Review and Benchmark of sparse CNNs on GPU

Author: Qilin Chen   Supervisor: Hasan Mohamed   Responsible Professors: Shih-Chii Liu, Nergis Tomen

TU Delft
Institute of Neuroinformatics
University of Zurich
ETH zürich

1. Introduction

- Pruning is often applied to Convolution neural networks (CNNs) to reduce the computation and memory cost for training and inference.
- As a result of pruning, CNNs in real-life applications can be highly sparse (with at least 70% sparsity) without much accuracy loss.
- GPUs cannot natively support sparse matrix calculations, sparse matrices are treated as dense matrices during computation.
- To take advantage of the sparsity in CNNs and accelerate inference, many methods are proposed, but there lacks a systematic summary and comparison for them.

2. Background

- The structure of modern GPUs is as follows: thread (with private register and access to the constant cache) --> thread groups (a warp of 32 threads) --> thread blocks.
- The most important operations for CNN inference: convolution and pooling are performed in thread blocks, convolution can be viewed as sparse matrix-matrix multiplication (SpMM).

3. Research Question

What is the recent literature on exploiting sparse CNNs on GPUs?

- Write a short summary that lists all the libraries
- Benchmark them on the Jetson Nano

4. Methods

- **Summarize the sparse CNN accelerators: TensorRT [1], SparseRT [2], Sputnik [3], Conv_Pool_Algorithm [4], and MinkowskiEngine [5]**

(1) TensorRT: 2:4 fine-grained sparsity for weight sparsity (50% sparsity), sparse matrix is stored in a compressed format

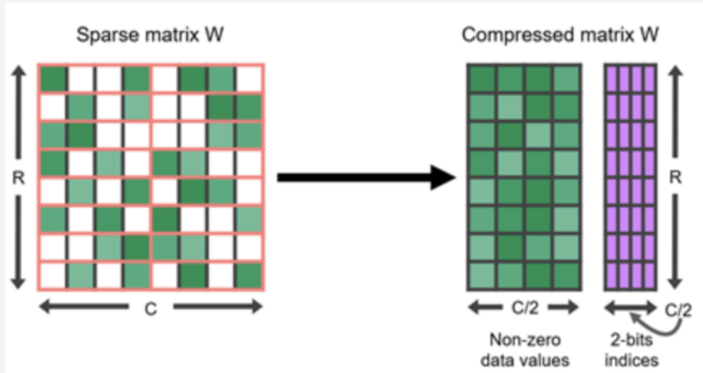

Figure 1: Structured sparse matrix W, and its compressed representation. [1]

(2) Sparse RT: unstructured weight sparsity, tiling and load balancing for the sparse matrix are computed at compile time and used as a part of the code

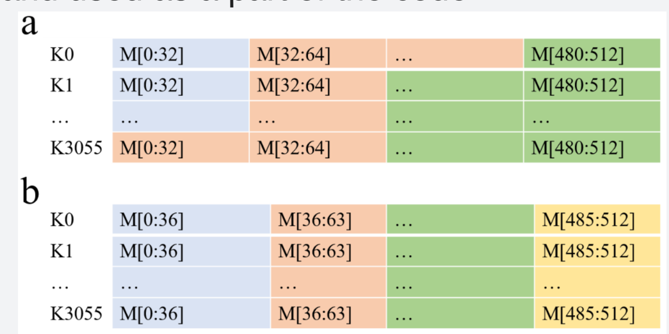

Figure 2: Thread block level load balancing [2]

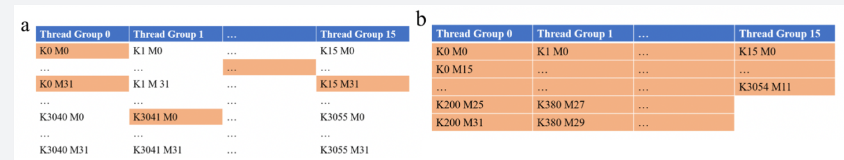

Figure 3: Thread group level load balancing [2]

(3) Sputnik: unstructured weight sparsity, tiling and load balancing to solve share memory load bottleneck

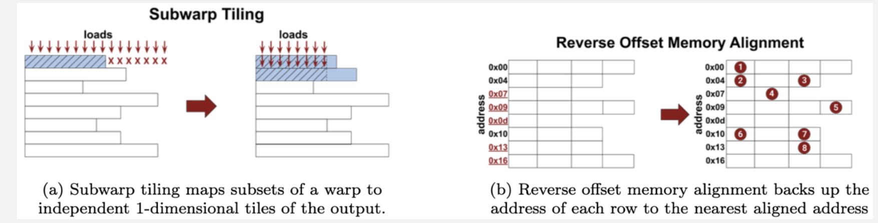

Figure 4: Subwarp tiling and ROMA [3]

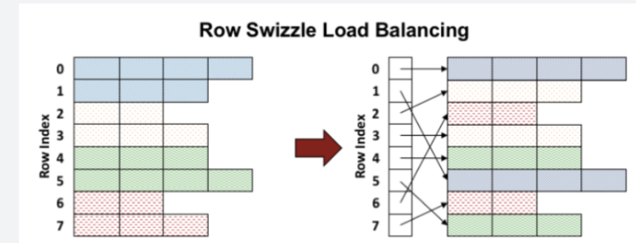

Figure 5: Row swizzle [3]

(4) Conv_Pool_Algorithm: feature map sparsity, stores only non-zeros with their kernel values and indices and combines convolution and pooling

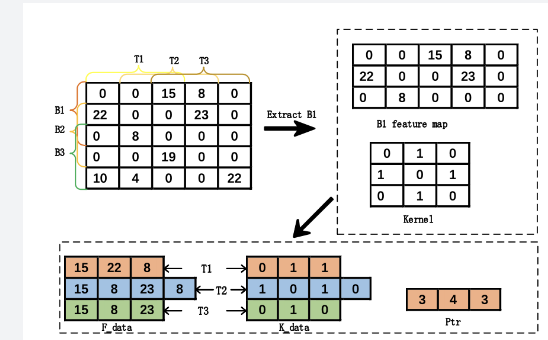

Figure 6: ECR storage format [4]

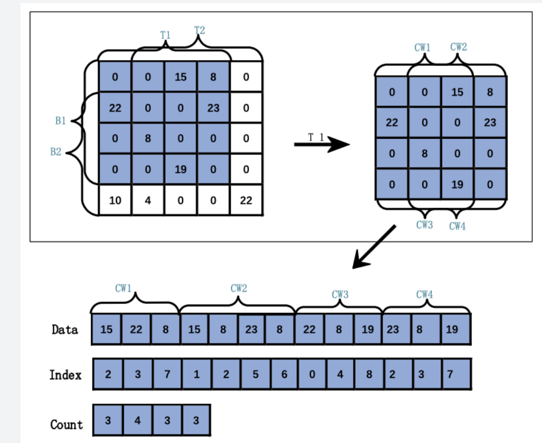

Figure 7: PECR storage format [4]

(5) MinkowskiEngine: feature map sparsity, stores the coordinates of non-zeros and pairs them with the output coordinates to generate kernel maps

- **Benchmark the reviewed methods to show the speedup they achieve: RTX 3080 with Ubuntu 18.04, CUDA 11.4, cuDNN 8.2, and Python 3.6.9**

TensorRT (v8.0 and above)	SparseRT and Sputnik	Conv_Pool_Algorithm
Benchmarked using TrafficCamNet (based on ResNet-18) running inference for speedup and accuracy	Benchmarked on SpMM using different matrix dimensions and sparsity levels for runtime and speedup	Benchmarked using VGG-19 and ImageNet for convolution and pooling of an entire layer for runtime and speedup

5. Results

- **TensorRT: Inference using TrafficCamNet and ImageNet data**

Condition	Unpruned w/o TensorRT	Unpruned w/ TensorRT	Pruned w/ TensorRT
Model size	44.32MB	44.32MB	5.2MB
Speedup	1x	1.15x	1.21x
Accuracy	84%	84%	84%

Table 1: Inference result for TrafficCamNet under different settings

- **Conv_Pool_Algorithm: convolution and pooling for different layers of VGG-19 speedup compared to cuDNN**
  Average speedup: ECR: 16.63x, PECR: 17.61x

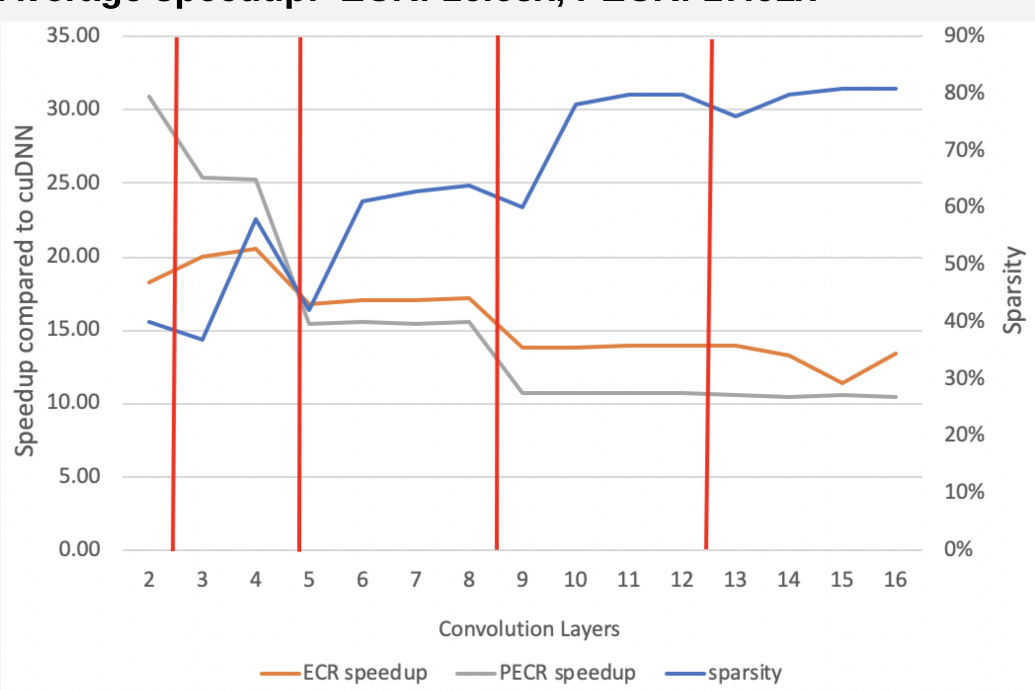

Figure 8 sparsity and speedup for conv_pool_algorithm, red line separates the feature maps with the same sizes

- **SparseRT and Sputnik: time and speedup to perform SpMM compared to cuBLAS**
  Average speedup: SparseRT: 1.86x, Sputnik: 1.78x

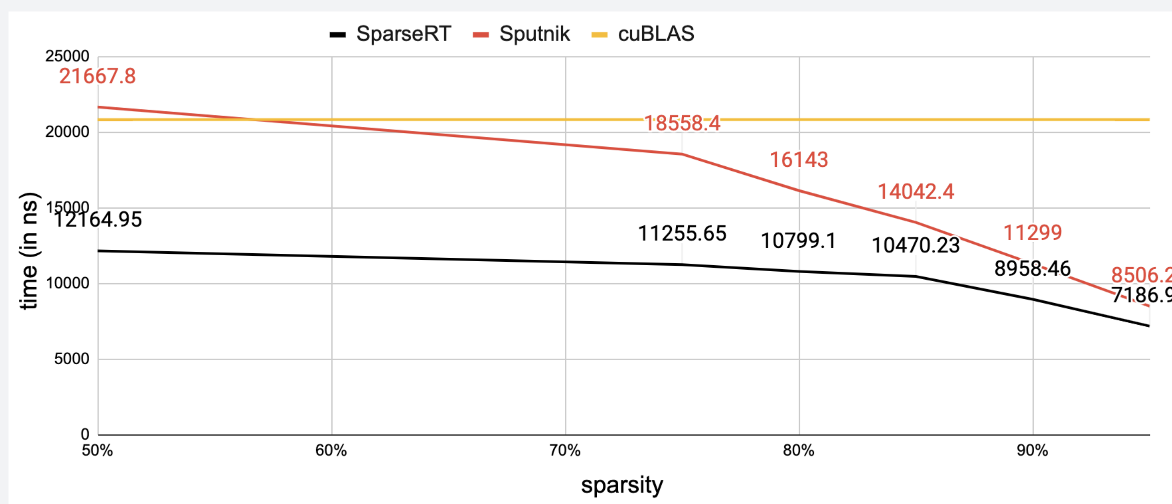

Figure 9: Runtime for SpMM with different sparsity

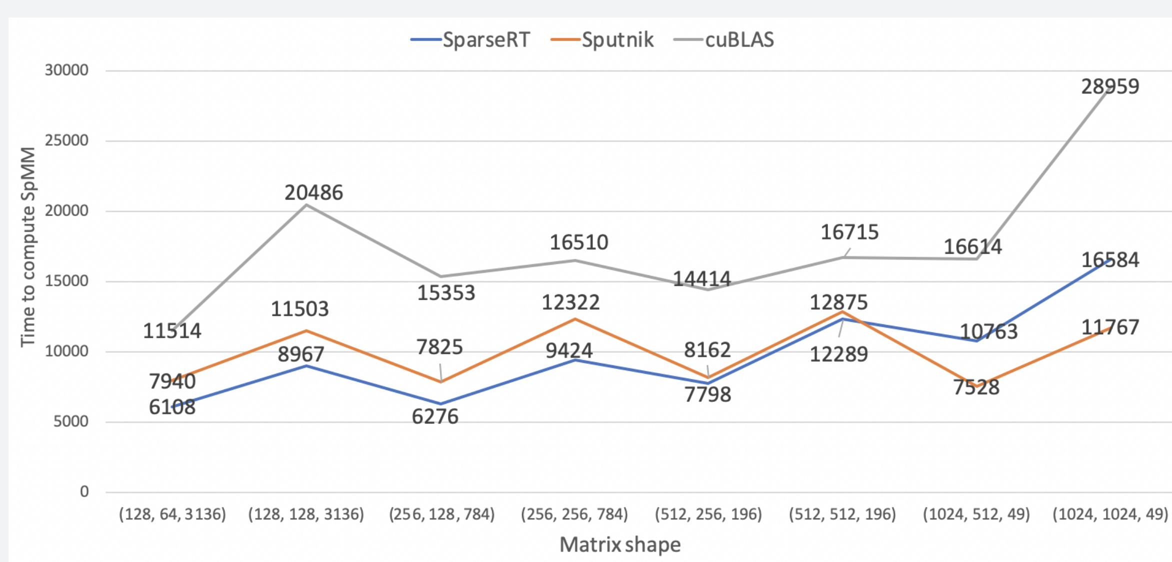

Figure 10: Runtime for SpMM with different dimensions (M, K, N), (M, K) is the dimension of the filter, (K, N) is the dimension of the input

6. Conclusions and Disscussion

- TensorRT, Conv_Pool_Algorhithm and MinkowskiEngine all proposed a way to compress the matrices and only stores non-zero values. TensorRT can accelerate CNN inference with 50% sparsity without accuracy loss, Conv_Pool_Algorithm can achieve high speedup for convolution and pooling.
- SparseRT and Sputnik both aims to accelerate weight sparsity by using tiling and load balancing and the speedup is higher when the sparsity is higher. SparseRT performs better for smaller-size filters, and Spunik is better for larger ones.
- Sputnik only achieves a high speedup when the matrix is highly sparse. More experiments are needed to analyze which step of the SpMM is slowing down the performance.
- For Conv_Pool_Algorithm, sparsity shows no effect on the speedup, more experiments using matrices of the same dimension but different sparsity could provide more information on this observation.
- Due to the low compatibility of the libraries, using the libraries on end-to-end models inference is infeasible, a program to convert the models and their input to the format each library supports needs to be developed.

7. References

[1] "Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT". [Online]. Available: https://developer.NVIDIA.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/
[2] Z. Wang, "SparseRT: Accelerating Unstructured Sparsity on GPUs for Deep Learning Inference", arXiv:2008.11849, 2020.
[3] T. Gale, M. Zaharia, C. Young, E. Elsen, "Sparse GPU Kernels for Deep Learning", arXiv preprint arXiv: 1902.10901, 2020.
[4] W. Xu, S. Fan, H. Yu, X. Fu, "Accelerating convolutional neural networks by exploiting sparsity on GPUs", arXiv preprint arXiv:1909.09927, 2019.
[5] C. Choy, J. Gwak and S. Savarese, "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 3070-3079, doi: 10.1109/CVPR.2019.00319.