DynaFlow: An ML Framework for Dynamic Dataflow Selection in SpGEMM accelerators

Sanjali Yadav, Bahar Asgari
University of Maryland, College Park

***Abstract*—Sparse matrix-matrix multiplication (SpGEMM) is a critical operation in numerous fields, including scientific computing, graph analytics, and deep learning, leveraging matrix sparsity to reduce both storage and computation costs. However, the irregular structure of sparse matrices poses significant challenges for performance optimization. Existing hardware accelerators often employ fixed dataflows designed for specific sparsity patterns, leading to performance degradation when the input deviates from these assumptions. As SpGEMM adoption expands across a broad spectrum of sparsity workloads, the demand grows for accelerators capable of dynamically adapting their dataflow schemes to diverse sparsity patterns. To address this, we propose DynaFlow, a machine learning-based framework that trains on the set of dataflows supported by any given accelerator and learns to predict the optimal dataflow based on the input sparsity pattern. By leveraging decision trees and deep reinforcement learning, DynaFlow surpasses static dataflow selection approaches, achieving up to a 50× speedup.**

***Index Terms*—SpGEMM accelerators, Dynamic Dataflow Selection, Reinforcement Learning, Decision Trees**

I. Introduction

Sparse matrix-matrix multiplication (SpGEMM) is a foundational kernel for a wide range of applications, including scientific computing, graph analytics, and machine learning. Sparse computations leverage the structure of input matrices by efficiently discarding ineffectual zero elements, thereby reducing storage overhead and computational load. However, the irregular structure of sparse matrices often leads to underutilized compute resources and memory bandwidth, posing significant challenges for fast and efficient processing.

To address the inefficiency challenges associated with sparsity, recent studies have developed various specialized hardware accelerators customized for the three widely recognized SpGEMM execution dataflow schemes: inner product (IP) [1], [2], outer product (OP) [3], [4], and row-wise product (RW) [5], [6]. These state-of-the-art hardware accelerators, however, are optimized for specific sparsity patterns that best utilize their underlying hardware architecture. They employ a fixed execution dataflow scheme. As a result, the performance is sub-optimal if the sparsity of the workload does not align with the rigid design of the accelerator.

To overcome these limitations, more recent designs aim to build versatile accelerators capable of adapting to a wide range of sparsity regimes. For instance, Trapezoid [7] introduces novel dataflows schemes to support a broad range of sparsity levels, while Flexagon [8] employs a reconfigurable architecture to handle diverse sparse workloads. Although these architectures represent an important step toward general-purpose sparse acceleration, they still lack key capabilities. In particular, neither provides a robust mechanism for selecting the optimal dataflow based on the input matrices. Flexagon relies on a simple offline profiling method, deferring the challenge of comprehensive dataflow selection to future work. Trapezoid, while supporting multiple dataflows, offers no dynamic strategy for selecting among them at runtime.

The problem of dynamic dataflow selection aligns naturally with machine learning (ML) techniques commonly used in data classification. By extracting features from sparse input matrices, it becomes possible to classify them into categories corresponding to different dataflow schemes. Building on this idea, DynaFlow introduces an ML-based framework for dynamic sparse acceleration, offering an adaptive and effective alternative to static hardware solutions. DynaFlow can be integrated with any accelerator that supports multiple dataflow schemes, including standard ones such as IP, OP, and RW, as well as novel designs such as Trapezoid's. The framework is supported by a dataset of 19,000 sparse matrices with varying sparsity levels, sourced from SuiteSparse [9] and pruned DNN models, and includes a hardware-agnostic feature set.

DynaFlow offers two predictive models: a lightweight decision tree for efficient deployment and a reinforcement learning (RL) model that captures more complex relationships for improved accuracy. The framework trains on the accelerator's available dataflows and learns to quickly predict the most suitable scheme for a given input. In a case study using an accelerator supporting IP, OP, and RW dataflows, a lightweight 512B DynaFlow model achieved up to a 50× speedup over static dataflow selection. When applied to Trapezoid's architecture, the framework attains 88% accuracy in predicting the optimal dataflow, highlighting its potential to significantly improve performance in dynamic sparse workloads.

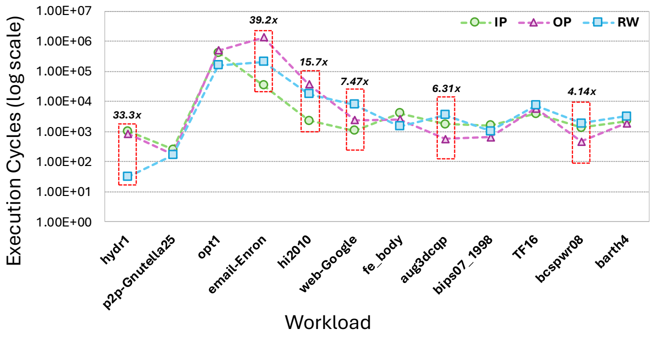

Fig. 1: **The Impact of Dataflow Selection -** Performance gains from dynamic dataflow switching.

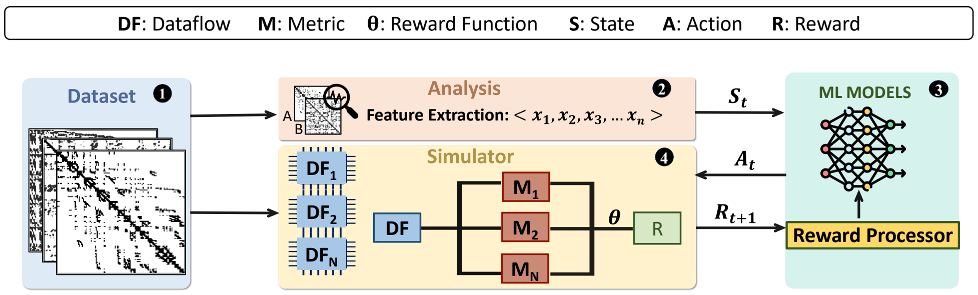

Fig. 2: **DynaFlow High-Level Framework-** Extracting features from sparse matrices, evaluating performance through simulation, and updating ML models for efficient runtime decisions.

II. Dynamic Dataflow Selection

We present the primary motivation for dynamically selecting dataflows by simulating an accelerator that supports IP, OP, and RW dataflow modes. To evaluate performance, we use matrices from the SuiteSparse collection [9], each multiplied by its own transpose. These matrices span a wide range of dimensions, sparsity levels, and structural patterns. Figure 1 shows the total execution cycles for each dataflow mode across these diverse workloads.

As illustrated in Figure 1, no single dataflow consistently achieves optimal performance across all workloads. Each matrix exhibits distinct sparsity characteristics, such as density and pattern regularity, that critically influence dataflow efficiency. Selecting the optimal dataflow can yield substantial speedups, while a static accelerator limited to one dataflow cannot fully exploit the diverse sparsity structures. Each dataflow is optimized to maximize operand reuse at different granularities. For instance, RW benefits from higher reuse in matrix B when it is denser, whereas IP suffers when both matrices are highly sparse because of limited intersections between rows and columns. Similar patterns hold for newer Trapezoid-based dataflows, where each dataflow excels only for specific workload types.

Recognizing these sparsity patterns and dynamically adapting the dataflow at runtime enables significant performance gains. By tailoring dataflow selection to the workload's structural characteristics, a dynamic accelerator can substantially outperform static configurations, highlighting the critical role of flexible dataflow strategies in sparse matrix processing.

III. Workflow of DynaFlow

To train and evaluate our machine learning models, we require a dataset of sparse matrices along with their characteristic features. As no such dataset exists, we constructed one using real-world matrices from the SuiteSparse collection [9], which spans domains such as electromagnetics, fluid dynamics, and chemical process simulation. These matrices vary in size and sparsity patterns. To further diversify the dataset, we also included matrices extracted from pruned layers of deep neural networks (DNNs) at different sparsity levels.

Figure 2 illustrates DynaFlow's workflow. DynaFlow takes as input a simulator that models various dataflows (DF) and outputs performance metrics (M) such as latency, energy consumption, and resource utilization. These metrics evaluate dataflow efficiency on given workloads. For demonstration, we developed a sparse matrix multiplication simulator supporting IP, OP, and RW dataflows, reporting latency and processing element utilization. This simulator is integrated into DynaFlow to initiate model training, which can target either a decision tree or a reinforcement learning model.

The training process begins by reading matrices from the dataset ❶ and extracting their features ❷. Features for newly added matrices are computed in real time. These features are packed into a state vector S_t, which is fed into the model ❸. During early training, when model weights are uninitialized, the agent operates in an exploration phase, randomly selecting actions A_t, each corresponding to a different dataflow. As training progresses, the model enters the exploitation phase, making decisions based on updated weights. The selected action A_t is passed to the simulator, which runs the chosen dataflow and reports performance metrics ❹. A reward function (θ) then computes a scalar reward based on these metrics, guiding model updates. The reward function is configurable, allowing different weights for optimizing specific objectives.

In contrast, training the decision tree model follows a supervised approach. Each matrix is evaluated under all available dataflows, and performance metrics are collected. The dataflow with the best performance is labeled as the optimal choice, creating a labeled dataset, which trains the decision tree to map matrix features to optimal dataflows. Once trained, DynaFlow produces a lightweight model capable of making fast, informed dataflow decisions at runtime on real-world workloads.

IV. Evaluations & Results

A. Feature Selection

Our feature set, detailed in Table I, comprises 11 distinct features, counting those for matrices A and B separately. The table also highlights the specific characteristics each feature captures in terms of sparsity and structural patterns. The overarching goal of this feature set is to characterize the relationship between input matrix properties and dataflow efficiency. For instance, in our simulator, the RW dataflow performs better when the variation in column length is low, as high variance implies more irregular and potentially costly memory accesses to matrix B. Importantly, these features remain relevant even when new dataflows are introduced, as they reflect general patterns in how rows and columns are scheduled during computation. This generality allows the same feature set to be used across different dataflows; for

TABLE I: Features of our dataset.

Feature	Description	Captures
Sparsity of A & B	Total nonzero in matrix / size of matrix	Overall sparsity level
Average row length of A & B	Average number of nonzero per row	Data distribution across rows
Average column length of A & B	Average number of nonzero per column	Data distribution across columns
Average row length variance of A & B	Variance in average number of nonzero per row	Irregularity and imbalance among rows
Average column length variance of A & B	Variance in average number of nonzero per column	Irregularity and imbalance among columns
Blocks accessed of B	Blocks of B accessed not in on-chip memory	Data movement from off-chip memory

example, we successfully reused this set to train models for the Trapezoid dataflow.

To support lightweight machine learning models, DynaFlow includes a mechanism to automatically prune less impactful features during training. We first train a decision tree on the full feature set, measure feature importance, and select the most influential features, as shown in Figure 3. Based on this analysis, five features are retained: blocks accessed, average row length variance of A, average column length of B, and the sparsities of A and B. Reducing the feature set lowers model metadata and storage requirements, enabling more efficient deployment, while retaining the full set may be preferable when maximizing predictive accuracy is the primary goal.

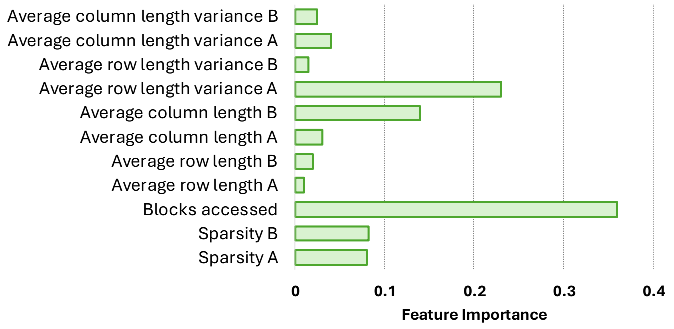

Fig. 3: **Feature Selection –** Analysis of the feature importance in the decision tree.

B. *Decision Tree Model*

We evaluate our framework by first training a decision tree model on our simulator, using the pruned feature set. The dataset is partitioned into 70% training and 30% testing samples. To assess the model's performance and robustness, we employ k-fold cross-validation. The decision tree achieves an accuracy of 90%, with a precision score of 88% and a recall score of 89%, all computed using a weighted average. Additionally, the final model requires only 512 bytes of storage, making it highly efficient for deployment in resource-constrained environments.

Figure 4 compares the performance of decision tree predictions against the baseline IP, OP, and RW dataflows on the test set. Each subfigure shows the speedup achieved by selecting the predicted dataflow over using a fixed static dataflow. The red line denotes a speedup of 1, indicating cases where the predicted and static dataflows perform equally—that is, the static dataflow was already optimal and correctly identified. Speedups greater than 1 reflect cases where the predicted dataflow outperformed the static one, while values below 1 indicate a less optimal choice. In most cases, the speedup is equal to or greater than 1, demonstrating the model's ability to accurately predict and improve dataflow selection.

In terms of performance gains, the trained decision tree model achieves average (geometric mean) speedups of **1.93**$\times$ over IP, **1.27**$\times$ over OP, and **2.28**$\times$ over RW across the dataset. These results indicate that, rather than relying on a static accelerator with a fixed execution dataflow, dynamically selecting the dataflow based on the characteristics of the input matrices can lead to substantial performance improvements.

C. *Reinforcement Learning Model*

We also train a RL model to compare its performance with that of the decision tree. The core of the RL model is a neural network that maps the current state (i.e., features of the input) to an appropriate action. To minimize storage requirements, the network is designed with a single hidden layer and comprises 9,219 parameters. Despite these optimizations, the RL model requires 38 KB of storage, which is expected due to the additional metadata necessary to support learning and exploration during training. Consequently, in terms of storage footprint, the decision tree remains the more lightweight option.

Similar to the decision tree evaluation, we partition the dataset into training and testing sets to assess the RL model's performance gains. Figure 5 presents the performance of the RL model across the IP, OP, and RW dataflows. The subfigures are configured identically to those used in the decision tree evaluation. We observe that a greater number of data points lie above the red line, with fewer below it, compared to the decision tree evaluation. This trend indicates that the RL model has learned a more efficient mapping from input features to optimal dataflows, likely due to its ability to store and leverage additional metadata. On average, the RL model achieves substantial improvements, with geometric mean speedups of **3.77**$\times$ over IP, **2.48**$\times$ over OP, and **4.46**$\times$ over RW. Therefore, while the RL model outperforms the decision tree by better capturing complex feature–dataflow relationships, it does so at the cost of higher storage requirements. Depending on system constraints and performance goals, either model may be preferable for deployment.

V. Conclusions and Discussions

This paper set out to explore how machine learning can optimize dataflow selection for SpGEMM. We investigated two complementary approaches: decision trees and reinforcement learning models. The methodology we developed for identifying the optimal dataflow is designed to be versatile, enabling adaptation across a wide range of hardware accelerators that dynamically select dataflows based on workload characteristics. Although dataflow specific implementations may vary depending on the accelerator architecture, the fundamental workflow aligns with the design principles established in the DynaFlow framework.

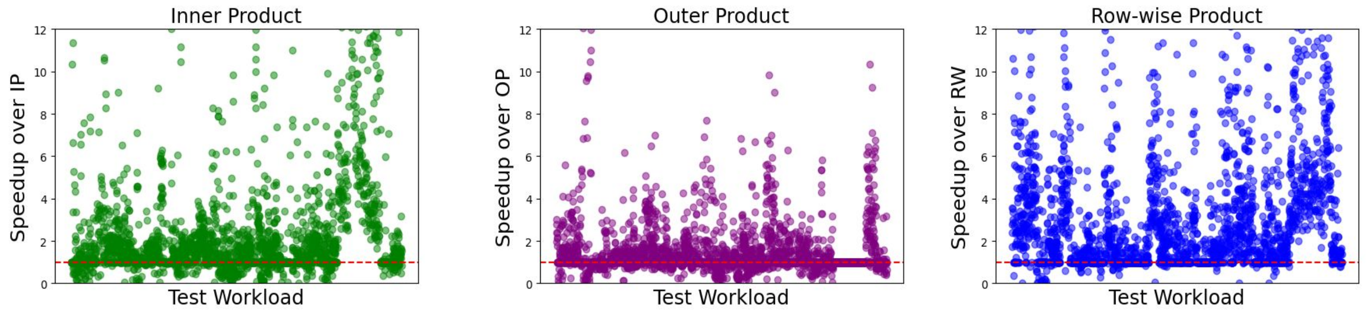

Fig. 4: **Decision Tree Evaluation–** The speedup of the decision tree model over applying IP, OP and RW for SpGEMM operations across all test workload. NOTE: Y-axis has been clipped for illustration purposes.

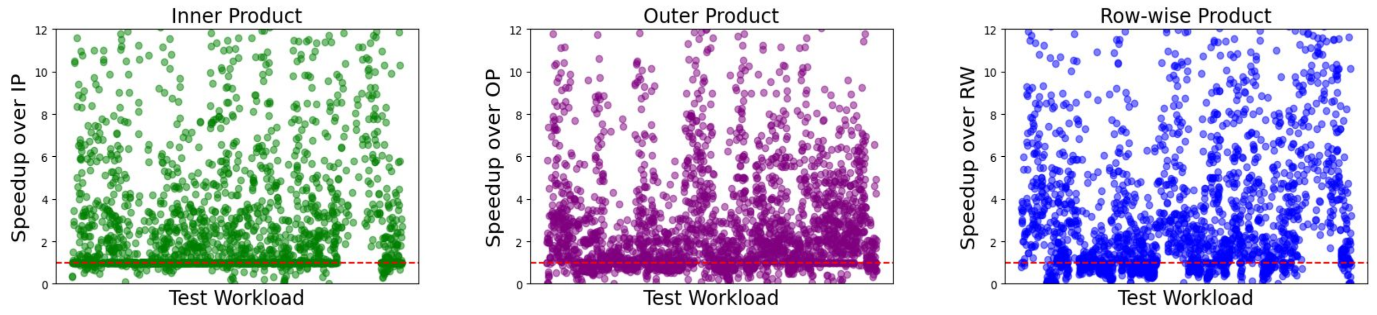

Fig. 5: **Reinforcement Learning Evaluation–** The speedup of the reinforcement learning model over applying IP, OP and RW for SpGEMM operations across all test workloads.

This work represents an important step toward intelligent, adaptive hardware design for SpGEMM acceleration. DynaFlow demonstrates significant advantages over static dataflow configurations, offering users the flexibility to choose between a lightweight decision tree model for minimal storage overhead or a reinforcement learning model that delivers superior speedups at the cost of additional storage. By providing a unified framework for data-driven dataflow selection, we set the foundation for further innovations in adaptive computing systems such as Acamar [10]. Our ongoing work continues to extend these ideas, pushing the frontiers of efficient, intelligent hardware-software co-design.

Acknowledgments

This work is supported by the U.S. DoE, Office of Science, ACSR, the ECRP program, under Award DE-SC0024079.

References

[1] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[2] D. Baek, S. Hwang, T. Heo, D. Kim, and J. Huh, "Innersp: A memory efficient sparse matrix multiplication accelerator with locality-aware inner product processing," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 116–128.

[3] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 724–736.

[4] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 261–274.

[5] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 766–780.

[6] G. Zhang, N. Attaluri, J. S. Emer, and D. Sanchez, "Gamma: leveraging gustavson's algorithm to accelerate sparse matrix multiplication," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 687–701. [Online]. Available: https://doi.org/10.1145/3445814.3446702

[7] Y. Yang, J. S. Emer, and D. Sanchez, "Trapezoid: A versatile accelerator for dense and sparse matrix multiplications," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2024, pp. 931–945. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISCA59077.2024.00072

[8] F. Muñoz Martínez, R. Garg, M. Pellauer, J. L. Abellán, M. E. Acacio, and T. Krishna, "Flexagon: A multi-dataflow sparse-sparse matrix multiplication accelerator for efficient dnn processing," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 252–265. [Online]. Available: https://doi.org/10.1145/3582016.3582069

[9] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, dec 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663

[10] U. Bakhtiar, H. Hosseini, and B. Asgari, "Acamar: A dynamically reconfigurable scientific computing accelerator for robust convergence and minimal resource underutilization," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1601–1616.