Optimizing Sparse Linear Algebra Through Automatic Format Selection and Machine Learning

Christodoulos Stylianou, Michèle Weiland
EPCC, The University of Edinburgh
c.stylianou@ed.ac.uk

Introduction

- Sparse matrices essential concept in computational science and engineering
- Sparse matrix storage formats are different in-memory representations of sparse matrices
  - Each designed to exploit strengths of the different hardware architectures or sparsity pattern of the matrix

- More than 70 formats have been developed over the years - still no single one performs best across:
  - Different sparsity patterns
  - Different target architectures
  - Different operations

- Most code-bases today still use a single format (CSR)
  - ➢ Adapting the data structure at run-time offers new optimization opportunities

Sparse Matrix Storage Formats

	0	1	2	3	4
0	1	2		11	
1		3	4		
2		5	6	7	
3				8	
4				9	10

(a) Dense Matrix

AI	0	0	0	1	1	2	2	2	3	4	4
AJ	0	1	3	1	2	1	2	3	3	3	4
AV	1	2	11	3	4	5	6	7	8	9	10

(b) COO Representation

IRP	0	3	5	8	9	11

AJ	0	1	3	1	2	1	2	3	3	3	4
AV	1	2	11	3	4	5	6	7	8	9	10

(c) CSR Representation

DOFF	-1	0	1	3

AV

*	1	2	11
0	3	4	0
5	6	7	*
0	8	0	*
9	10	*	*

(d) DIA Representation

AJ

0	1	3
1	2	*
1	2	3
3	*	*
3	4	*

AV

1	2	11
3	4	*
5	6	7
8	*	*
9	10	*

(e) ELL Representation

Morpheus: A Library for Dynamic Sparse Matrices

- Templated C++ library
- Functional Design
  - Containers & Algorithms
- Data Management
- Support for Heterogeneous Platforms
  - Host-Device Model
  - Mirroring
- Efficient dynamic switching
- Continuous addition of new formats and backends under the same interface.
  - Increased life-time of software
  - Current developments support 6 formats:
    - COO, CSR, DIA, ELL, HYB, HDC

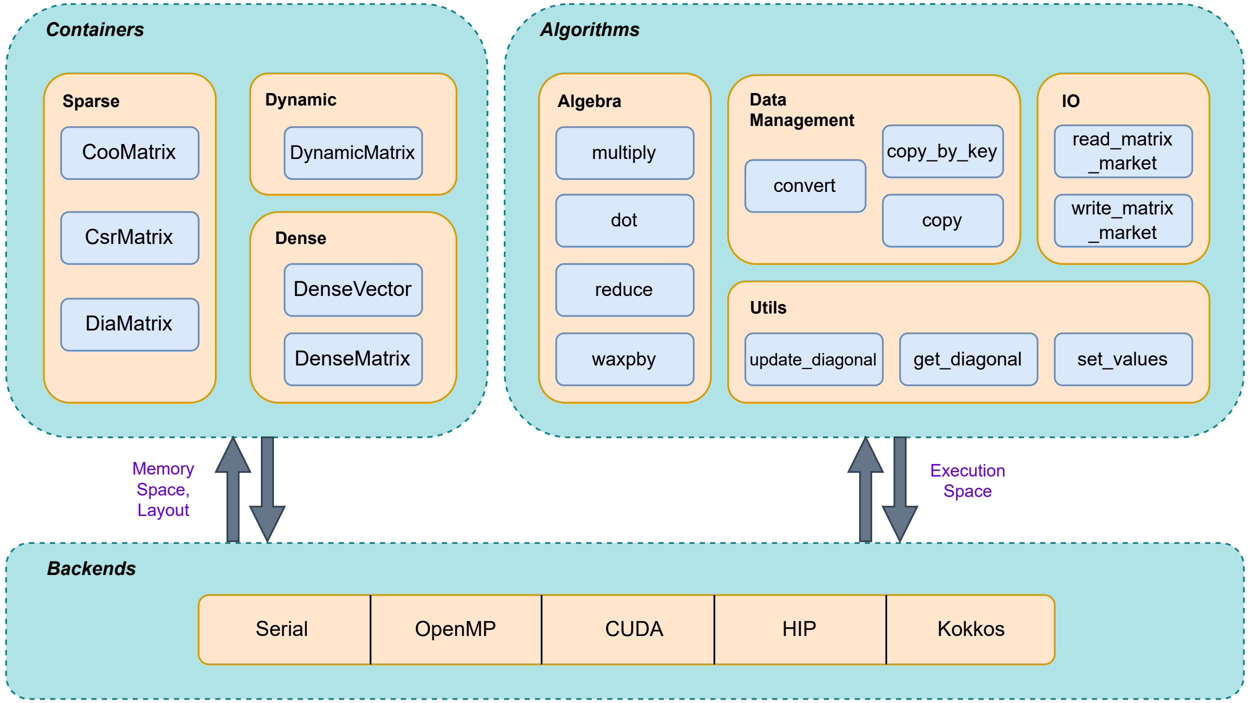


Figure 1: High-level overview of Morpheus v1.0.0.

Link to *Morpheus*: https://github.com/morpheus-org/morpheus

Motivation

- New formats are proposed every time a new architecture emerges
  - Aim to exploit the new characteristics and features of the new hardware.
- In the era of heterogeneous computing, hardware has become more diverse
  - Applications often require the **use of multiple formats** across the **different types of hardware** to remain optimal.
- Still, **no single format** would **perform optimally** across sparsity patterns, operations and hardware architectures.
  - ➢ Select the optimal format from a pool of candidate formats at runtime.

- Experience users may have a feeling about the choice of the optimal format for a category/type of matrices they frequently use.
  - However, a decision as such is not always trivial.
- Choosing the optimal format by running the available options first can result in prohibitive overheads.
- Adopting a Machine Learning (ML) model has the potential to offer an **accurate** and **low-overhead solution** to the problem of **automatic format selection**.

Auto-tuning Pipeline: High-Level Overview

- The focus of this work is to develop an **auto-tuner for selecting the optimal format to switch to**, given a matrix, an operation and target hardware.
- Most accurate prediction can be optained by utilizing a run-first approach.
  - Requires **multiple** expensive conversions.
- A better approach that reduces the prediction cost is to use ML models, **by relaxing the accuracy requirements**.
- The pipeline is divided in the **offline** (red) and **online** (green) stages.

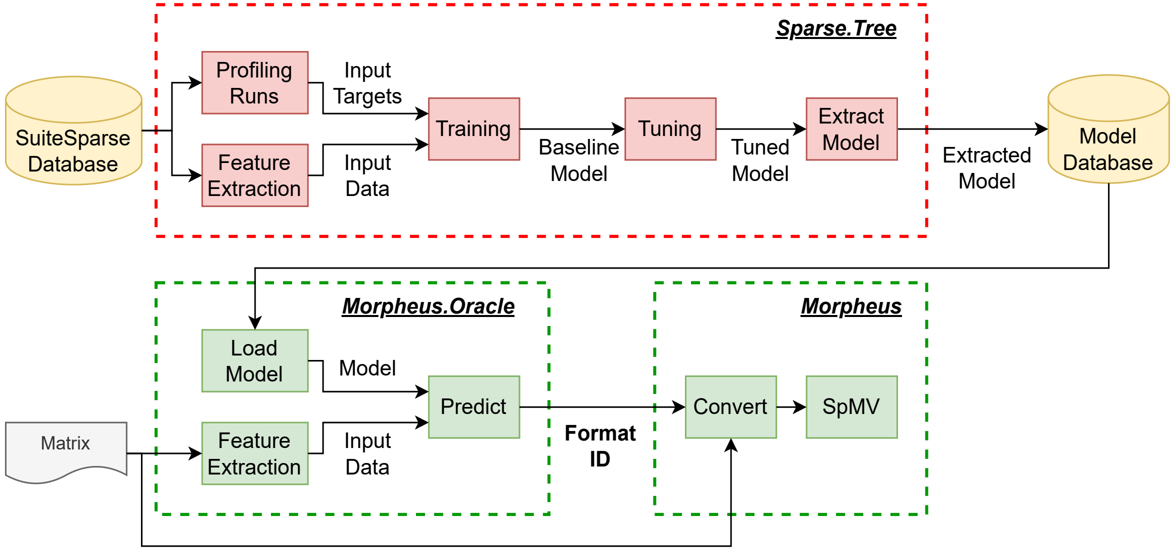


Figure 2: High-level overview of the auto-tuning pipeline. Red and green boxes represent offline and online operations respectively.

Auto-tuning Pipeline: Offline Stage

- Database of 2200 real-valued and square matrices from SuiteSparse Collection
  - varying sizes, sparsity patterns and application domains
- For every matrix, we obtain the **optimal format** (*input targets*) through profiling runs.
- The *input data* for training are generated by performing **feature extraction**.
- **Offline** stage responsible for **train, tune** and **extract** the ML model in a file.
- For **each architecture and operation** of interest a **different ML model** is generated.
- Process is streamlined by wrapping the offline pipeline in a *Python* framework (*Sparse.Tree*).

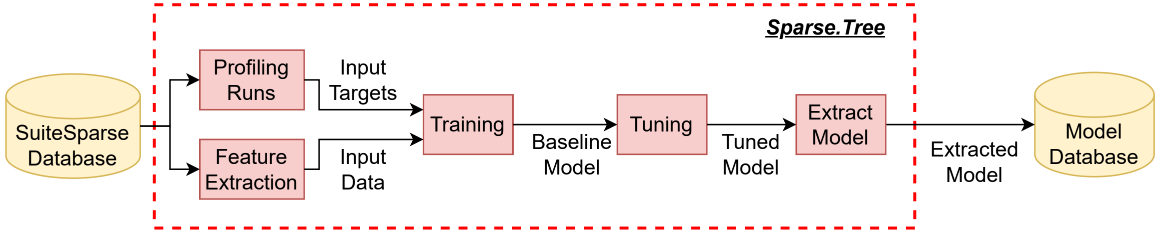


Figure 2a: Offline stage of the auto-tuning pipeline

Link to *Sparse.Tree*: https://github.com/morpheus-org/sparse.tree

Auto-tuning Pipeline: Online Stage

- To be able to select the optimal format in *Morpheus*, we need to be able to **make the decision efficiently and online**.
- The online stage employes *Morpheus-Oracle*
  - *C++ architecture-independent* auto-tuner.
- *Oracle* is responsible for **predicting the optimal format** by:
  - **loading the ML model from file**
  - **performing feature extraction**, in the same way as during the offline stage.
- The **optimal format ID** is then **passed to *Morpheus*** to perform the **runtime switching**.

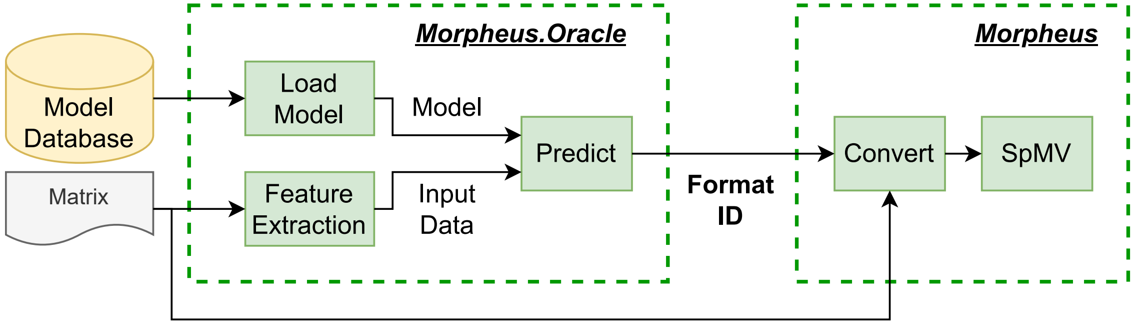


Figure 2b: Online stage of the auto-tuning pipeline

Link to *Morpheus-Oracle*: https://github.com/morpheus-org/morpheus-oracle

Feature Extraction

- The process of transforming the original sparse matrix into a set of numerical "features".
- Features can be processed by the model while preserving the information about the sparsity pattern of the original matrix.
- **Trade-off** between the **overheads** required for **computing** these features and the **accuracy** of the decision that is made based them.
- For this work, a set of 10 features was selected that captures information about the:
  - Basic structure of the sparse matrix
  - Distribution of non-zeros across the rows
  - Distribution of non-zeros across the diagonals.

Parameter	Description	Formula
M	# of rows	-
N	# of columns	-
NNZ	# of non-zeros	-
$\overline{NNZ}$	avg. NNZ per row	$\overline{NNZ} = \frac{NNZ}{M}$
ρ	density	$\rho = \frac{NNZ}{M*N}$
$max(NNZ)$	max NNZ per row	$max(NNZ) = max_{i=1}^{M} NNZ_i$
$min(NNZ)$	min NNZ per row	$min(NNZ) = min_{i=1}^{M} NNZ_i$
σ_{NNZ}	std of NNZ per row	$\sigma_{NNZ} = \frac{\sum_{i=1}^{M} \lvert NNZ_i - \overline{NNZ}\rvert^2}{M}$
N_D	# of diagonals	-
N_{TD}	# of true diagonals	-

Table 1: Feature parameters used for training the model and, where relevant, the corresponding formula used for computing each one.

Machine Learning Model (i)

- Our aim is to train a model that can **predict the optimal storage format** of a given sparse input matrix.
- This type of problem falls into the category of **multi-class classification problems**.
- The objective of the model is to try and determine a **mapping between the input features and the optimal format ID**:

$$f(\vec{x_1}, \vec{x_2}, \dots, \vec{x_n}) \to y_n(COO, CSR, \dots, HDC)$$

where $\vec{x}_i$ represents the feature vector of the i_{th} sparse matrix in the training set and y_n represents the target vector with each entry containing the index of one format from the six available.

- Training is done using a **decision tree** ML algorithm that effectively learns simple decision rules inferred from the data features.
  - Simple to understand and interpret this method.
  - Requires little to **no data preparation** before training the model or using it for prediction.
- To improve the robustness of the model, an **ensemble of decision trees** is built (***Random Forest***).
  - Effectively fits a number of decision tree classifiers onto different sub-samples of the dataset.

Morpheus-Oracle (i)

- *Oracle* is a *C++* library that offers a **systematic way of performing automatic format selection.**
- Developed to **complement the dynamic switching** capabilities in ***Morpheus***
- Follows similar **functional design** philosophy as *Morpheus:*
  - **Containers** (Tuners) & **Algorithms** (Tuning Operations)
- **Tuners** are responsible for:
  1. Encapsulating the specifics of each tuner's implementation
  2. Exposing the user only to an interface that configures and runs the tuner.
- **Tuning operations** are responsible for:
  1. performing the actual tuning process and figuring out the optimal format.
- Currently three tuners are supported:
  - ***RunFirstTuner, DecisionTreeTuner*** and ***RandomForestTuner***:
- The **performance** of each of the three tuners is a **direct trade-off** between **runtime** overhead and prediction **accuracy**.

Experimental Setup

- All experiments were carried out on **three supercomputers**:
  - Archer2, Isambard and Cirrus.
- Experiments run on a representative set of all major hardware architectures:
  - x86 (Intel and AMD) and ARM CPUs
  - NVIDIA and AMD GPUs.
- Dataset uses **2200 real-valued and square matrices**
  - Available from the SuiteSparse library
  - Train-Test Split: **80%-20%.**

SYSTEM	SUBSYSTEM	QUEUE	CPU	GPU
ISAMBARD	A64FX	A64FX	1X FUJITSU A64FX (48 Cores)	-
	P3	INSTINCT	1x AMD EPYC 7543P (32 Cores)	4x AMD Instinct MI100
		AMPERE		4x NVIDIA Ampere A100 40GB
	XCI	ARM	1X MARVELL THUNDERX2 ARM (32 CORES)	-
CIRRUS		STANDARD	2X INTEL XEON E5-2695 (18 CORES)	-
		GPU	2X INTEL XEON GOLD 6248 (18 CORES)	4X NVIDIA VOLTA V100 16GB
ARCHER2		STANDARD	2X AMD EPYC 7742 (64 CORES)	-

Table 2: Node configurations for the systems used in the experiments.

epcc

Format Distribution

- Overall optimal format is *CSR*.
- Even on the same hardware distribution can change drastically between backends:
  - A64FX/Serial : ~50% HDC, DIA and COO
  - A64FX/OpenMP: ~75% CSR.
- Can also stay the same (Archer2 and Cirrus).
- Optimal format very different between MI100 and A100.

- The **format distribution** for every target is **unbalanced**.
  - **Imbalanced classification problem** or **rare event prediction.**
  - ➢ An **auto-tuner** that can **predict rare events** is useful if in the case where selecting a different format benefits performance noticeably.

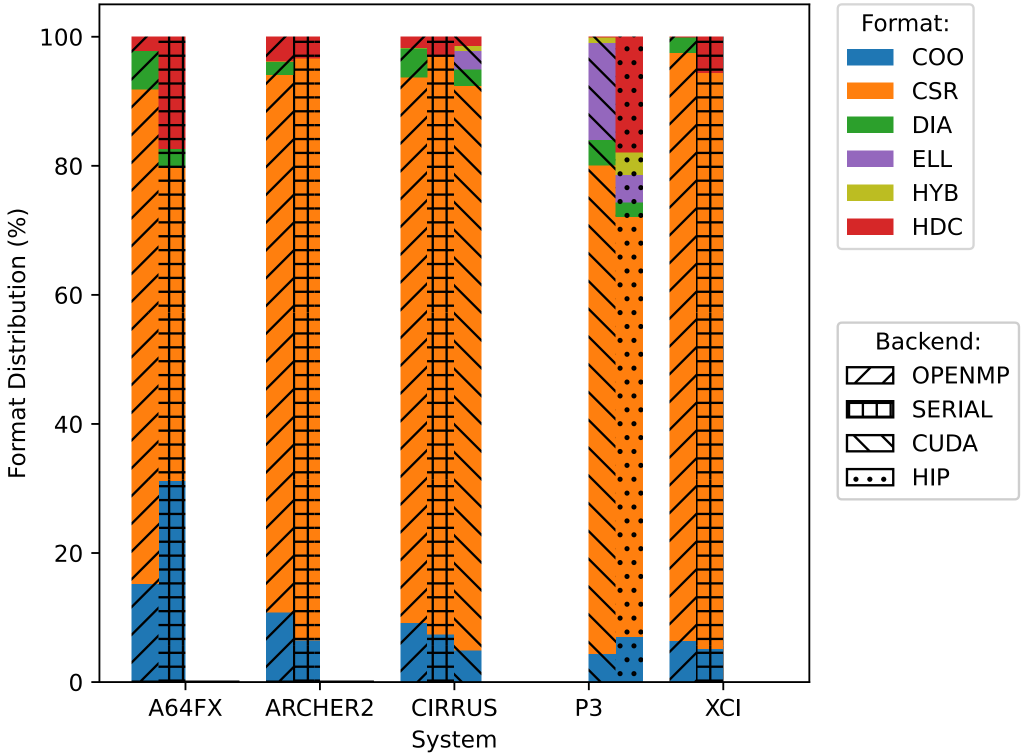


Figure 3: Optimal Format distribution for 1000 repetitions of SpMV using the SuiteSparse dataset. The optimal format for each matrix is selected to be the one with the smallest runtime.

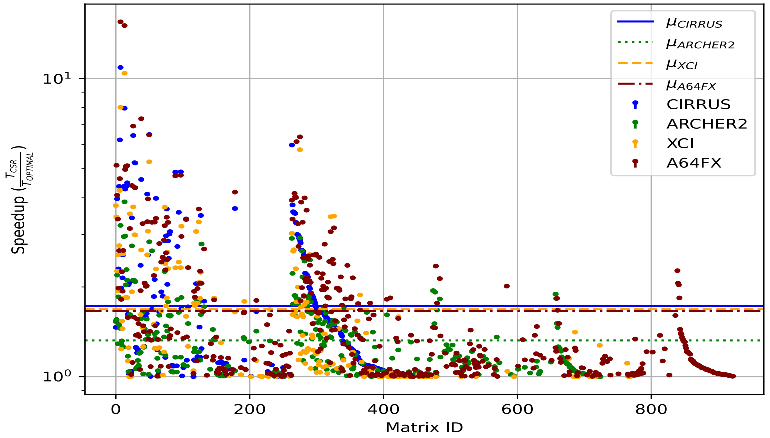
Optimal Format Performance

- Experiment quantifies the **real benefit (speedup) for the SpMV operation when the optimal format is not CSR**.

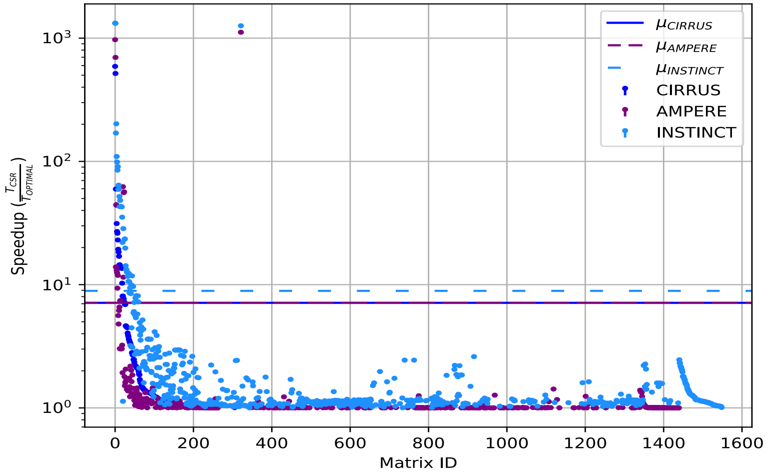
$$Speedup = \frac{T_{CSR}}{T_{OPTIMAL}}$$

- For OpenMP backend, noticeable number of matrices exhibits speedups *1.5×-10.5×*
  - Average speedup of approximately *1.8×* for Cirrus, XCI and A64FX
  - Average speedup of *1.3×* on Archer2
- For the CUDA and HIP backends, runtime speedups are more noticeable compared to the CPU backends
  - The average speedup is *8×* and *10×* – max up to *1000×.*

➢ Results justify the development and use of an auto-tuner
  ➢ must be **lightweight** to avoid performance degradation



(a) OpenMP backend



(b) CUDA and HIP backends

Figure 4: Runtime speedup of SpMV using the optimal format against CSR for the SuiteSparse dataset. *Matrices with optimal format set to CSR are omitted for clarity.*

Hyperparameter Tuning

- To account for overfitting we perform a **5-fold Cross Validation** (CV) on the training set.
- A **grid search** is performed to search for the optimal **hyperparameter** values.
  - e.g. max depth of tree, max number of features, number of estimators etc.
- Metrics of interest: 1) **Accuracy** and 2) **Balanced accuracy** (since dataset is unbalanced).
- Average accuracy and Balanced accuracy scores of the models on the **Test** set:
  - DecisionTree (Tuned): *90.85% ± 7.87%* and *78.12% ± 4.91%.*
  - RandomForest (Baseline): *92.36% ± 2.93% and 80.22% ± 11.04%*
  - RandomForest (Tuned): ***92.63% ± 3.02%** and **84.42% ± 6.64%***
- The tuned models are using significantly fewer and shallower trees → **Faster prediction times**.
- For some system and backend pairs, change in balanced accuracy quite drastic

  e.g. +10% on Cirrus/OpenMP pair.
- The development of both *DecisionTreeTuner* and *RandomForestTuner* is justified.

Auto-tuner Performance

- The tuned classifier is deployed in C++ as a tuner
  - e.g. the *RandomForestTuner* in *Oracle.*
- The benchmark performs 1000 SpMV operations:
  - Optimum format **selected by the tuner** at runtime.
- Benchmark uses the matrices in the test set.
- **Runtime cost of tuning** is measured in terms of SpMV operations in CSR format:

$$C_{tuning} = \frac{T_{CSR}}{T_{FE} + T_{PRED}}$$

- ≥75% of the matrices in the test set require **fewer than 100 repetitions** for the tuning process.

System	Backend	Mean	Std	Min	Max
Archer2	Serial	10	19	2	303
	OpenMP	25	20	2	179
Cirrus	Serial	10	30	2	359
	OpenMP	64	72	2	643
	CUDA	7	3	1	29
A64FX	Serial	6	9	1	120
	OpenMP	45	40	1	246
P3	CUDA	2	3	1	42
	HIP	15	9	1	30
XCI	Serial	12	28	2	335
	OpenMP	17	29	2	203

Table 3: The runtime cost, **expressed as number of SpMV operations using CSR**, of using the auto-tuner.

C_{tuning}: the runtime cost of tuning.
T_{CSR}: the runtime of a single CSR SpMV.
T_{FE}: the runtime of feature extraction.
T_{PRED}: the runtime for prediction.

epcc

Tuned SpMV Performance (i)

- The runtime speedups in SpMV obtained by adopting the auto-tuner compared to SpMV using CSR is given by:

$$Speedup = \frac{T_{CSR}}{T_{TUNE} + T_{OPT}} = \frac{T_{CSR}}{T_{FE} + T_{PRED} + T_{OPT}}$$

- On CPUs, the runtime from using the auto-tuner **on average** is similar to as if we were to use the CSR format.
  - Consistent ~1.1× average speedup across all systems.
  - In many cases the auto-tuning process results in noticeable speedups, with maximum achieved speedup of 7×.
- For the majority of matrices the overheads from the auto-tuner do not reduce overall performance.
  - The few for which performance falls significantly below 1, we do observe the impact from wrongly classifying the optimal format.

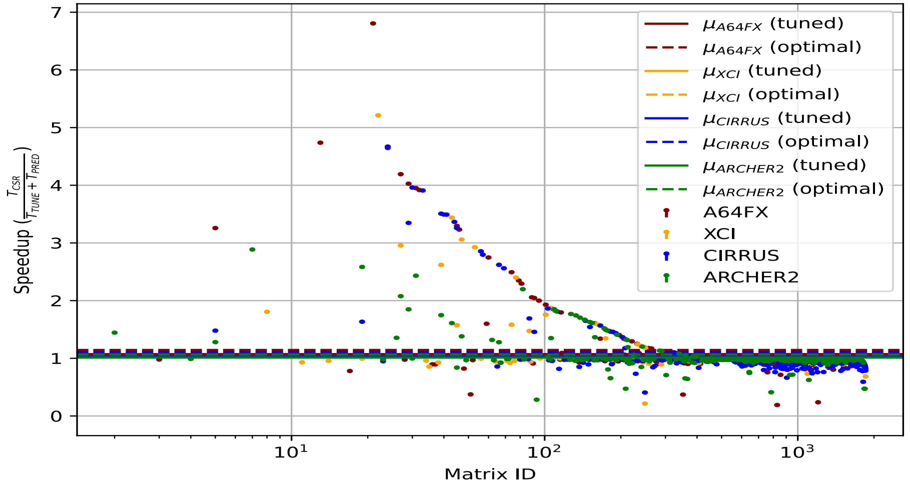


Figure 5: Obtained runtime speedup from using the auto-tuner and predicted format against using CSR in performing 1000 SpMV operations on the available systems (OpenMP backend) for every matrix in the test set.

T_{CSR}: the runtime of 1000 CSR SpMV.
T_{OPT}: the runtime of 1000 Optimal SpMV.
T_{FE}: the runtime of feature extraction.
T_{PRED}: the runtime for prediction.

epcc

Tuned SpMV Performance (ii)

- On the GPU backends **auto-tuning is much more beneficial** with the following average speedups:
  - *1.5*× for the NVIDIA A100 and *3*× for V100 GPUs
  - *8x* for AMD MI100.
- For a number of matrices the achieved speedup improves performance by orders of magnitude.
- For the majority of matrices the overheads from the auto-tuner do not reduce overall performance.
- On GPUs a mis-classification is less severe.

➢ $\mu_{tuned} \cong \mu_{optimal}$

  ➢ The overheads introduced by the auto-tuner become negligible as the number of SpMV repetitions increases.

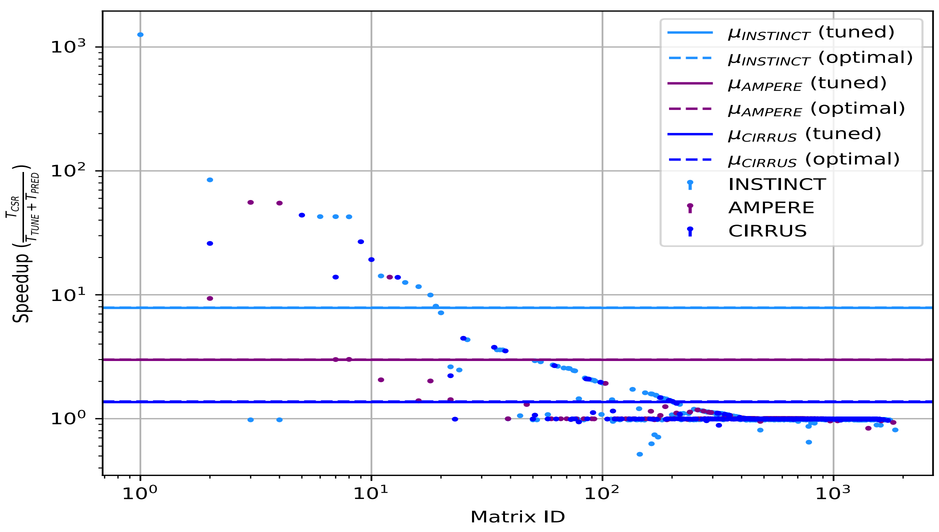


Figure 6: Obtained runtime speedup from using the auto-tuner and predicted format against using CSR in performing 1000 SpMV operations on the available GPU systems (CUDA and HIP backends) for every matrix in the test set.

μ_{tuned}: average speedup with tuning.
$\mu_{optimal}$: average speedup of optimal format without tuning.

Conclusions

- Selecting the optimal sparse matrix storage format is important for allowing applications to remain optimal across the available hardware architectures
  - However, the selection process is not a trivial task.
- ML offers a **systematic solution** to this problem by approaching it as a classification task.
- By training, tuning and deploying an ensemble of *decision trees*, we are able to **accurately predict** the optimum format to be used for the SpMV operation across the main HPC architectures.
- Most of the time the best option is to use CSR
  - In some cases, the runtime is **improved by orders of magnitude** from switching to the optimal format.
- Our proposed light-weight auto-tuning approach introduces overheads in the overall runtime of SpMV
  - **Overheads are amortised quickly** within a few SpMV operations on average (more noticeable benefit to GPUs).
- Further work can explore ways of further improving the accuracy of our models either through balancing the dataset or other ML methods.
- Furthermore, eliminating the manual feature extraction remains an avenue for further research.