PR}(\cdot)$ performs calculating the pruning ratio by loading the weight at the corresponding input.

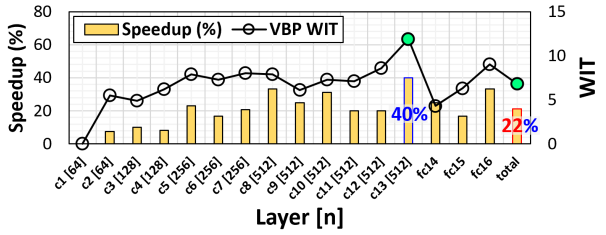
Tile-Width (τ) We verify BCBP on two tile widths (τ) because GPU has a fixed tile width. We use two magic numbers, 16 and 32, since NVIDIA GPU typically uses a multiple of 32 to set τ (Guo et al. 2020) and Mobile GPU, such



(a) VBP (PR: **60.6**%, τ: **16**) and BCBP (PR: **60.2**%, τ: **16**)



(b) VBP (PR: **60.2**%, τ: **32**) and BCBP (PR: **60.2**%, τ: **32**)



(c) VBP (PR: **80.0**%, τ: **32**) and BCBP (PR: **80.1**%, τ: **32**)

Figure 5: Measurement of VBP *WIT* and speedup of BCBP compared to VBP of VGG-16 on ImageNet within a 3% accuracy drop. δ of BCBP is 9. n is the number of filters. The *WIT* value of BCBP is 0 in all the layers because BCBP balances the workload between tiles. PR denotes pruning ratio.

as ARM Mali, sets τ as 16 (Harris 2014). Figure 5-(a) and (b) give a comparison in inference time and *WIT* with two τ settings: 16 and 32. Changing τ from 32 to 16 doubles the number of tiles (T) in almost every layer. When T is higher, the variance of pruning ratio is large; thus, *WIT* of VBP also increases. It presents that VBP makes imbalanced workloads relying on τ. As shown in Figure 5-(a) and (b), when τ is varied from 32 to 16, *WIT* of VBP rises from 10.4 to 13.9. However, BCBP shows balanced resource utilization and performs better than VBP in the same τ, δ, and PR.

Layer (l) In DNN architecture, channels typically increase as a layer, l, increases. As shown in Figure 5, VGG-16 has four-layer groups varying channel size, 64, 128, 256, and 512. Each group introduces different *WIT*, 4.83 ± 1.79, 6.61 ± 2.53, 10.05 ± 2.32 and 17.93 ± 4.26, respectively, which represents mean and standard deviation. Moreover, T is large as the number of channels increases. As a result, we can identify that *WIT* with the variance is generally larger as T increases. Thus, a layer with large channels is possible to make imbalanced workloads.

Model (Type)	Pruning unit size	PR (%)	Top-1 ↓(%)	$time$ (ms)	speed up
VGG-16	-	-	-	6.8	1.00×
STC (BLP)	$\mathbb{R}$	50.0	-0.88	3.5	1.94×
BLS (BLP)	$\mathbb{R}$	91.9	-1.31	5.2	1.31×
BS (BP)	$\mathbb{R}^{32\times32}$	71.6	-2.54	5.9	1.16×
TS (VBP)	$\mathbb{R}^{32\times1,1\times m}$	80.0	-2.32	3.0	2.28×
BCBP	$\mathbb{R}^{32\times1}$	49.8	**-0.97**	4.5	**1.52**×
BCBP	$\mathbb{R}^{32\times1}$	80.1	**-1.96**	2.5	**2.77**×

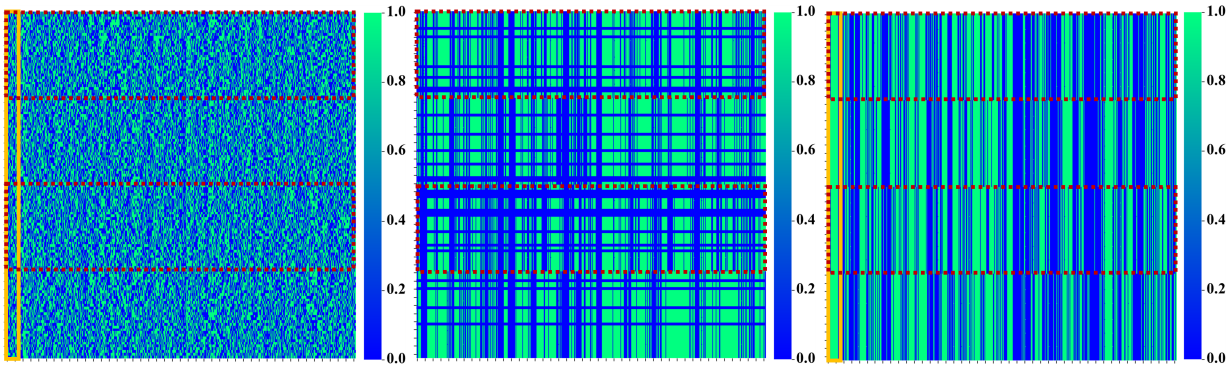
Table 1: Comparison of inference time (ms) with BLP, BP and VBP of VGG-16 on ImageNet. m is the product of k_w, k_h, and c whose order is a lowering order of a filter. PR denotes pruning ratio. Top-1↓ means the top-1 accuracy drop rate compared to dense model. The $time$ denotes inference time.

Pruning Ratio (PR) When PR increases, the lowered weight matrix on whole layers becomes too sparse. Therefore, when PR varies from 60% to 80%, the magnitude of *WIT* becomes small in general. Figure 5-(b) and (c) show that the *WIT* of VBP (Guo et al. 2020) decreases from 10.4 to 6.8 in the varying. Increasing PR can be another approach to make balanced workloads by reducing *WIT*. However, the result can sacrifice accuracy.

Validation of Perception Task

Comparison with BLP The pruning unit size of BCBP is a $\mathbb{R}^{32\times1}$. However, the pruning unit size of BLP is $\mathbb{R}$. Therefore, as shown in Table 1, Balanced Sparsity (BLS) (Yao et al. 2019) has the highest pruning ratio with negligible accuracy drop, which is about 11.8% higher pruning ratio than BCBP. Table 1 shows that BCBP (PR: 80.1%) is 2.11× faster than BLS (PR: 91.9%) using customized GPU kernel in VGG-16. In the case of BLP, due to a much longer index calculation time, the advantage of zero skip is insignificant. When the 50% pruning ratio of NVIDIA A100 Sparse Tensor Core (STC) (Mishra et al. 2021) and BCBP is similar in VGG-16, STC with hardware support is about 1.27× faster than BCBP. However, the STC has a limit that can only accelerate on a 50% pruning ratio (Mishra et al. 2021).

Comparison with BP and VBP The pruning unit size of Block Sparse (BS) (Narang, Undersander, and Diamos 2017) is a $\mathbb{R}^{32\times32}$. The pruning unit size of Tile Sparsity (TS) (Guo et al. 2020), which is both row and column pruning, is a $\mathbb{R}^{32\times1}$ and $\mathbb{R}^{1\times m}$. However, pruning unit size of BCBP, which is column pruning, is only $\mathbb{R}^{32\times1}$. Due to the 32× smaller pruning unit size than BS, Table 1 shows that BCBP can accommodate the higher pruning ratio than BS. In addition, BCBP is 21.1% faster than TS in VGG-16. Even with a similar pruning ratio of 80%, TS is slower than BCBP due to delay time in some weight tiles from imbalanced workloads. Furthermore, BCBP has even 0.36% less accuracy drop than TS.



(a) BLP (balanced range: 18) (b) VBP (τ: 32) (c) BCBP (τ: 32, δ: 18)

Figure 6: The pruning mask visualization for the third convolutional layer of VGG-16 on ImageNet. The mask is represented by the output of $\mathbb{1}_{\mathbf{W}^{(3)}}(x)$ where x is a weight value of $\mathbf{W}^{(3)}$. The pruned region is blue. The dotted red line denotes a tile. Rigid orange line denotes a WBB.

Model	PR (%)	Top-1 ↓(%)	$time$ (ms)	$Index\ ratio$	speed up
ResNet-18	-	-	7.3	-	1.00×
TAS	33.3	-1.50	4.9	-	1.48×
LCCL	34.6	-3.65	4.8	-	1.52×
BCBP	41.3	**-0.46**	4.6	4%	**1.59**×
FPGM	41.7	-1.87	4.3	-	1.68×
BCBP	84.7	**-2.09**	1.5	9%	**4.79**×
ResNet-50	-	-	12.5	-	1.00×
GAL-0.5	16.8	-4.20	10.6	-	1.18×
BCBP	29.8	**-0.72**	9.4	3%	**1.33**×
HRank	36.7	-1.17	8.3	-	1.51×
SFP	41.8	-1.54	7.8	-	1.61×
GAL-1	42.5	-6.27	7.7	-	1.63×
BCBP	45.9	**-1.26**	7.5	4%	**1.66**×
HRank	46.0	-4.17	7.2	-	1.74×
GAL-1-joint	59.9	-6.84	5.5	-	2.26×
ThiNet-50	66.0	-7.73	4.8	-	2.58×
HRank	67.5	-7.05	4.7	-	2.63×
BCBP	79.4	**-2.47**	3.5	6%	**3.59**×

Table 2: Comparison of inference time (ms) with filter pruning (FP) of ResNet-18 and ResNet-50 on ImageNet. PR denotes pruning ratio. Top-1↓ means the top-1 accuracy drop rate compared to dense model. The $time$ denotes inference time. $Index\ ratio$ denotes ratio of index computation kernel time in inference time.

Comparison with FP Predicting output in the pruning model of FPs (TAS (Dong and Yang 2019), LCCL (Dong et al. 2017), FPGM (He et al. 2019), GAL (Lin et al. 2019), HRank (Lin et al. 2020), SFP (He et al. 2018), Thinet (Luo, Wu, and Lin 2017)) requires only GEMM operation. However, BCBP requires additional index computation to perform GEMM operations. As shown in Table 2, the execution time of GEMM is similar when BCBP and FPGM have a similar pruning ratio of 41% in ResNet-18. However, since BCBP requires an addition 4% of index computation time, BCBP is about 6% slower than FPGM overall. As shown in Table 2, for the same reason as ResNet-18, when the 46% pruning ratio of HRank and BCBP is similar in ResNet-50, BCBP is about 5% slower than HRank. When the pruning ratio increases from 41.3% to 84.7%, the inference time decreases by 67.0% on ResNet-18. Due to the higher pruning ratio, the GEMM and indexing computation time are reduced by 68.7% and 25.7%, respectively. Despite the additional indexing time, BCBP overwhelms the overall performance because of the benefits of zero skipping compared to FP. This high pruning rate reduces accuracy to 2.09% however is similar to FP.

Pruning Mask Visualization

We illustrate how our balanced column-wise block pruning (BCBP) has regular data patterns and balanced workloads among tiles. In Figure 6, we visualize balanced pruning (BLP) (Yao et al. 2019), vector-wise block pruning (VBP) (Guo et al. 2020), and BCBP. We now define an indicator function to represent which weights are pruned in binary numbers as follows:

$$\mathbb{1}_{\mathbf{W}^{(l)}}(x) = \begin{cases} 1, & \text{if } x \in \mathcal{K}^{(l)} \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where $\mathbf{W}^{(l)}$ is a 2D weight matrix, which is a converted 4D filter weight of the convolutional layer by the lowering, $\mathcal{K}^{(l)}$ denotes the unpruned parameter set of $\mathbf{W}^{(l)}$, and l is the layer number. For the visualization, we use the pruning ratio of 50%. We exhibit each mask of BLP, VBP, and BCBP based on the aforementioned indicator function of $\mathbf{W}^{(l)}$. We use the third convolutional layer of VGG-16. Thus, $\mathbf{W}^{(3)}$ is used for mask visualization. Balanced pruning shows balance data patterns in a balanced range as shown in Figure 6-(a). In Figure 6-(b), the vector-wise block pruning shows regular data patterns but workload imbalance among tiles. BCBP has the same pruning ratio per tile as shown in Figure 6-(c).

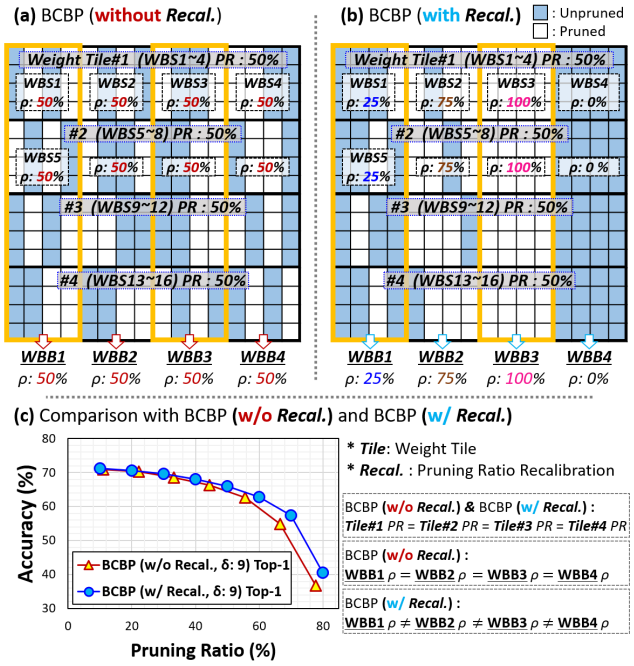


Figure 7: Comparison with BCBP (without pruning ratio recalibration) and BCBP (with pruning ratio recalibration) of VGG-16 on ImageNet. *Recal.* denotes pruning ratio recalibration. PR denotes pruning ratio.

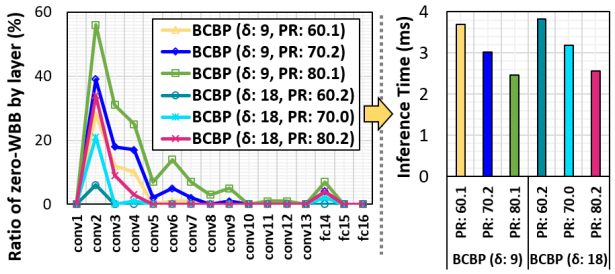


Figure 8: Ratio of WBB with 100% sparsity by δ variation of BCBP (VGG-16 on ImageNet within a 3% accuracy drop, τ: 32). The inference time of dense model is about $6.8ms$. PR denotes pruning ratio.

Discussion

Effect of Pruning Ratio Recalibration

Our proposed BCBP without pruning ratio recalibration (*Recal.*) has the same ρ among all WBSs, which maximizes GPU parallelism by balancing workloads among SMs. However, simply applying the same ρ in all WBSs can cause information loss due to removing important weights and is a coarse-grained method. As shown in Figure 7-(a), all WBSs have the same 50% ρ. In this case, to determine pruned weights, only weights in each WBS are considered. Therefore, more important weights, which are more sensitive to the accuracy, can be pruned even if the weights are more important than the unpruned weights, which are not sensitive to the accuracy, in the other WBS. To prevent this problem, we propose pruning ratio recalibration in the Pruning Ratio Recalibration Subsection. BCBP can balance workloads between SMs if one condition is satisfied: all the weight tiles have the same PR. The *Recal.* also maintains the same ρ in WBSs of WBB as shown in Figure 7-(b). With *Recal.*, the pruned model can be different ρ between WBBs. Figure 7-(c) shows that BCBP with *Recal.* has lower information loss than BCBP without *Recal.*.

Effect of WBB Width (δ)

As shown in Balanced Column-wise Block Pruning Section, the final pruning ratio, PR, is decided by two steps; the first is vector-wise tile pre-pruning, and the second is pruning ratio recalibration. Once PR is very high, the lowered weight matrix can be sufficiently sparse; therefore, we have an opportunity to find out fully zeroized WBB in advance. Moreover, we can also consider that when WBB width, δ, is small, resulting in small-sized WBB, discovering WBBs to be thoroughly pruned is possible enough. In this subsection, we denote WBB, which owns only WBSs to be pruned, as zero-WBB, and study how frequently making the zero-WBB, varying on PR and δ. We use two δ: 9 and 18. As shown in Figure 8, when δ is 9, BCBP produces zero-WBBs more increasingly as PR is more significant, especially in low-level convolutional layers such as conv2, conv3, and conv4. As δ varies from 9 to 18, such trend is still valid, but the ratio of zero-WBBs becomes low. In this regard, the elapsed time of VGG-16 becomes faster by increasing PR on both δ with an acceptable variation of accuracy drop. Furthermore, as δ changes from 18 to 9, BCBP shows 3.79%, 5.29%, and 4.07% elapsed time improvement, respectively, when PR is 60, 70, and 80; This gain is an extra benefit that comes from the index skipping operation in utilizing the acceleration API (Guo et al. 2020). In short, BCBP presents an opportunity to take time profit and similar accuracy if we exploit zero-WBBs even though making the same sized compressed model, i.e., from 138M parameters to 27.6M parameters (PR is 80) on VGG-16.

Conclusion

In this paper, we propose BCBP for fast DNN on GPUs. It brings a low pruning unit size in a balanced workload while leveraging lowering (im2col) and tiling. Instead of cross-wise tile pruning, BCBP adopts vector-wise tile pre-pruning to reach a high pruning ratio. It solves the workload imbalance among tiles by analyzing and balancing the workload between tiles. BCBP is compatible with existing NVIDIA GPU libraries that are highly hardware optimized. Through our experiments and analysis, BCBP outperforms in GPU inference time compared to prior works. Our proposed BCBP can be advantageous for computer vision tasks to leverage standard convolution operations such as object detection and semantic segmentation. We plan to apply our BCBP to the applications in the future.

Acknowledgements

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1A2B5B01002932).

References

Cai, Y.; Li, H.; Yuan, G.; Niu, W.; Li, Y.; Tang, X.; Ren, B.; and Wang, Y. 2021. Yolobile: Real-time object detection on mobile devices via compression-compilation co-design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 955–963.

Chellapilla, K.; Puri, S.; and Simard, P. 2006. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft.

Chen, T.; Ji, B.; Ding, T.; Fang, B.; Wang, G.; Zhu, Z.; Liang, L.; Shi, Y.; Yi, S.; and Tu, X. 2021. Only Train Once: A One-Shot Neural Network Training And Pruning Framework. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 19637–19651. Curran Associates, Inc.

Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; and Shelhamer, E. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, X.; Huang, J.; Yang, Y.; and Yan, S. 2017. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5840–5848.

Dong, X.; and Yang, Y. 2019. Network pruning via transformable architecture search. *Advances in Neural Information Processing Systems*, 32.

Elsen, E.; Dukhan, M.; Gale, T.; and Simonyan, K. 2020. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 14629–14638.

Gale, T.; Zaharia, M.; Young, C.; and Elsen, E. 2020. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–14. IEEE.

Gray, S.; Radford, A.; and Kingma, D. P. 2017. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3: 2.

Guo, C.; Hsueh, B. Y.; Leng, J.; Qiu, Y.; Guan, Y.; Wang, Z.; Jia, X.; Li, X.; Guo, M.; and Zhu, Y. 2020. Accelerating sparse DNN models without hardware-support via tile-wise sparsity. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3): 243–254.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 1135–1143.

Harris, P. 2014. The Mali GPU: An Abstract Machine, Part 2-Tile-based Rendering. https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/the-mali-gpu-an-abstract-machine-part-2---tile-based-rendering. Accessed: 2022-12-02.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, 2234–2240. AAAI Press. ISBN 9780999241127.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

Hill, P.; Jain, A.; Hill, M.; Zamirai, B.; Hsu, C.-H.; Laurenzano, M. A.; Mahlke, S.; Tang, L.; and Mars, J. 2017. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 786–799. ACM.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*.

Kim, W.; Kim, S.; Park, M.; and Jeon, G. 2020. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33: 585–595.

Kirk, D. B.; and Wen-Mei, W. H. 2016. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann.

Kloosterman, J.; Beaumont, J.; Wollman, M.; Sethia, A.; Dreslinski, R.; Mudge, T.; and Mahlke, S. 2015. WarpPool: Sharing requests with inter-warp coalescing for throughput processors. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 433–444. IEEE.

Kung, H.; McDanel, B.; and Zhang, S. Q. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 821–834.

Kung, J.; Park, J.; Park, S.; and Kim, J.-J. 2019. Peregrine: A flexible hardware accelerator for LSTM with limited synaptic connection patterns. In *Proceedings of the 56th Annual Design Automation Conference 2019*, 209. ACM.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations*.

Li, Z.; Yuan, G.; Niu, W.; Zhao, P.; Li, Y.; Cai, Y.; Shen, X.; Zhan, Z.; Kong, Z.; Jin, Q.; et al. 2021. NPAS: A Compiler-aware Framework of Unified Network Pruning and Architecture Search for Beyond Real-Time Mobile Acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14255–14266.

Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; and Shao, L. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1529–1538.

Lin, M.; Zhang, Y.; Li, Y.; Chen, B.; Chao, F.; Wang, M.; Li, S.; Tian, Y.; and Ji, R. 2022. 1xn pattern for pruning convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lin, S.; Ji, R.; Yan, C.; Zhang, B.; Cao, L.; Ye, Q.; Huang, F.; and Doermann, D. 2019. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2790–2799.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, 5058–5066.

Luong, M.; Brevdo, E.; and Zhao, R. 2017. Neural Machine Translation (seq2seq) Tutorial. https://github.com/tensorflow/nmt. Accessed: 2023-03-24.

Luong, M.-T.; and Manning, C. D. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Mishra, A.; Latorre, J. A.; Pool, J.; Stosic, D.; Stosic, D.; Venkatesh, G.; Yu, C.; and Micikevicius, P. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Narang, S.; Undersander, E.; and Diamos, G. 2017. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.

Naumov, M.; Chien, L.; Vandermersch, P.; and Kapasi, U. 2010. Cusparse library. In *GPU Technology Conference*.

NVIDIA. 2022. Nvidia Hopper architecture in-depth. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/. Accessed: 2023-03-24.

Nvidia, C. 2008. Cublas library. *NVIDIA Corporation, Santa Clara, California*, 15(27): 31.

Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S. W.; and Dally, W. J. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2): 27–40.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Rajpurkar, P.; Jia, R.; and Liang, P. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822*.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Sze, V.; Chen, Y.-H.; Yang, T.-J.; and Emer, J. S. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12): 2295–2329.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tanaka, H.; Kunin, D.; Yamins, D. L.; and Ganguli, S. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33: 6377–6389.

Vooturi, D. T.; Mudigere, D.; and Avancha, S. 2018. Hierarchical block sparse neural networks. *arXiv preprint arXiv:1808.03420*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, 2074–2082.

Yao, Z.; Cao, S.; Xiao, W.; Zhang, C.; and Nie, L. 2019. Balanced sparsity for efficient dnn inference on gpu. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 5676–5683.

Yu, J.; Lukefahr, A.; Palframan, D.; Dasika, G.; Das, R.; and Mahlke, S. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 548–560. IEEE.

Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.-Y.; and Davis, L. S. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9194–9203.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In Richard C. Wilson, E. R. H.; and Smith, W. A. P., eds., *Proceedings of the British Machine Vision Conference (BMVC)*, 87.1–87.12. BMVA Press. ISBN 1-901725-59-6.

Zhou, A.; Ma, Y.; Zhu, J.; Liu, J.; Zhang, Z.; Yuan, K.; Sun, W.; and Li, H. 2021. Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch. In *International Conference on Learning Representations*.