15618 Project Proposal
CUDA Optimization for SpMM

Zepeng Zhao (AndrewID zepengz)
Carnegie Mellon University
Pittsburgh, PA 15213
zepengz@andrew.cmu.edu

Mingxin Li (AndrewID mingxinl)
Carnegie Mellon University
Pittsburgh, PA 15213
mingxinl@andrew.cmu.edu

Abstract

The ever-increasing size of deep learning models, especially LLMs, demand higher and higher computational efficiency. Many techniques exploiting the sparsity of model weights are invented to accelerate the inference and training process, an example of which is pruning. After removing the less significant weights, sparse matrix-matrix multiplication can be applied to replace dense MM and achieve more speedup. This project focuses on implementing and optimizing Sparse Matrix-Matrix Multiplication, a.k.a. SPMM, in the context of LLM inference. We plan to analyze the characteristics of different storage formats, explore different algorithms, and implement a high-performance SPMM CUDA kernel optimized with respect to GPU hardware features. We aim to reduce memory communication overhead and increase computation speed for large SPMM in real-world applications. We will also compare the performance of our optimization with the NVIDIA cuSparse[7] library and the dense counterpart cuBLAS[6] to understand and evaluate the effectiveness of our implementation.

Our project will be released on https://github.com/mli43/Cuda-Optimization-for-SpMM

1 Background

Large Language Models (LLMs) have become a cornerstone of modern natural language processing, enabling advances in tasks like machine translation, summarization, and conversational AI. However, LLMs demand significant computational resources during inference, where the need to generate responses or predictions in real-time becomes crucial. Efficient inference is therefore a critical area of study, where methods to reduce computational overhead and memory usage are continuously sought after.

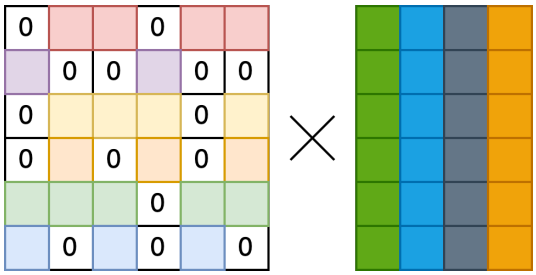

Figure 1: SpMM (The left matrix is sparse and the right one is dense)

Preprint. Under review.

Pruning is a technique to accelerate inference by exploiting the sparsity of weight matrices of LLMs. By removing the trivial weights and compressing the weight matrices, pruning can reduce the memory footprint and unnecessary computation, thus improving throughput. Sparse matrix multiplication (SpMM) is the core computation of this process. SpMM describes a matrix-matrix multiplication where at least one matrix is sparse. In the context of LLMs, the activations are usually dense while the weights can be pruned to be sparse. Therefore, we are to deal with a SpMM where the left matrix is sparse and the right one is dense (Figure 1). Besides LLMs, this computational paradigm is also widely used nowadays in many other applications, such as graph algorithms[5] and algebraic solvers[2].

GPUs are specialized processors designed to handle massive parallel computations, originally for rendering graphics but now widely used for general-purpose computing tasks, especially in AI and scientific computing. CUDA is NVIDIA's parallel computing platform and programming model that allows developers to harness the computational power of GPUs by writing code in C, C++, and Python. CUDA is particularly well-suited for accelerating SpMM due to several reasons: First, CUDA is designed to execute thousands of threads simultaneously. SpMM, especially with large matrices like model weights, can exploit this parallelism by distributing computations across threads to handle non-zero elements efficiently. Second, GPUs have very high bandwidth to load big matrices much faster than CPUs. Third, current Deep Learning environments are deeply coupled with CUDA and it's very easy to implement and tune a high-performance kernel using CUDA toolkit, making CUDA a great environment to be based on.

2 The Challenge

2.1 Irregular Data Distribution

Sparse matrices often contain non-uniform distribution of non-zero elements. Unlike dense matrices where each row or column has a consistent structure, there may be rows that are densely populated or entirely empty in a sparse matrix. The sparsity pattern and poor locality characteristic lead to highly irregular memory access patterns. Because of this reason, it is difficult to achieve balanced thread workload on GPU or minimize thread divergence. Additionally, the sparsity pattern also leads to more frequent fetching of non-zero elements, followed by few computation operations such as multiplication and addition. This means that the application has a high communication-to-computation ratio, where the communication cost dominates the execution and becomes the bottleneck for performance increase.

2.2 Performance Goal

The target we will compare to is cuSparse, which is a highly optimized CUDA library for sparse matrix multiplication. It is very challenging to implement a custom CUDA Kernel outperforming these fine-tuned libraries by experts from NVIDIA. They are highly optimized for hardware and have been continuously updated for years. Even though we don't aim to surpass them, comparing with them is already very ambitious.

2.3 CUDA Implementation

While it's easy to implement a rudimentary SpMM on CPU, there is much more complicated works to design a GPU version with good throughput where thousands of threads are to be synthesized. We should figure out the best grid size for a specific matrix, how to evenly distribute among the threads, how to accommodate different matrix sizes, how to fully exploit GPU hardware features like shared memory, and so on. Debugging and fine-tuning a CUDA kernel is not an easy task even for experienced programmers.

2.4 Computational Features

While the computational features of GEMM are regular and predictable and there are many high-performance kernel implementations of it, SpMM has much more complicated computational features, thus being more difficult to optimize. There are many factors that influence each other: storage format, data loading method, computation order and hardware constraints. We need to do a lot of

experiments and analysis to gain a deeper understanding of the features of this computation paradigm to support our development, and explore a myriad of optimization techniques to achieve the best performance.

3 Resources

1. To test our implementation, we need access to machines with NVIDIA GPUs, like GHC and PSC machines.
2. Code base: We will start from scratch for our implementation and compare performance with NVIDIA's cuSparse library and its dense counterpart cuBLAS.
3. Data: We plan to use real world LLM weight matrices as our test data to best reflect the effectiveness of our implementation.
4. Books/papers: We are still researching different optimization techniques. Here are some resources that we think can help us develop our parallel algorithms[8][4][3].

4 Goals and Deliverables

1. 50%: Basic Implementation
   (a) Implement CPU versions as baseline and verifier serving our GPU version
   (b) Randomly generate sparse matrix for correctness test
   (c) Analyze different storage formats to choose the best one
   (d) Implement basic CUDA kernels without any performance requirements
2. 75%: Advanced Implementation
   (a) Use advanced CUDA techniques like shared memory to reduce memory traffic, mitigate warp divergence, avoid bank conflicts and improve throughput
   (b) Perform thorough profiling on our implementation using Nsight Compute and continue fine-tune the kernel based on the analysis
   (c) Achieve at least 20% throughput of cuSparse.
3. 100%: Acceleration, benchmarking, and analysis
   (a) Continue fine-tuning the kernel based on critical profiling and analysis
   (b) Explore different implementations for different settings
   (c) Collect real-word LLM weight matrices for final evaluation
   (d) Conduct comprehensive experiments and evaluate our implementation from different perspectives
   (e) Achieve near 50% throughput of cuSparse.
4. 125%: Stretch Goals
   (a) Starting with the NVIDIA Ampere architecture and the introduction of the A100 Tensor Core GPU, NVIDIA GPUs have the fine-grained structure sparsity feature, which can be used to further accelerate SPMM[1].
   (b) Wrap our implementation as a plug-and-play library that can be easily used by the community.

5 Platform Choice

Because our project is based on CUDA, we mainly focus on the GPU type of the platform. At the beginning of the project, we will use GHC (RTX 2080 Turing architecture) and PSC (V-100 Volta architecture) machines for development. Once we get access to GPUs that are no older than Ampere architecture, we will shift our development onto it, because some features of cuSparse are only supported by the Ampere architecture and beyond.

6 Schedule

Date	Milestone
11/13	Finish proposal.
11/17	Complete environment setup and request resource access.
11/20	Complete the 50% goal.
11/24	Get familiar with cuSparse and cuBLAS and write benchmark scripts.
11/27	Finish milestone writeup; Complete the 75% goal.
12/1	Achieve the 100% perf goal.
12/4	Complete other parts of the 100% goal and prepare the poster and the final report; Try Stretch Goals.
12/8	Finish poster.
12/13	Review and finalize report.

References

[1] Roberto L. Castro, Andrei Ivanov, Diego Andrade, Tal Ben-Nun, Basilio B. Fraguela, and Torsten Hoefler. Venom: A vectorized n:m format for unleashing the power of sparse tensor cores, 2023.

[2] Jianhua Gao, Weixing Ji, Fangli Chang, Shiyu Han, Bingxin Wei, Zeming Liu, and Yizhuo Wang. A systematic survey of general sparse matrix-matrix multiplication. *ACM Computing Surveys*, 55(12):1–36, March 2023.

[3] Lukas Gianinazzi, Alexandros Nikolaos Ziogas, Langwen Huang, Piotr Luczynski, Saleh Ashkboosh, Florian Scheidl, Armon Carigiet, Chio Ge, Nabil Abubaker, Maciej Besta, Tal Ben-Nun, and Torsten Hoefler. Arrow matrix decomposition: A novel approach for communication-efficient sparse matrix multiplication. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '24, page 404–416, New York, NY, USA, 2024. Association for Computing Machinery.

[4] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, page 300–314, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Guyue Huang, Guohao Dai, Yu Wang, and Huazhong Yang. Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks, 2020.

[6] NVIDIA. cublas, 2024.

[7] NVIDIA. cusparse, 2024.

[8] Walid Abu Sufah and Khalid Ahmad. On implementing sparse matrix multi-vector multiplication on gpus. In *Proceedings of the 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, HPCC '14, page 1117–1124, USA, 2014. IEEE Computer Society.