EasyChair Preprint

№ 13973

A Source-to-Source Format and Schedule Auto-Tuning Framework for Sparse Tensor Program

Xiangjun Qu, Lei Gong, Wenqi Lou, Chao Wang and Xuehai Zhou

July 15, 2024

A Source-to-Source Format and Schedule Auto-Tuning Framework for Sparse Tensor Program

Xiangjun Qu, Lei Gong, Wenqi Lou, Chao Wang, and Xuehai Zhou

University of Science and Technology of China, Hefei, China
quxiangjun@mail.ustc.edu.cn, {leigong0203, louwenqi, cswang, xhzhou}@ustc.edu.cn

Abstract. Sparse tensor compilers simplify the development and optimization of operators through high-level abstract representation and auto-tuning. However, existing work also relies on compilation and hardware knowledge to specify the design space for tuning, and also their search strategy is limited, which creates unavoidable cost and efficiency issues. In this paper, we propose a source-to-source auto-tuning framework that targets sparse format and schedule for sparse tensor program. The framework extracts sparse tensor computational patterns based on computational graphs to automatically generate design space, and designs a adaptive exploration scheme based on reinforcement learning and heuristic algorithm to find the optimal format and schedule configurations in it. Preliminary experiments show that we achieve significant performance gains compared to state-of-the-art high-performance arithmetic libraries, manual optimization schemes, and auto-tuning frameworks.

Keywords: Sparse Computation · Sparse Tensor Compiler · Code Generation and Optimizations · Auto-Tuning.

1 Introduce

Tensor algebraic computations are crucial across various domains, where tensor data in practical applications often exhibit large-scale and sparse characteristics. Over the past few decades, academia have proposed numerous sparse tensor storage formats and schedule strategies. They aim to improve the efficiency of sparse tensor program execution through operator optimization.

Currently, sparse tensor computation system commonly rely on manually high-performance libraries provided by hardware vendors, such as MKL. These low-level kernel operators implement common storage formats and algorithms. Recently, there has also been work related to handwritten operators optimizing sparse tensor computations through tiling reordering algorithms [2]. However, such solutions require developers to perform a lot of manual optimizations for storage formats, schedule and compilation levels on different hardware platforms, which suffers from slow development cycles and poor portability. The deep learning compiler TVM [1] solves this problem by performing automatic operator generation and tuning through high-level abstract representations, but it has very

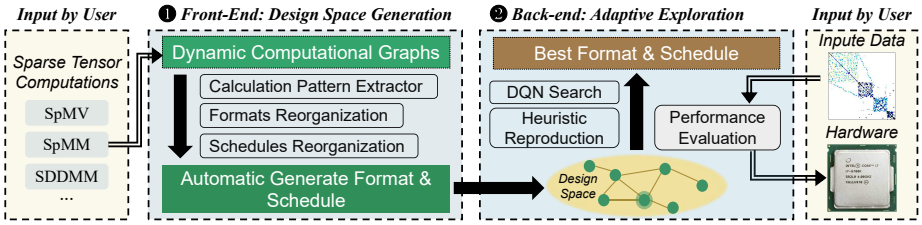

Fig. 1. Workflow of auto-tuning framework.

limited support for sparse tensor format representations. Recent work WACO [7] introduces a collaborative auto-tuning scheme for sparse format and loop schedule based on the sparse tensor compiler TACO [3]. However, programmer still needs to design parameter templates for each new operator that contain format and schedule to specify the design space, and its effectiveness depends heavily on the programmer's professional experience. Meanwhile, WACO constructs KNN graphs from datasets and executes ANNS algorithms for simple navigation on them, whose effectiveness is limited to the size of the KNN graph and the search capability.

Our Approach. To address the above problems, this paper presents a source-to-source auto-tuning framework for sparse tensor program. Users only need to describe sparse tensor data and algorithms in the DSL provided in the front-end of the framework, without relying on any knowledge of compilation and hardware. The framework automatically generates a design space for format and schedule for the target workload and hardware platform, and searches for the optimal configuration.

2 Approach

As shown in Fig. 1, the user describes the sparse tensor algorithm (e.g., SpMM) while inputting the sparse tensor data of the workload and specifying the target hardware platform. After receiving the user input, the framework determines the optimal storage format and schedule configration through a two-phase process of front-end and back-end co-tuning, which is a fully automated tuning process.

Front-end. We provide a DSL at the top-level programming interface to accept user descriptions of sparse tensor algorithms, and automatically organize the algorithms into computational graphs. The format representation of sparse storage uses TACO's coordinate hierarchy, where each level is expressed as a format attribute. We traverse the nodes and edges of the graph to collect information representing the sparse tensor computational patterns, including the axis, tensor, and computational graph structure, etc. Based on this information, front-end automatically adds the format and schedule primitives from Table 1 to generate design space. The points in the space are N-D feature vectors encoded by the parameters of the primitives. In addition, we design rules to prune inefficient and invalid configurations in the space.

Table 1. Storage format and schedule primitives.

Primitive Type	Name	Description
Format	FSplit	Divide tensor axis into several sub-axes.
	FReorder	Change storage orders of axes.
	Mode	Set format property for an axis.
Schedule	LSplit	Divide a loop into several sub-loops.
	LReorder	Change execution orders of loops.
	Parallelize	Executing loops in parallel.
	Vectorize	Leverage vector instruction parallelism.
	Unroll	Unroll a loop with factor.

Back-end. We adopt an adaptive exploration scheme combining DQN with heuristic algorithms on the back-end. Relying on DQN, we learn and make decisions about navigational actions in a hyperscale design space. Subsequently, we propagate the next generation population through simulated annealing algorithm. Finally, back-end finds the optimal format and schedule for the target sparse data and hardware platform.

3 Evaluation

We evaluate the auto-tuning framework using the typical sparse tensor operator SpMM on both Intel Xeon E5-2020 v4 and AMD EPYC 7543 platforms. Experiments use TACO [3] and ICC compiler 2021.3.0 as the code generation back-end. Baseline selects the state-of-the-art high-performance operator library MKL, the manual optimization scheme ASpT [2], the auto-tuning framework TVM-S [1] and WACO [7]. TVM-S is a routine on SpMM provided by TVM, and we perform 1000 rounds of tuning. We reproduce the ANNS search scheme of WACO using a pre-training dataset and perform tuning for 100 rounds.

Experiments were conducted on sample of matrices randomly selected from the SuiteSparse dataset [4] with the number of non-zero elements in the 0.1-5 million range. The operator is tuned for 100 rounds using single precision floating point data. As shown in Fig. 2, our approach obtains average speedups of 1.24-6.34× and 1.19-3.57× on both Xeon and EPYC platforms relative to all baselines, respectively. ASpT can only outperform MKL on 11 data due to the inability of the fixed manual optimization to adapt to different sparse patterns. TVM-S performs poorly in most data scenarios due to it can only be tuned on schedule. WACO almost always outperforms other baselines due to its coordinated optimization in format and schedule. We have further improved compared to WACO. MKL achieves optimal performance on the EPYC platform for 4 data, but performs poorly on the Xeon platform for the same data scenarios.

In fact, MKL uses more low-level manual associated with hardware characteristics optimization methods that are effective in all cases, but the impact of format and schedule is more significant. This means that in most cases, the benefits brought by low-level manual methods cannot offset the performance loss

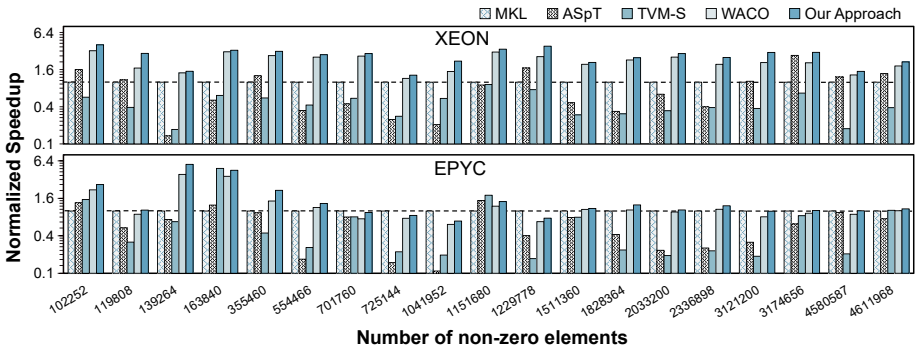

Fig. 2. Normalized speedups compared to MKL on different sparse data and CPU.

caused by inefficient format and schedule. Therefore, our optimization of format and schedule is of great significance, and further in-depth low-level optimization based on it can theoretically achieve the optimal performance. Attempting to further analyze manual low-level optimization methods and integrate them into framwork will be our future work.

4 Conclusion and Future Work

We propose a source-to-source sparse tensor program auto-tuning framework and achieved significant performance improvements on the CPU platform. In the future, we plan to extend the framework to support GPUs and domain-specific FPGA accelerators [5] [6].

Acknowledgments. This work was supported in part by the National Key R&D Program of China under Grants 2022YFB4501600 and 2022YFB4501603.

References

1. Chen, T., et al.: Tvm: An automated end-to-end optimizing compiler for deep learning. In: Proc. of OSDI. pp. 578–594 (2018)
2. Hong, C., et al.: Adaptive sparse tiling for sparse matrix multiplication. In: Proc. of PPoPP. pp. 300–314 (2019)
3. Kjolstad, F., et al.: The tensor algebra compiler. Proceedings of the ACM on Programming Languages **1**(OOPSLA), 1–29 (2017)
4. Kolodziej, S.P., et al.: The suitesparse matrix collection website interface. Journal of Open Source Software **4**(35), 1244 (2019)
5. Wang, C., et al.: A ubiquitous machine learning accelerator with automatic parallelization on fpga. IEEE Transactions on Parallel and Distributed Systems (2020)
6. Wang, C., et al.: Wookong: A ubiquitous accelerator for recommendation algorithms with custom instruction sets on fpga. IEEE Transactions on Computers (2020)
7. Won, J., et al.: Waco: learning workload-aware co-optimization of the format and schedule of a sparse tensor program. In: Proc. of ASPLOS. pp. 920–934 (2023)