Retrospective: Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism

Jiecao Yu
Meta Platforms, Inc.
jiecaoyu@meta.com

Reetuparna Das
University of Michigan
reetudas@umich.edu

Scott Mahlke
University of Michigan
mahlke@umich.edu

I. Background

Back in 2017, Deep Neural Networks (DNN) models were fundamentally changing the development of artificial intelligence applications. Across various application domains, including computer vision and natural language processing, advanced DNN models were proposed to drive higher prediction accuracy and increased functionality.

However, these new models were wider and deeper, resulting in higher computation and memory utilization. This trend increased the hardware demand and resulted in larger training and inference costs. Considering the high redundancy inside these larger DNN models, many researchers proposed pruning to remove unnecessary parameters [1], thereby reducing both the corresponding computation and memory footprints.

One critical but common issue of most pruning techniques was neglecting the gap between FLOP reduction and performance improvement on real hardware. Conventional pruning algorithms inserted irregular sparsity into DNN models, thereby converting dense linear algebra computation into the sparse computation. Sparse computation is inherently less efficient because it is very difficult to exploit data-parallel hardware to accelerate the computation. At the same time, due to such irregular sparsity, we need to use special sparse formats (e.g., compressed sparse row format) to store the weight tensors. The DNN computation then needs extra computation (and memory accesses) to decode those sparse formats.

Due to these issues, although pruning was dramatically reducing the FLOPs, in many cases by 90% or more, the incurred sparsity can actually hurt the overall performance on both CPUs and GPUs. Further, encoding the sparse format of pruned DNN models incurs additional storage space, which makes it difficult to use pruning to reduce the memory cost. Figure 1 shows the relative execution time, model sizes, and MAC operations of DNN models pruned by Deep Compression [1] with respect to the original DNN models. The first three bars show the relative execution time on the microcontroller, CPU, and GPU. As shown in the figure, the relative execution time is much higher than the relative model sizes and MAC operations. Weight pruning hurts the performance of LeNet-5 (on GPU), ConvNet, NIN, and AlexNet, which causes an execution time increase for these networks.

II. Developing Ideas

To address the performance issues of DNN pruning, our paper proposed to customize DNN pruning based on the underlying data-parallel hardware structure of the target hardware.

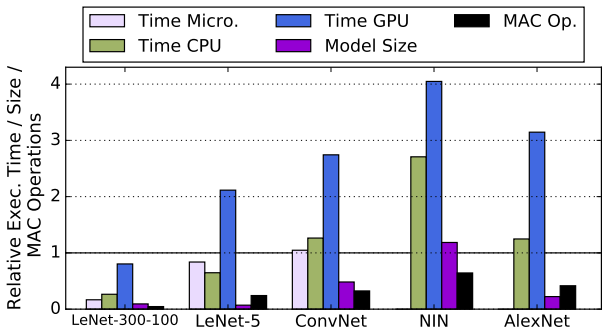

Fig. 1. Relative execution time (Time Microcontroller/ CPU/ GPU), model sizes, and MAC operations of the networks pruned with the traditional pruning technique in Deep Compression. NIN and AlexNet are not tested on the microcontroller due to its limited storage size.

We first started our exploration targeting Intel CPUs. One important hardware support that Intel CPUs provide for utilizing computation parallelism is the SIMD extensions (e.g., AVX/SSE). Fully utilizing the SIMD support usually requires the input and output data to be put in continuous memory space so the memory access and computation can be vectorized. With conventional pruning algorithms, although we can store the remaining weights after pruning into continuous memory addresses, the corresponding input values that need to be fetched for matrix multiplication and convolution operations were not stored consecutively. The result is the underutilization of the SIMD extensions. In this case, if we can redesign the pruning algorithms to make sure the required input values are also stored in continuous memory space, we can make much better utilization of the SIMD support.

To achieve this goal, we developed a SIMD-aware pruning algorithm. It puts weights in groups and performs pruning at the granularity of weight groups. The weights in the same group are either all removed or kept. The size of the weight group is enforced to be the same as the SIMD width. In this case, each weight group and also the corresponding input values can be fetched in parallel using the SIMD instructions. This can help dramatically improve the computation performance under the same model sparsity. At the same time, the storage cost is also significantly reduced since we only need to encode the position of weight groups instead of the position of every weight.

As part of the initial exploration, the first co-author, Jiecao Yu, went for a summer internship at ARM to explore SIMD-aware pruning in more depth. Through the internship, prun-

ing algorithms were explored for a variety of processors including low-power microcontrollers. ARM microcontrollers also provide SIMD extensions. Therefore, we evaluated the SIMD-aware pruning algorithm, and observed noticeable performance improvement and storage size reduction for both fully-connected (FC) layers and convolutional (CONV) layers.

For Intel processors, the results were different. SIMD-aware pruning provided much better computation efficiency than the conventional pruning methods for FC layers, but it was often not helpful for CONV layers. There are many more data reuse opportunities in CONV layers, thus the sparsity required for achieving performance improvement with both conventional and SIMD-aware pruning algorithms is much higher than we could create at a reasonable accuracy budget. Thus, we realized that we need to further increase the granularity of pruning to mitigate the negative impact of pruning and proposed node pruning which removes redundant nodes (e.g., FC neurons and CONV channels) instead of individual weights. With node pruning, DNN models after pruning can still maintain dense regular computation, and the computation reduction can be effectively transferred into performance improvement even with lesser amounts of pruning.

After the exploration of desktop processors and microcontrollers, we further extended our work for GPUs. Because of the high hardware parallelism supported by GPUs, node pruning provided better performance efficiency for both FC and CONV layers. Across the microcontroller, CPU, and GPU, Scalpel achieves mean speedups of 3.54x, 2.61x, and 1.25x while reducing the model sizes by 88%, 82%, and 53%.

Through the exploration process, we concluded that the pruning granularity and structure need to be adjusted based on the layer types and target hardware parallelism. Combining the proposed algorithms and findings, our paper made the following contributions:

- We demonstrated that the impact of DNN pruning is closely coupled to both the layer types and the data-parallel structure of the target hardware.
- We proposed to customize the pruning algorithms to the target hardware platform.
- We introduced two new pruning algorithms which can effectively transfer the computation reduction from pruning into computation efficiency improvement and storage space reduction.

III. Looking Back

Through the years after our paper got published, the computation cost of DNNs continued to scale upward with larger models that provide higher accuracy, more features, and personalization. Recently, the trends of transformers and large language models like ChatGPT are further pushing the model size to an extreme. How to serve those extremely large models is becoming a critical issue blocking the deployment of those models in the industry. As a conventional method for compressing DNN models, pruning remains one of the most important research topics for enabling the serving of expensive DNN models while maintaining manageable server costs.

Our work is one of the pioneering works proposing the concept of structured pruning and node pruning. The following works introduced more granularity levels, e.g., shape-wise and depth-wise, that pruning can be performed on. Besides, researchers proposed various new criteria to choose the weight groups that need to be pruned, achieving higher sparsity levels while maintaining model accuracy. For example, network slimming [3] eliminated redundant CONV layer channels using scaling factors in batch normalization layers, and StructADMM [7] proposed to use alternating direction method of multipliers (ADMM) to identify the important neurons.

At the same time, different hardware designs were proposed to provide better support for structured sparsity. As an example, Bit Prudent [5] explored utilizing structured sparsity for in-cache computation, bringing 1.6× speedup compared against other in-cache accelerators. Besides, Nvidia Tensor Cores provided support for accelerating 2:4 fine-grained structured sparsity [4].

As DNN models involving, structured pruning was studied on various model and layer types besides conventional computer vision models and CONV layers. For example, Wen et al. [6] and Lagunas et al. [2] explored applying structured pruning on LSTM and transformer models, respectively.

Structured pruning can also be combined with other techniques to further accelerate DNN execution. Similar to pruning, low-precision quantization, knowledge distillation, neural architecture search (NAS), and other techniques were explored to eliminate the redundancy inside DNN models. These techniques including pruning can be applied and evaluated together to maximize the computation efficiency while maintaining the model accuracy.

References

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[2] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, "Block pruning for faster transformers," *arXiv preprint arXiv:2109.04838*, 2021.

[3] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.

[4] J. Pool, A. Sawarkar, and J. Rodge, "Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt," 2021. [Online]. Available: https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

[5] X. Wang, J. Yu, C. Augustine, and R. D. Ravi Iyer, "Bit prudent in-cache acceleration of deep convolutional neural networks," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 81–93.

[6] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," *arXiv preprint arXiv:1709.05027*, 2017.

[7] T. Zhang, S. Ye, X. Feng, X. Ma, K. Zhang, Z. Li, J. Tang, S. Liu, X. Lin, Y. Liu, M. Fardad, and Y. Wang, "Structadmm: Achieving ultrahigh efficiency in structured pruning for dnns," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 5, pp. 2259–2273, 2021.