Pacific Northwest NATIONAL LABORATORY

HPDC 2025
July 20-23, 2025, Notre Dame, IN, USA

LiteForm: Lightweight and Automatic Format Composition for Sparse Matrix-Matrix Multiplication on GPUs

Zhen Peng (PNNL)
Polykarpos Thomadakis (PNNL)
Jacques Pienaar (Google)
Gokcen Kestor (Barcelona Supercomputing Center)

July 21, 2025

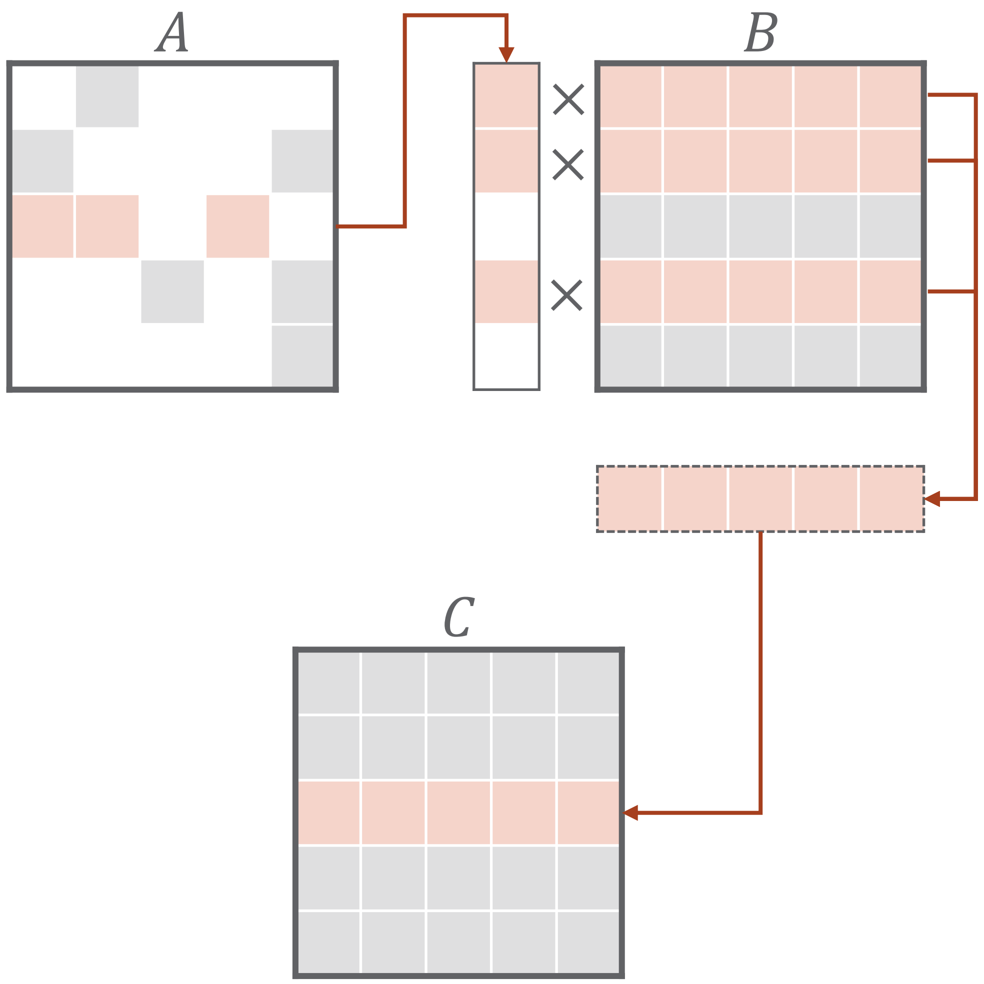
YEARS of SCIENTIFIC BRILLIANCE @PNNL

U.S. DEPARTMENT of ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Motivation: Sparse Computations on GPUs

- GPUs
  - Excel at dense, regular tasks
  - Struggle with sparse ones like SpMM* ($C = A \cdot B$, A sparse, C and B dense)
- Challenges
  - Irregular memory access
  - Load imbalance
  - Warp divergence
- Key
  - Choosing right sparse format is crucial, but matrices vary in sparsity—single format often suboptimal
- Existing issues
  - Fixed formats lack flexibility
  - Composable formats require costly tuning

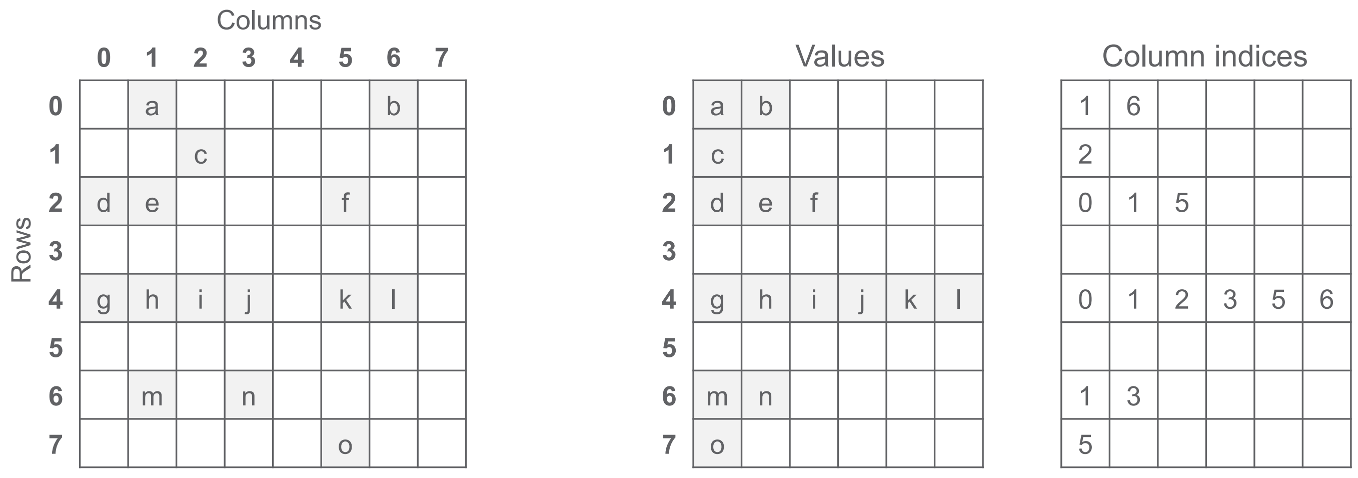


SpMM Computational Pattern

* Sparse matrix-matrix multiplication

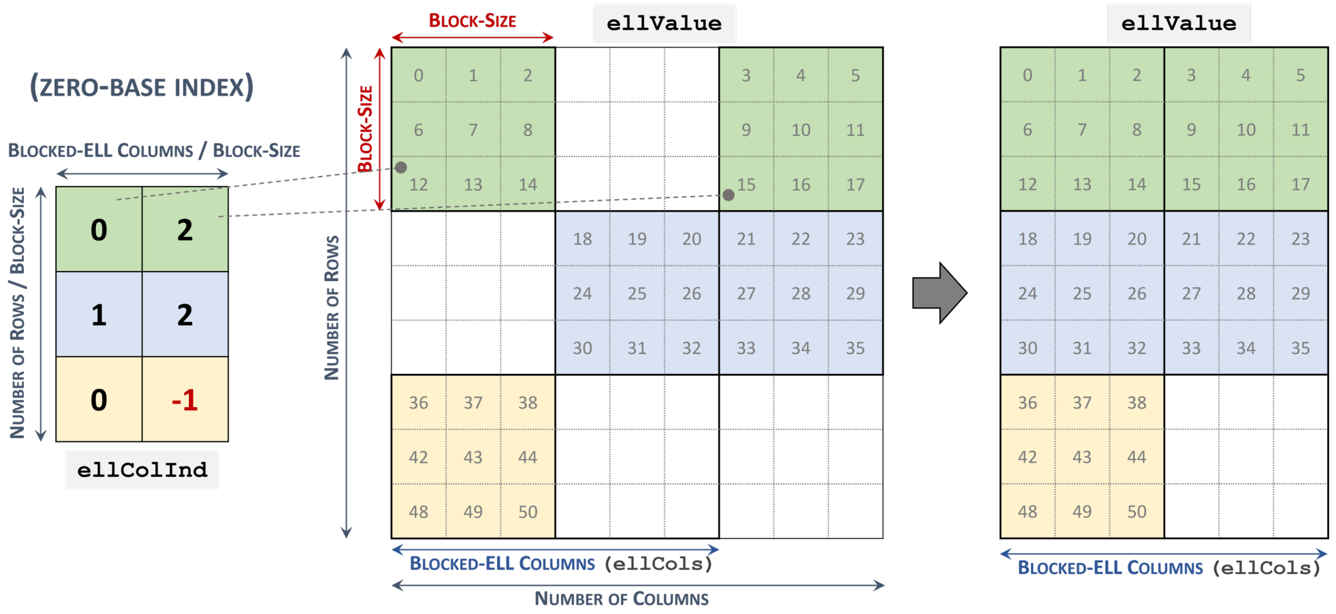
Background: Sparse Formats

- Elementwise formats
  - COO, CSR, Ellpack (ELL)
- Blockwise formats
  - BCSR, Blocked-ELL, Sliced-ELL
- Benefits of blocks
  - Shared memory reuse
  - Aligned access
  - Loop unrolling
- Drawbacks
  - Padding ratio up to 99% → memory explosion
  - Less flexible for various sparsity



Sparse Matrix

Ellpack Format



Blocked Ellpack Format

https://docs.nvidia.com/cuda/cusparse/index.html#blocked-ellpack-blocked-ell

Prior Work and Limitations

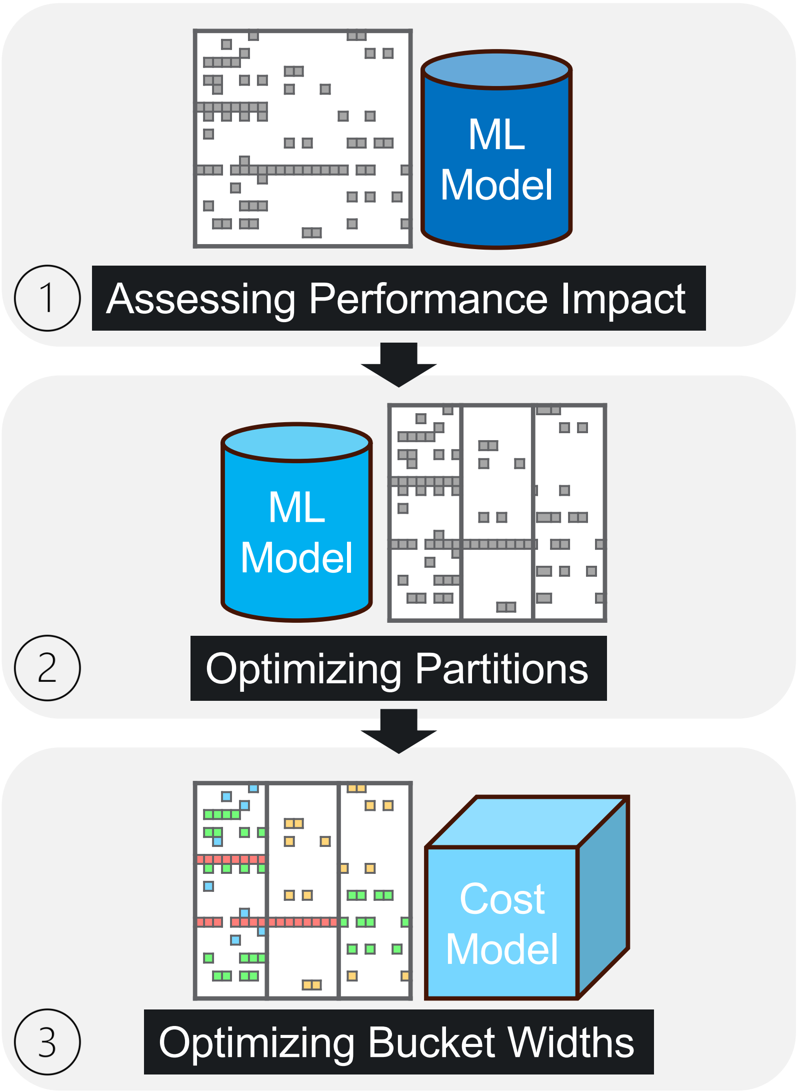
- Fixed Formats
  - cuSPARSE, Triton, etc
  - Optimized but input-dependent
- Auto-Selection
  - Auto-SpMV, Seer, etc
  - Machine learning (ML) picks format but ignores intra-matrix sparsity variations
- Composable Formats
  - SparseTIR, STile
  - Adapt to patterns but high construction overhead (auto-tuning/microbenchmarks)

Type	Work	Auto Format Selection	Sparsity Aware	Format Construction Overhead
Fixed Formats	cuSPARSE[1], Triton[2], etc.	❌	❌	Low
Auto-Selection	Auto-SpMV[3], Seer[4], etc.	✅	❌	Low
Composable Formats	SparseTIR[5], STile[6]	❌ ✅	✅	High
	LiteForm (ours)	✅	✅	Low

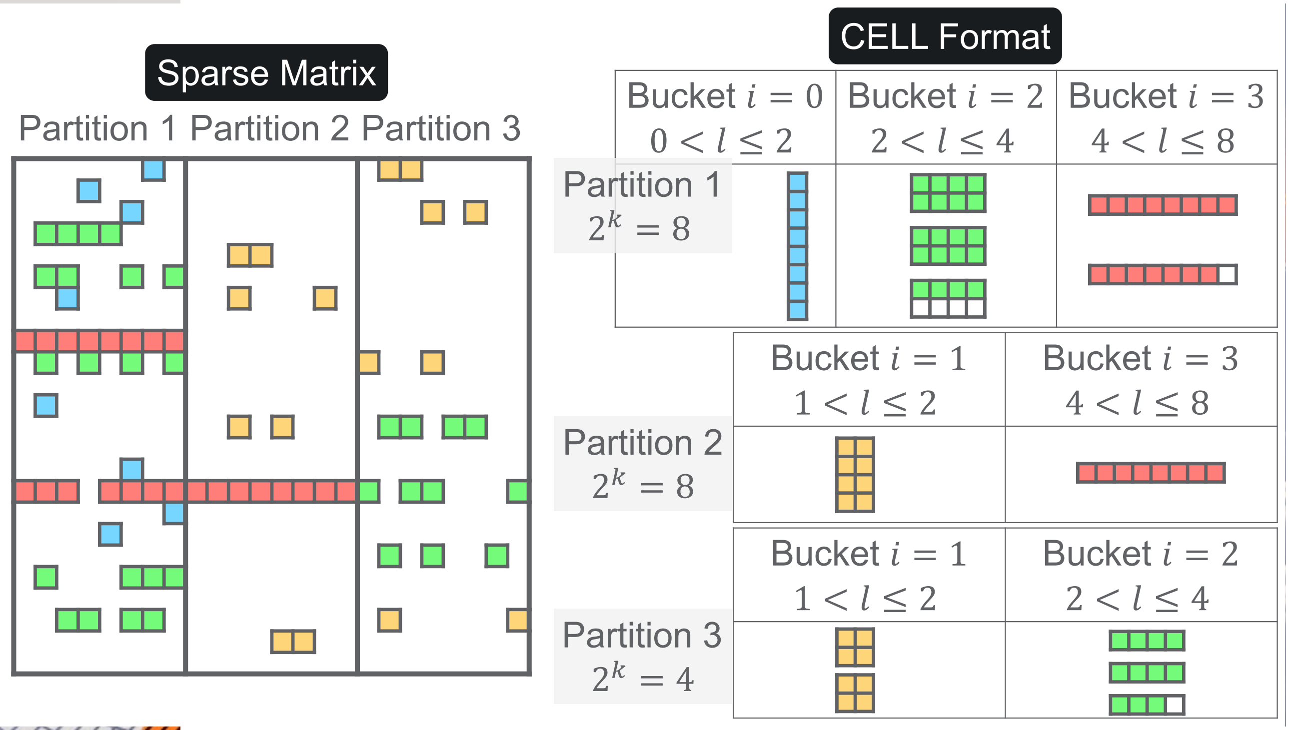
[1] NVIDIA, cuSPARSE
[2] P. Tillet, et al., Triton, MAPL 2019
[3] M. Ashoury, et al., Auto-SpMV, arXiv 2023
[4] R. Swann, et al., Seer, CGO 2024
[5] Z. Ye, et al., SparseTIR, ASPLOS 2023
[6] J. Fang, et al., STile, PACMMOD 2024

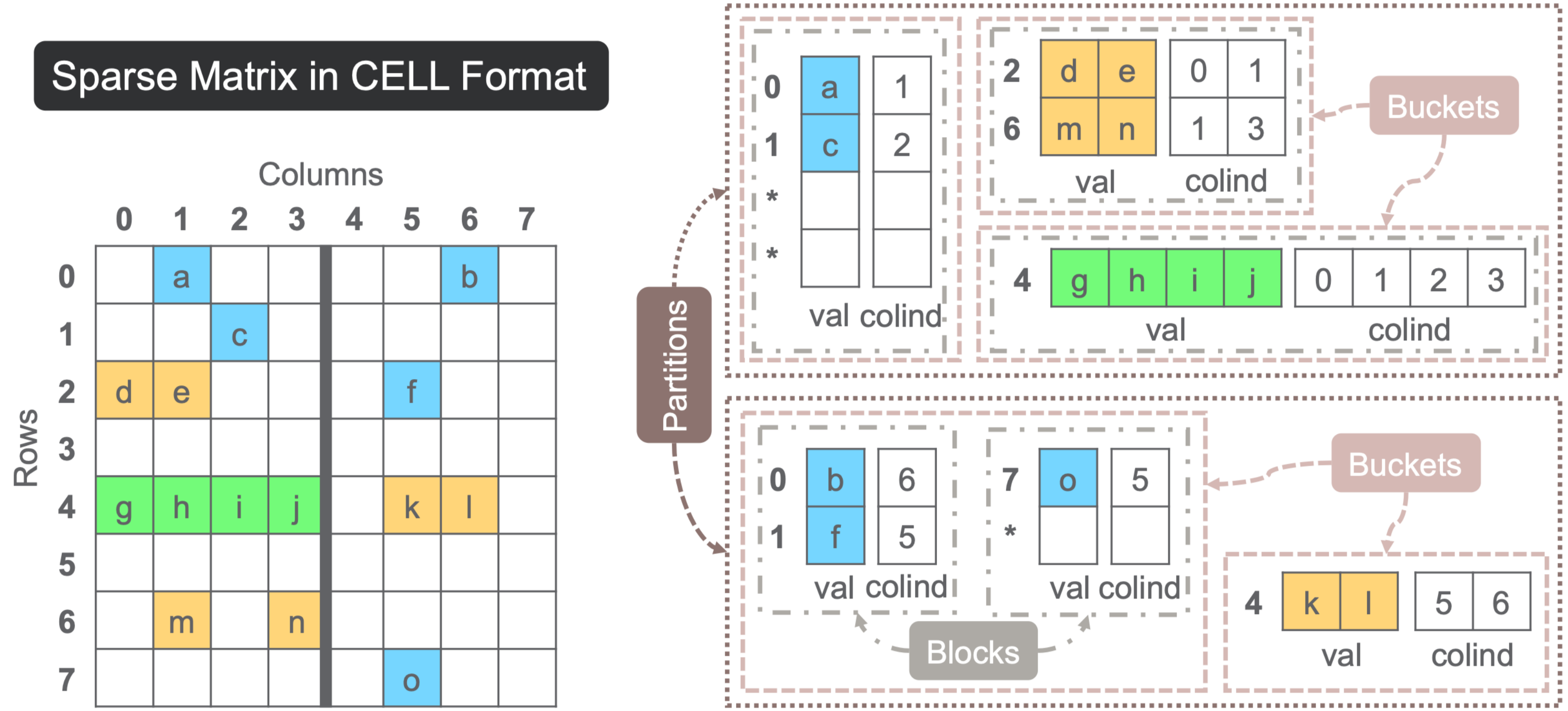
Overview of LiteForm

- Lightweight framework for automatic format composition for SpMM
- LiteForm's Composable Format
  - CELL (Composable Ellpack) format
  - 3-level blockwise: partitions, buckets, blocks
- LiteForm's Workflow
  - ML predicts if CELL > fixed formats
  - ML sets partition count
  - Cost model + search optimizes bucket widths
- Contributions
  - CELL design
  - ML predictors and cost model
  - No runtime auto-tuning



Composable Ellpack (CELL) Format Design





- 3 Levels:
  - Columns → Partitions (even divide, reduce padding)
  - Rows → Buckets (group by row length l that $2^{i-1} < l \le 2^i$)
  - Elements → Blocks (2^{k-i} rows, fixed non-zeros 2^k , map to GPU thread blocks)
- Flexible buckets per-partition
- Balancing non-zeros in blocks

Automatic Format Composition

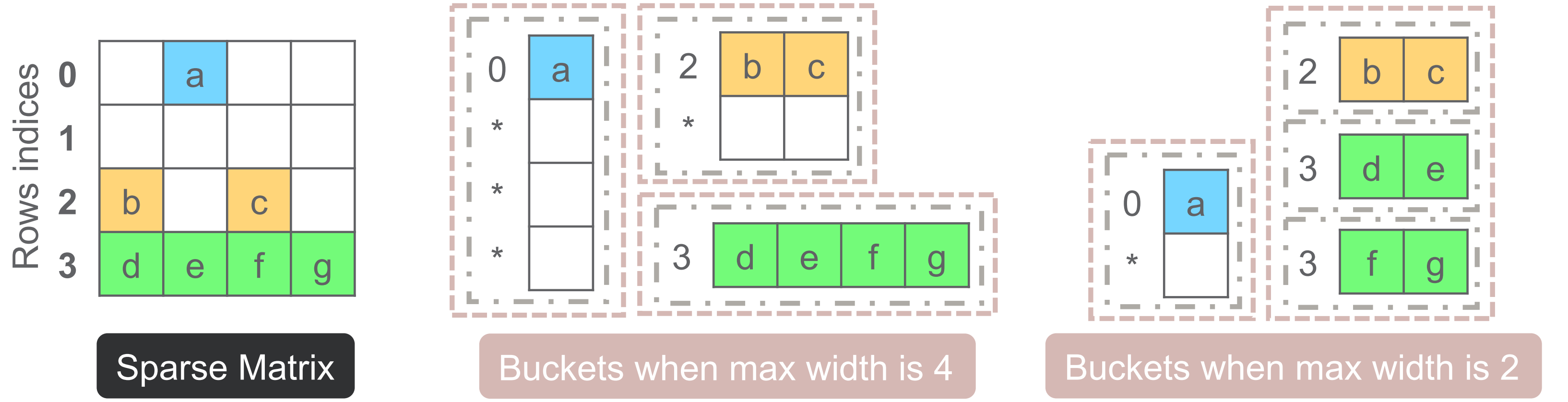
- Step 1: ML assesses CELL, predicts if >1.1x speedup over CSR and BCSR
- Step 2: ML classifier predicts number of partitions
- Step 3: Use a cost Model and search for bucket widths (see in the next slide)
- Training: run SpMM on formats to collect best execution time and configuration.
  - Overhead is amortized over future uses
  - Use Random Forest after evaluation

Sparse matrix features to predict format
Number of rows
Number of columns
Number of non-zero elements
Average number of non-zeros per row
Minimum number of non-zeros per row
Maximum number of non-zeros per row
Standard deviation of non-zeros per row

Sparse matrix features to predict number of partitions
Number of rows
Number of columns
Number of non-zero elements
Average density of non-zeros per row
Minimum density of non-zeros per row
Maximum density of non-zeros per row
Standard deviation of non-zeros density per row
Product of other dimensions in the kernel

Optimizing Bucket Widths

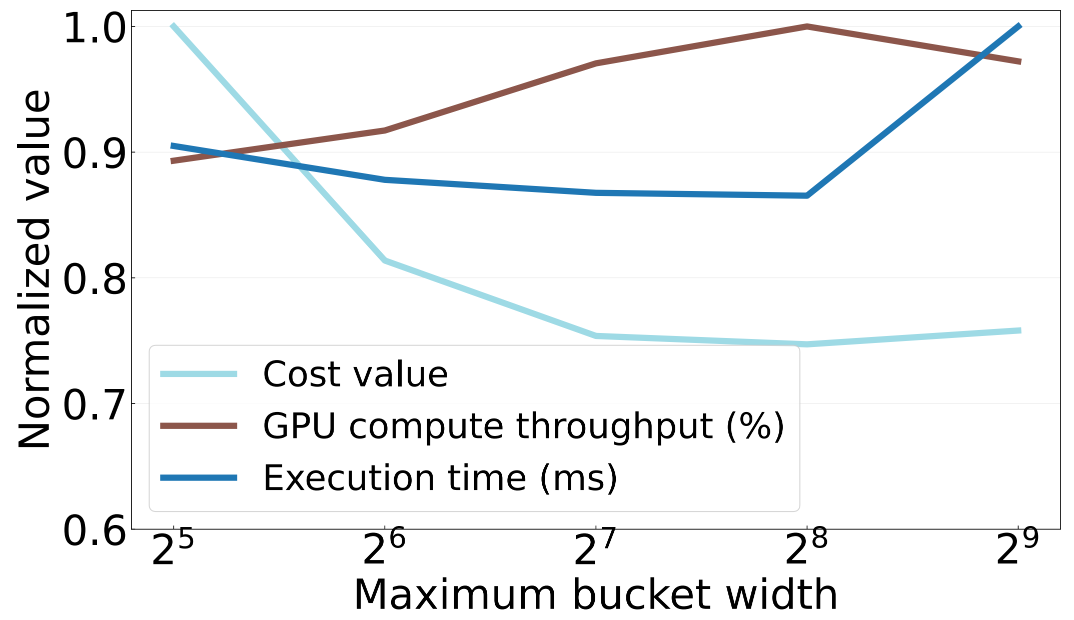
- The max bucket width trade-off:
  - Larger widths → 👍 fewer row index accesses, coarser workloads, 🚨 more zero padding
  - Smaller widths → 👍 less zero padding, 🚨 more index accesses, more atomic writing



- Memory-centric cost model

$$\begin{aligned} cost(x) = 2 * (rows\ in\ bucket * width) &\text{ for } A \\ + unique\ columns * J &\text{ for } B \\ + Atomic * (rows\ in\ C * J) &\text{ for } C \end{aligned}$$

- Use a search algorithm to find optimal widths

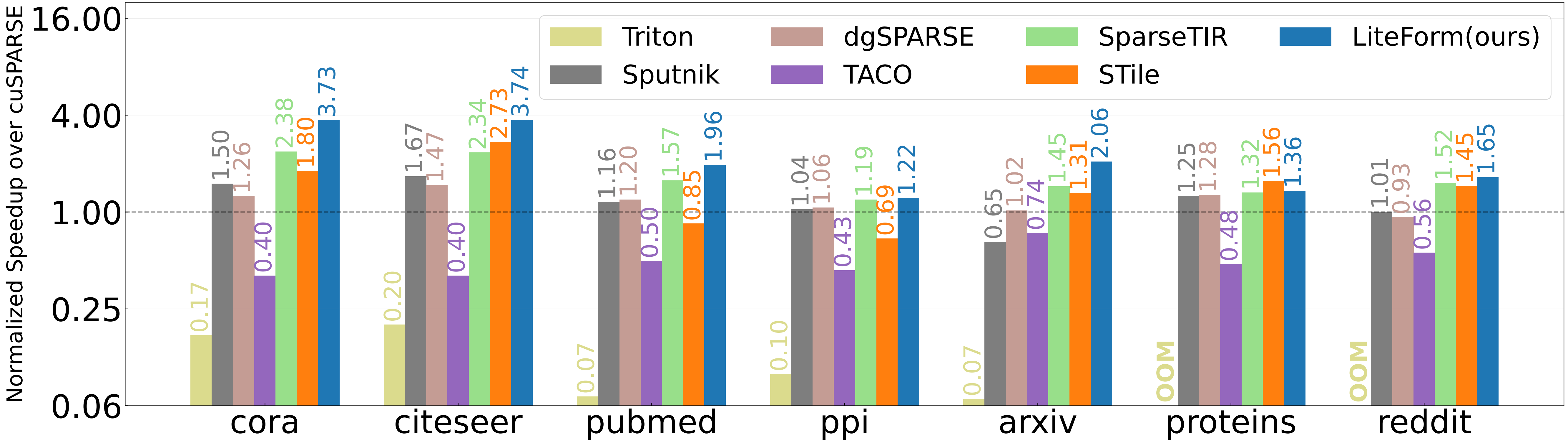


Implementation and Evaluation

- Built on SparseTIR and TVM; Use scikit-learn for ML model (Random Forest)
- Hardware: NVIDIA V100 GPU
- Baselines: cuSPARSE, Triton, Sputnik, dgSPARSE, TACO, SparseTIR, STile
- Datasets: 7 GNN graphs + 1,351 SuiteSparse matrices
- Metrics: Speedup, overhead

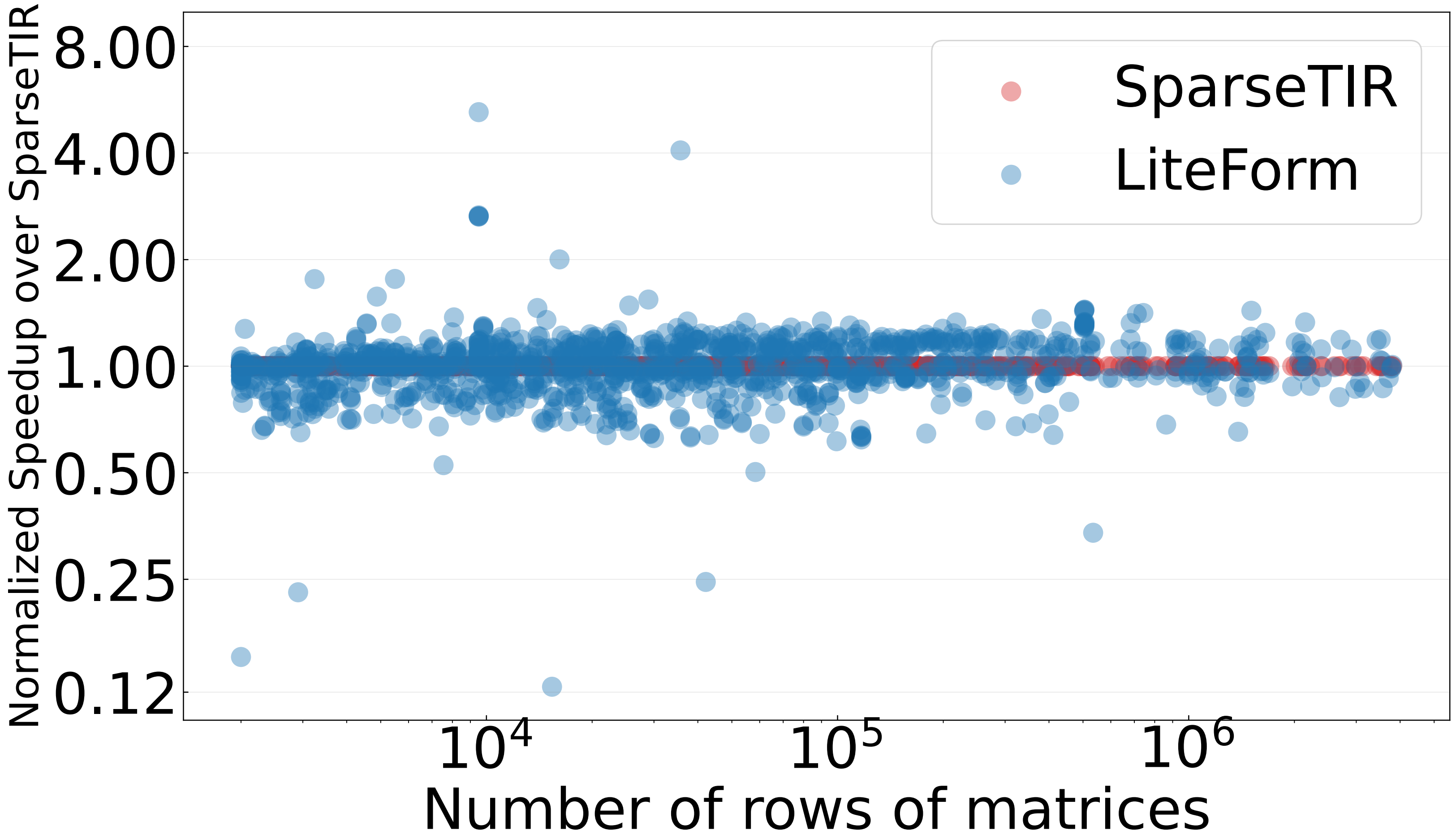
Graph	#nodes	#edges	Density
cora	2,708	10,556	1.44E-03
citeseer	3,327	9,228	8.34E-04
pubmed	19,717	88,651	2.28E-04
ppi	44,906	1,271,274	6.30E-04
arxiv	169,343	1,166,243	4.07E-05
proteins	132,534	39,561,252	2.25E-03
reddit	232,965	114,615,892	2.11E-03
SuiteSparse	2.0K–3.8M	3.1K–300.9M	8.7E-07–0.1

Performance Evaluation (GNN Graphs)



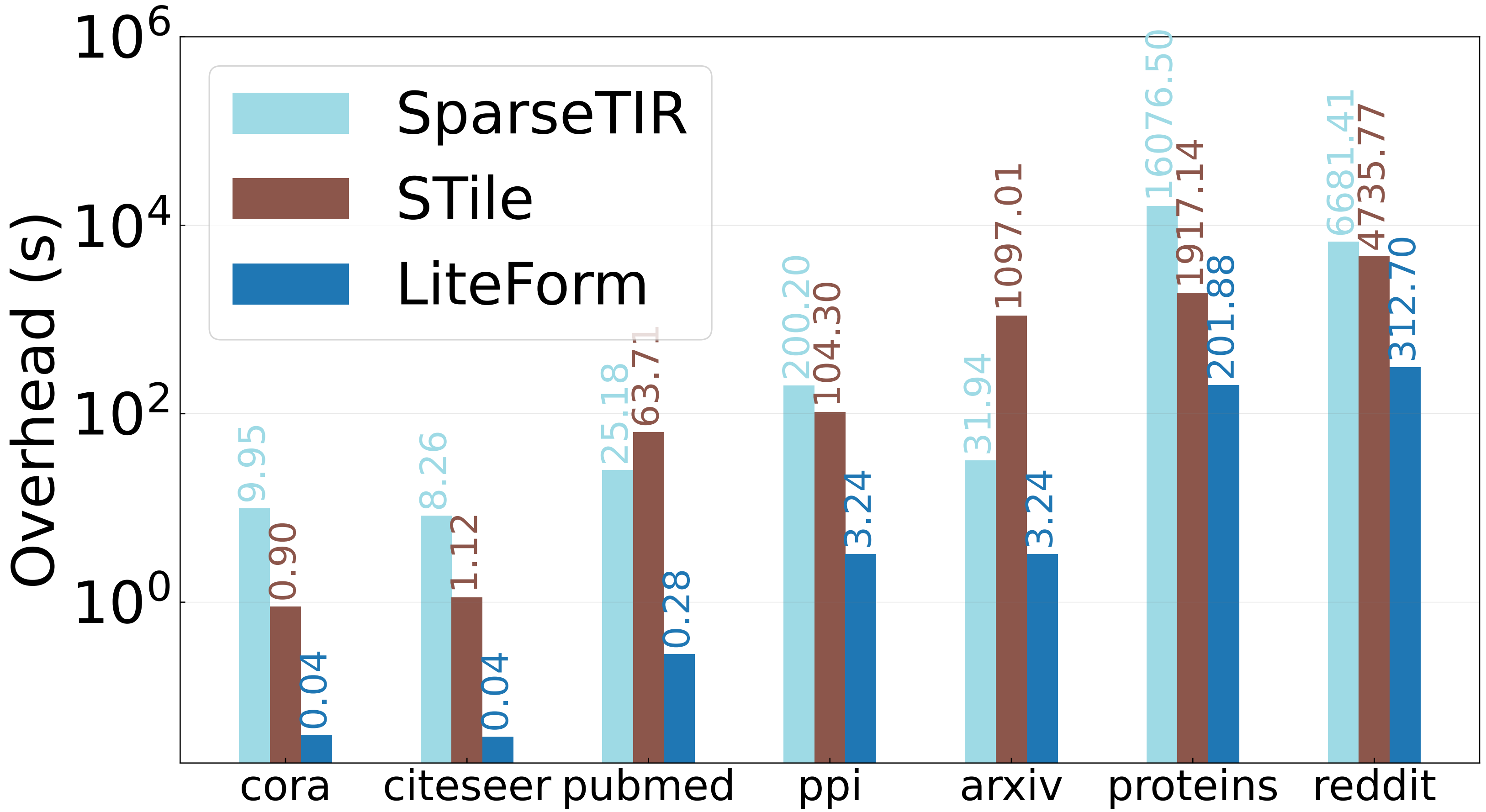
- LiteForm achieved 2.06X geometric mean speedup over cuSPARSE, 1.26X over SparseTIR, and 1.52X over STile

Performance Evaluation (SuiteSparse Graphs)



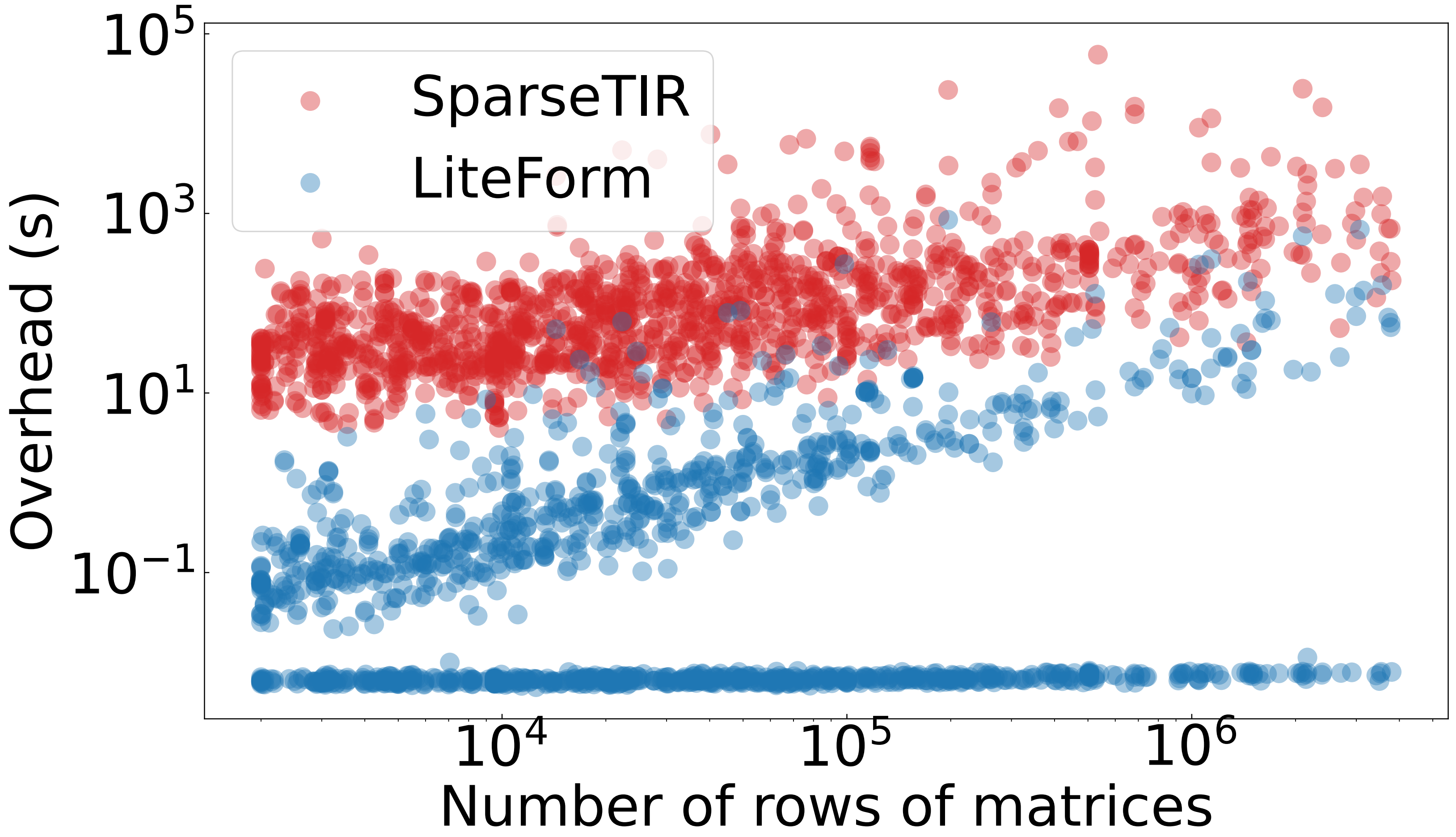
- LiteForm achieved 0.99X geometric mean speedup over SparseTIR
- SparseTIR used auto-tuning to determine the optimal configuration

Overhead Evaluation (GNN Graphs)



- SparseTIR: auto-tuning
- STile: microbenchmarking
- LiteForm: inference and searching (not including training)
- SparseTIR has 65.5X geometric mean overhead, STile has 42.3X

Overhead Evaluation (SuiteSparse Graphs)



- SparseTIR has 1150.2X geometric mean overhead

Limitations of LiteForm

- Needs collecting historical performance data, and requires model retraining for new architectures and kernels
- Historical data may not cover extreme cases, such as a large number of partitions, and extremely wide buckets
- Has not utilized Tensor Cores, or multiple GPUs

Conclusion

- LiteForm: a lightweight and automatic format composition framework for SpMM on GPUs
- Propose the Composable Ellpack (CELL) format with a 3-level blockwise design
- Utilize ML models and a cost model for automatic composition
- Eliminate the need for runtime auto-tuning

Thank you!

Zhen Peng
zhen.peng@pnnl.gov

- Backup slides

Cost Model

- For $C_{ij} = A_{ik} \cdot B_{kj}$, the cost of bucket is measured as memory accesses to matrix A, B, and C (dimensions I, J, K)
- $Cost(x) = cost^{(1)}(x) + cost^{(2)}(x) + cost^{(3)}(x)$

$$= 2 \cdot I^{(1)}W + |set(Ind[i,w])|J + Atomic \cdot I^{(2)}J$$

  - $I^{(1)}$: the number of rows in the bucket of matrix A
  - $I^{(2)}$: the number of corresponding rows in C. $I^{(1)}$ can $> I^{(2)}$ because of folded rows in A
  - W: the bucket width
  - $set(Ind[i,w])$: the set of unique column indices
  - $Atomic$: weight of atomic operation, can be set as $Atomic = \frac{I^{(1)}}{I^{(2)}}$
- Bucket width ↗ (larger W) $\rightarrow cost^{(2)}$ ↗ ($|set(Ind[i,w])|$ larger), but $cost^{(3)}(x)$ ↘ ($Atomic \cdot I^{(2)}$ and $I^{(1)}$ smaller)
- Bucket width ↘ (smaller W) $\rightarrow cost^{(2)}$ ↘ ($|set(Ind[i,w])|$ smaller), but $cost^{(3)}(x)$ ↗ ($Atomic \cdot I^{(2)}$ and $I^{(1)}$ larger)

Prediction Evaluation (Predict CELL)

Table 5: Overhead and accuracy of the tested ML models for predicting performance improvement of CELL format.

name	training(s)	inference(s)	accuracy	precision	recall	f1
Random Forest	0.2859	0.0079	88.92%	88.92%	88.92%	88.92%
KNeighbors	0.0024	0.0127	79.31%	79.31%	79.31%	79.31%
Linear SVM	0.0849	0.0098	67.00%	67.00%	67.00%	67.00%
RBF SVM	0.0856	0.0199	73.40%	73.40%	73.40%	73.40%
Gaussian Process	346.2509	0.0697	84.24%	84.24%	84.24%	84.24%
Decision Tree	0.0292	0.0004	85.96%	85.96%	85.96%	85.96%
Neural Net	2.8343	0.0016	66.50%	66.50%	66.50%	66.50%
AdaBoost	0.1828	0.0079	86.45%	86.45%	86.45%	86.45%
Naive Bayes	0.0018	0.0004	63.30%	63.30%	63.30%	63.30%
QDA	0.0022	0.0004	66.75%	66.75%	66.75%	66.75%

- Used 80% of 514 matrices as the training set and the other 20% as the test set
- Random Forest achieved the best accuracy

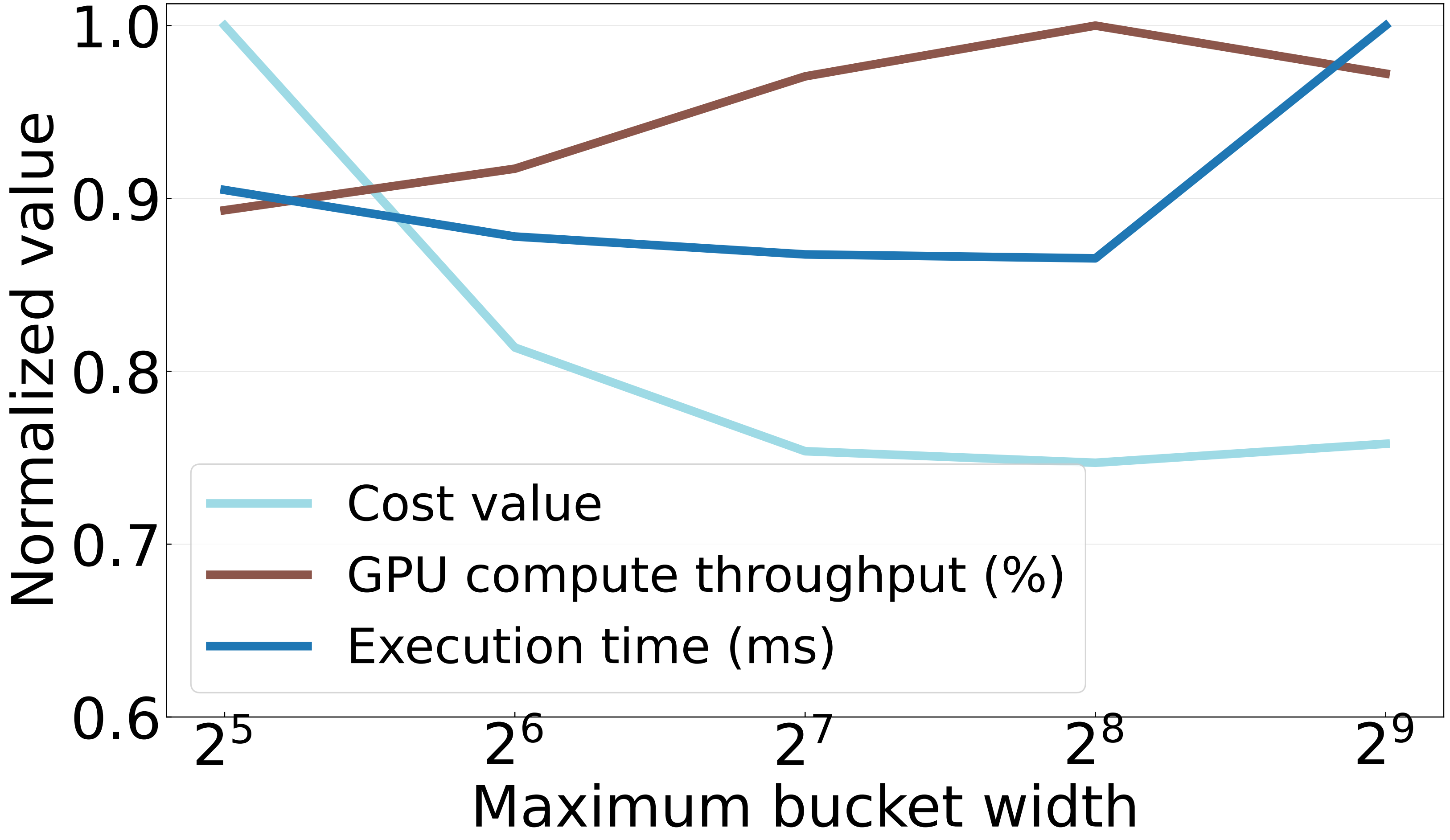
Prediction Evaluation (Predict Num. of Partitions)

Table 6: Overhead and accuracy of the tested ML models for predicting the optimal number of partitions in the CELL format. *cos_sim* stands for cosine similarity.

name	training(s)	inference(s)	accuracy	precision	recall	f1	cos_sim
Random Forest	0.4778	0.0127	87.30%	87.30%	87.30%	87.30%	0.77
KNeighbors	0.0046	0.0321	82.98%	82.98%	82.98%	82.98%	0.23
Linear SVM	0.2273	0.0244	82.45%	82.45%	82.45%	82.45%	0.25
RBF SVM	0.5688	0.0692	82.56%	82.56%	82.56%	82.56%	0.25
Gaussian Process	1481.1395	24.0115	82.56%	82.56%	82.56%	82.56%	0.25
Decision Tree	0.0200	0.0005	85.40%	85.40%	85.40%	85.40%	0.77
Neural Net	3.0432	0.0017	82.45%	82.45%	82.45%	82.45%	0.25
AdaBoost	0.1952	0.0106	82.13%	82.13%	82.13%	82.13%	0.25
Naive Bayes	0.0025	0.0008	56.41%	56.41%	56.41%	56.41%	0.29
QDA	0.0036	0.0011	0.21%	0.21%	0.21%	0.21%	0.25

- Random Forest achieved the best accuracy

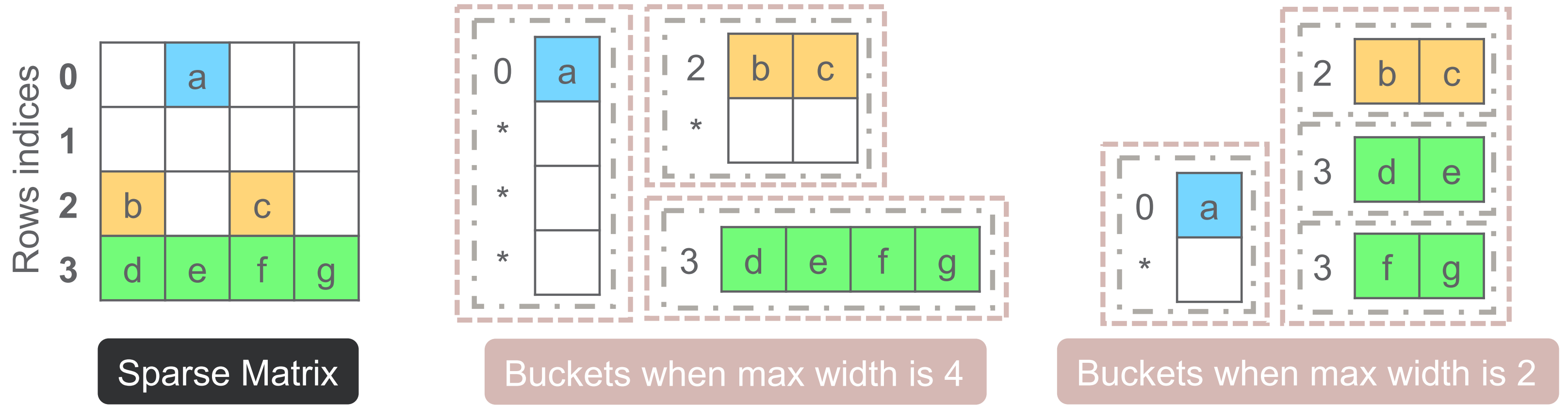
Cost Model Evaluation



- Tested the reddit data set
- The bucket width influence the cost value from the cost model
- When the cost value is the lowest, the GPU compute throughput reaches the highest and the execution time is the shortest

Optimizing Bucket Widths

- CELL format can represent a single long row as multiple rows
- The number of non-zeros in a block is set by the maximum bucket width



A larger maximum bucket width → fewer row index accesses, but more padding
A smaller maximum bucket width → more row index accesses, but fewer padding

- Use a cost model to estimate memory access overhead for given widths
  - Larger bucket width → More overhead to access B, but less overhead to access C
  - Smaller bucket width → Less overhead to access B, but larger overhead to access C
- Use a search algorithm to find optimal widths