Balanced Column-Wise Block Pruning for Maximizing GPU Parallelism

Cheonjun Park¹, Mincheol Park¹,², Hyun Jae Oh³, Minkyu Kim¹, Myung Kuk Yoon¹, Suhyun Kim², and Won Woo Ro¹

¹Yonsei University
²Korea Institute of Science and Technology
³Samsung Electronics
⁴Ewha 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 multi-processors (SMs), which are independent processing units...
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 importance of 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, $\ell_1$-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 BCWP is performed on GPUs, following the order in Figure 3.

**Prerequisites** Given $n$ filters of $l^{th}$ convolution layer as $F^{(l)} \in \mathbb{R}^{n \times c \times k_h \times k_w}$, 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 $F^{(l)}$ can be denoted as $W^{(l)} \in \mathbb{R}^{n \times m}$, 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, $W^{(l)}$ is divided into multiple equal-sized tiles. It is denoted as $W^{(l)} = \{ \hat{W}^{(l)}_i \in \mathbb{R}^{\tau \times m}, i \in [1, T] \}$, where $\tau$ is tile width and $T$ is the number of tiles. If $n$ cannot be absolutely divided into $T$, this step of BCWP is disregarded and our pruning is converted as the group pruning.

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

$$s_{score}^{(l)} = C^{(l)}(\hat{W}^{(l)}_i).$$  

(1)

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

$$C^{(l)}(W^{(l)}) = \left\{ C^{(l)}(\hat{W}^{(l)}_i) = \| w^{(l)}_{i,k} \|_g \forall (i, k) \right\},$$

$$i \in [1, T], k \in [1, m].$$  

(2)
Pruning Ratio Recalibration

Pruning $W^{(l)}$ 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 $W^{(l)} = \{W_{i,j}^{(l)} \in \mathbb{R}^{n \times 8}, i \in [1,T], j \in [1,B]\}$, where $\tau$ is the tile width, $\delta$ is the WBB width, $T$ is the number of tiles and $B$ is the number of WBBs.

In a certain $j^{th}$ WBB, $W_{j}^{(l)}$ for a partial set of weights are denoted as $W_{j}^{(l)} = \{w_{i,j,k}^{(l)} \in \mathbb{R}, k \in [1, \delta T]\}$. 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:

$$\rho_{j}^{(l)} = \frac{\text{Count}(rid_{j}^{(l)})}{\text{Count}(W_{j}^{(l)})}, j \in [1,B].$$  (4)

In this paper, we use $\delta$ based on a spatial dimension of the convolution filter such as multiple of $k_hk_w$. 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 subblocks with tile width, defined as workload balanced subblocks (WBS). The lowered matrix with WBS can be denoted as $W^{(l)} = \{W_{i,j}^{(l)} \in \mathbb{R}^{\tau \times 8}, i \in [1,T], j \in [1,B]\}$, where $\tau$ is the tile width, $\delta$ is the WBB width, $T$ is the number of tiles and $B$ is the number of WBBs. A certain $(i,j)^{th}$ WBS, $W_{i,j}^{(l)}$, for a partial set of grouped weights are denoted as $W_{i,j}^{(l)} = \{w_{i,j,k}^{(l)} \in \mathbb{R}, k \in [1, \delta]\}$. Note that each grouped weight, $w_{i,j,k}^{(l)}$, is a column vector of a $W_{i,j}^{(l)}$.

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

Figure 4: Measurement of inference time for a whole tiled matrix of VGG-16 on ImageNet with a 1.5% accuracy drop. BCBP uses \( \tau \) of 32, which is the same as the VBP (Guo et al. 2020) and \( \delta \) 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.

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, \( 1 \times 10^{-4} \) 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

\[
\text{WIT} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} |\mathcal{PR}(x) - \mathcal{PR}(x^*)|,
\]

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 \((\tau)\) We verify BCBP on two tile widths \((\tau)\) 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 \( \tau \) (Guo et al. 2020) and Mobile GPU, such
as ARM Mali, sets $\tau$ as 16 (Harris 2014). Figure 5-(a) and (b) give a comparison in inference time and WIT with two $\tau$ settings: 16 and 32. Changing $\tau$ 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 $\tau$. As shown in Figure 5-(a) and (b), when $\tau$ 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 $\tau$, $\delta$, 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, $4.83 \pm 1.79$, $6.61 \pm 2.53$, $10.05 \pm 2.32$ and $17.93 \pm 4.26$, 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.

**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 $\mathbb{R}^{32 \times 32}$. 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 times 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 times 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 $\mathbb{R}^{32 \times 32}$. The pruning unit size of Tile Sparsity (TS) (Guo et al. 2020), which is both row and column pruning, is a $\mathbb{R}^{32 \times 1}$ and $\mathbb{R}^{1 \times 32}$. However, pruning unit size of BCBP, which is column pruning, is only $\mathbb{R}^{32 \times 1}$. Due to the $32 \times$ 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.

<table>
<thead>
<tr>
<th>Model (Type)</th>
<th>Pruning unit size</th>
<th>$PR$ (%)</th>
<th>Top-1 $\downarrow$(%)</th>
<th>time (ms)</th>
<th>speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STC (BLP)</td>
<td>$\mathbb{R}$</td>
<td>50.0</td>
<td>-0.88</td>
<td>3.5</td>
<td>1.94 x</td>
</tr>
<tr>
<td>BLS (BLP)</td>
<td>$\mathbb{R}$</td>
<td>91.9</td>
<td>-1.31</td>
<td>5.2</td>
<td>1.31 x</td>
</tr>
<tr>
<td>BS (BP)</td>
<td>$\mathbb{R}^{32 \times 32}$</td>
<td>71.6</td>
<td>-2.54</td>
<td>5.9</td>
<td>1.16 x</td>
</tr>
<tr>
<td>TS (VBP)</td>
<td>$\mathbb{R}^{32 \times 1 \times m}$</td>
<td>80.0</td>
<td>-2.32</td>
<td>3.0</td>
<td>2.28 x</td>
</tr>
<tr>
<td>BCBP</td>
<td>$\mathbb{R}^{32 \times 1}$</td>
<td>49.8</td>
<td>-0.97</td>
<td>4.5</td>
<td>1.52 x</td>
</tr>
<tr>
<td>BCBP</td>
<td>$\mathbb{R}^{32 \times 1}$</td>
<td>60.1</td>
<td>-1.96</td>
<td>2.5</td>
<td>2.77 x</td>
</tr>
</tbody>
</table>

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-$1\downarrow$ means the top-1 accuracy drop rate compared to dense model. The time denotes inference time.
Figure 6: The pruning mask visualization for the third convolutional layer of VGG-16 on ImageNet. The mask is represented by the output of $\mathbb{I}_{W(l)}(x)$ where $x$ is a weight value of $W^{(l)}$. The pruned region is blue. The dotted red line denotes a tile. Rigid orange line denotes a WBB.

### 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:

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

where $W^{(l)}$ is a 2D weight matrix, which is a converted 4D filter weight of the convolutional layer by the lowering, $K^{(l)}$ denotes the unpruned parameter set of $W^{(l)}$, 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 $W^{(l)}$. We use the third convolutional layer of VGG-16. Thus, $W^{(l)}$ 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).
Discussion

Effect of Pruning Ratio Recalibration

Our proposed BCBP without pruning ratio recalibration (Recal.) has the same \( \rho \) among all WBSs, which maximizes GPU parallelism by balancing workloads among SMs. However, simply applying the same \( \rho \) 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\% \( \rho \). 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 \( \rho \) in WBSs of WBB as shown in Figure 7-(b). With Recal., the pruned model can be different \( \rho \) between WBSs. Figure 7-(c) shows that BCBP with Recal. has lower information loss than BCBP without Recal.

Effect of WBB Width (\( \delta \))

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, \( \delta \), 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 \( \delta \). We use two \( \delta \): 9 and 18. As shown in Figure 8, when \( \delta \) 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 \( \delta \) 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 \( \delta \) with an acceptable variation of accuracy drop. Furthermore, as \( \delta \) 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 (ITTP) 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


