From Sparse Patterns to Smart Acceleration: Machine Learning Methods for Next-Generation Accelerators

Bahar Asgari

11/20/25

Future of Computing?

More DSA papers from academia, more investments from industry

d-Matrix

cerebras

GRAPHCORE

groq™

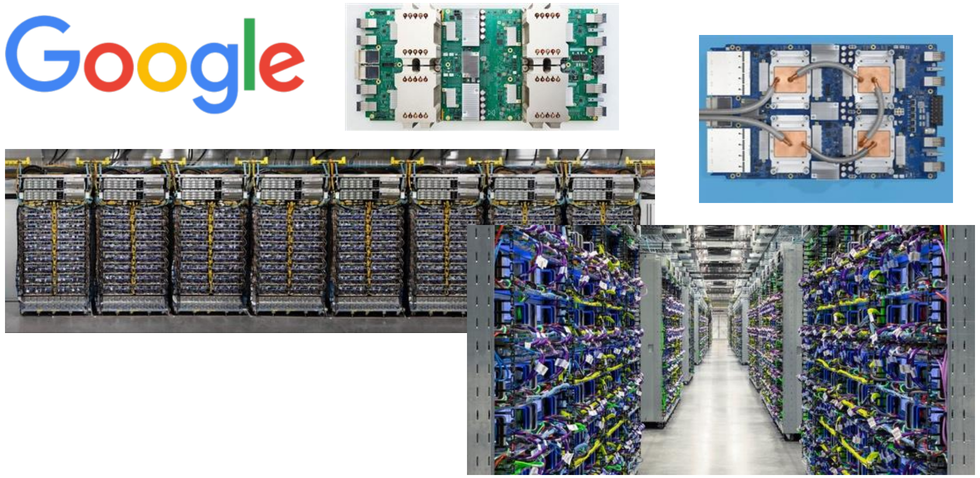
LIGHTMATTER

SambaNova® SYSTEMS

tenstorrent

NVIDIA

amazon

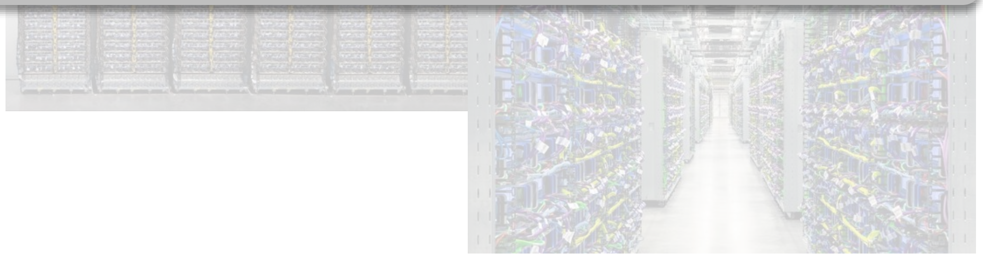


Future of Computing?

More DSA papers from academia, more investments from industry

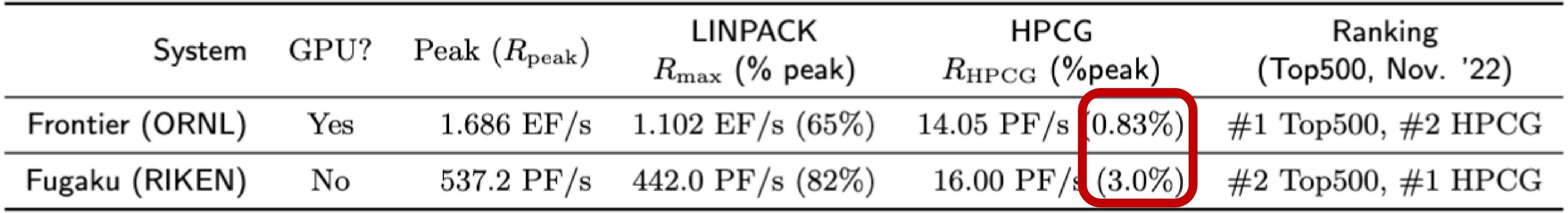
Are DSAs efficient and effective?

Can DSAs be the future of computing?



How about when problems are <u>sparse</u>?

HPCG benchmark, which is dominated by distributed sparse matrix-vector multiplication



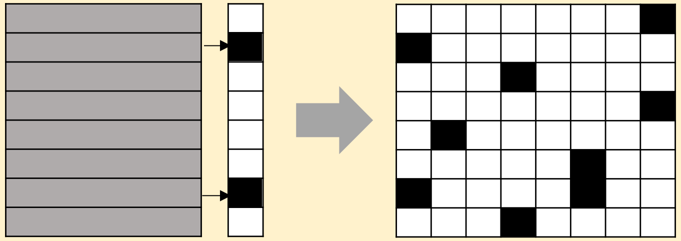
System	GPU?	Peak (R_{peak})	LINPACK $R_{\max}$ (% peak)	HPCG R_{HPCG} (%peak)	Ranking (Top500, Nov. '22)
Frontier (ORNL)	Yes	1.686 EF/s	1.102 EF/s (65%)	14.05 PF/s (0.83%)	#1 Top500, #2 HPCG
Fugaku (RIKEN)	No	537.2 PF/s	442.0 PF/s (82%)	16.00 PF/s (3.0%)	#2 Top500, #1 HPCG

We don't want DSAs to face the same issue!

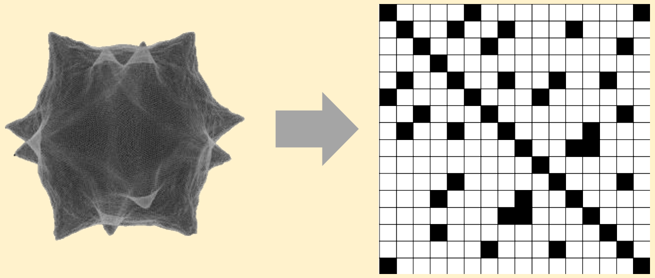
See top500.org lists for more recent lists!

Sparsity dominates,
we should design for **sparsity**

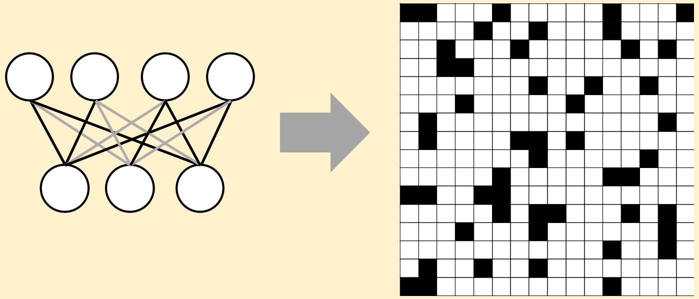
DLRMs



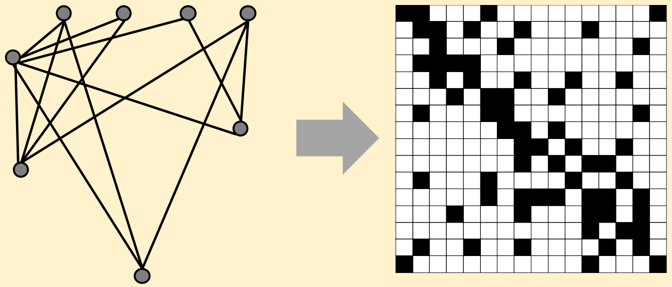
Scientific Computing



DNNs, LLMs, …



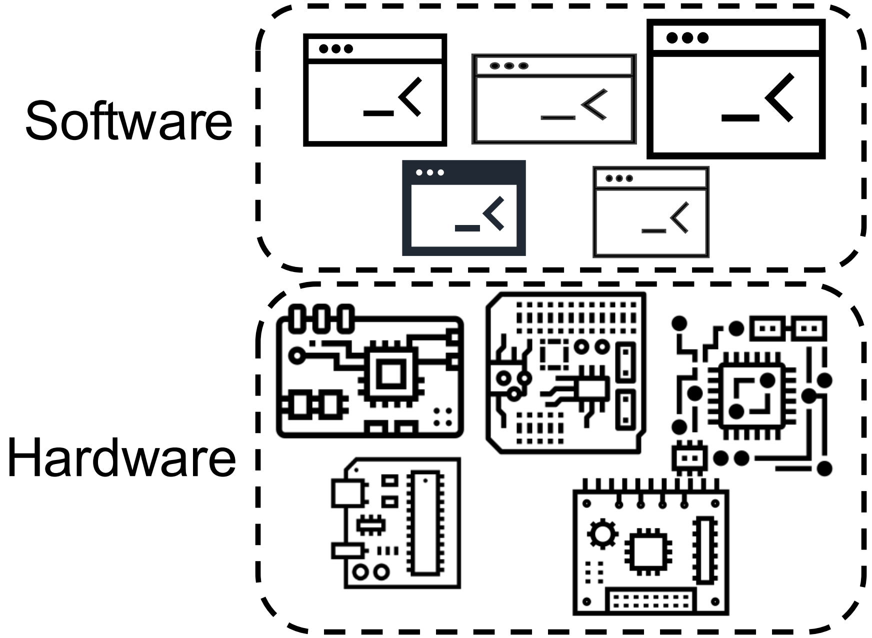
Graph Analytics



Sparsity dominates, we design for **sparsity**

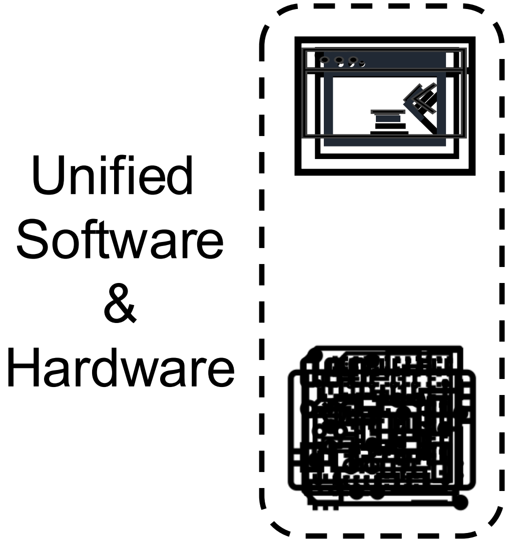
Programs are diverse, we design for **diversity**

Costly
Slow Development
Not Scalable



Our Mission:

Cost Efficient
Quick Development
Scalable



Research Questions

- How to design for sparsity? What needs to be optimized?
- How to make decisions based on sparsity?
- Where [in DSAs] a sparsity-aware decision can help?

Research Question 1:

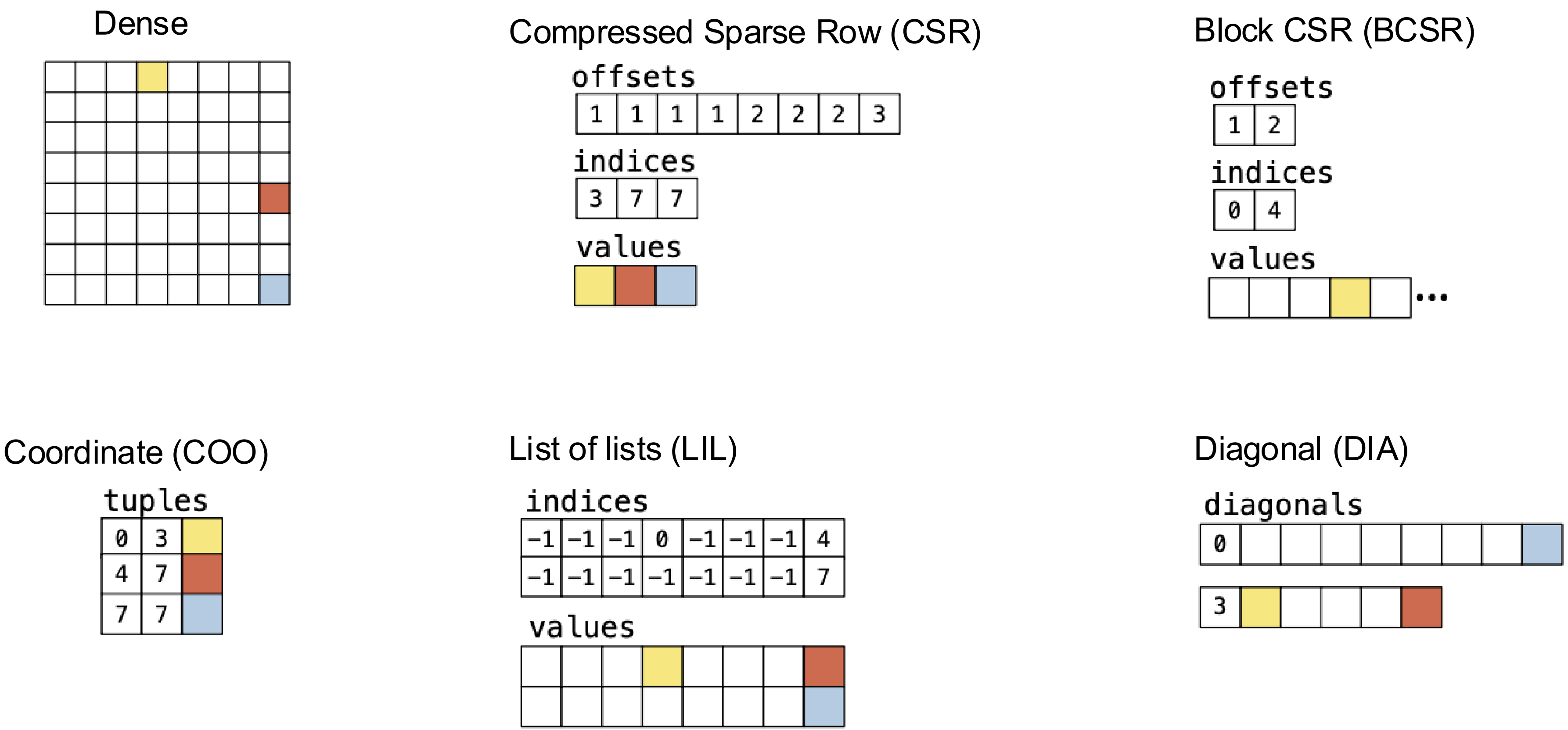
How to design for sparsity? What needs to be optimized?

Sparsity and compression

When data is sparse, we often compress it to eliminate

- Unnecessary data movement
- Unnecessary data storage

What compression format to use? How to design other components based on it?



Dense

Compressed Sparse Row (CSR)

offsets

1	1	1	1	2	2	2	3

indices

3	7	7

values

🟨	🟥	🟦

Block CSR (BCSR)

offsets

1	2

indices

0	4

values

			🟨	

…

Coordinate (COO)

tuples

0	3	🟨
4	7	🟥
7	7	🟦

List of lists (LIL)

indices

-1	-1	-1	0	-1	-1	-1	4
-1	-1	-1	-1	-1	-1	-1	7

values

			🟨				🟥
							🟦

Diagonal (DIA)

diagonals

0								🟦
3	🟨				🟥			

And several other formats: Bitmap, CSC, BCSC, ELLpack (ELL) and sliced ELL (SELL), Directory of Keys (DoK)

Source of figures: COPERNICUS, Asgari, et al. IISWC'21

CORUSCANT: Co-Designing GPU Kernel and Sparse Tensor Core to Advocate Unstructured Sparsity in Efficient LLM Inference

- Led by Donghyeon Joo, a 2nd year PhD student at CASL
- Fresh from the oven! **MICRO 2025**
- **Authors:** Donghyeon Joo, Helia Hosseini, Ramyad Hadidi, and Bahar Asgari
- Please check Donghyeon's **NeurIPS 2025** paper as well:
  - **Mustafar: Promoting Unstructured Sparsity for KV Cache Pruning in LLM Inference**

LLMs Face Critical Memory Constraints

The Challenge:

- LLMs are getting massive (GPT-3: 175B, Llama-2: 70B parameters)
- Memory is the bottleneck for deployment
- Decode phase dominates runtime (>80% of total latency)

Model Pruning Promise:

- Can reduce memory footprint by 30-70%
- Unstructured pruning preserves accuracy better than structured

BUT: Current hardware can't exploit unstructured sparsity efficiently

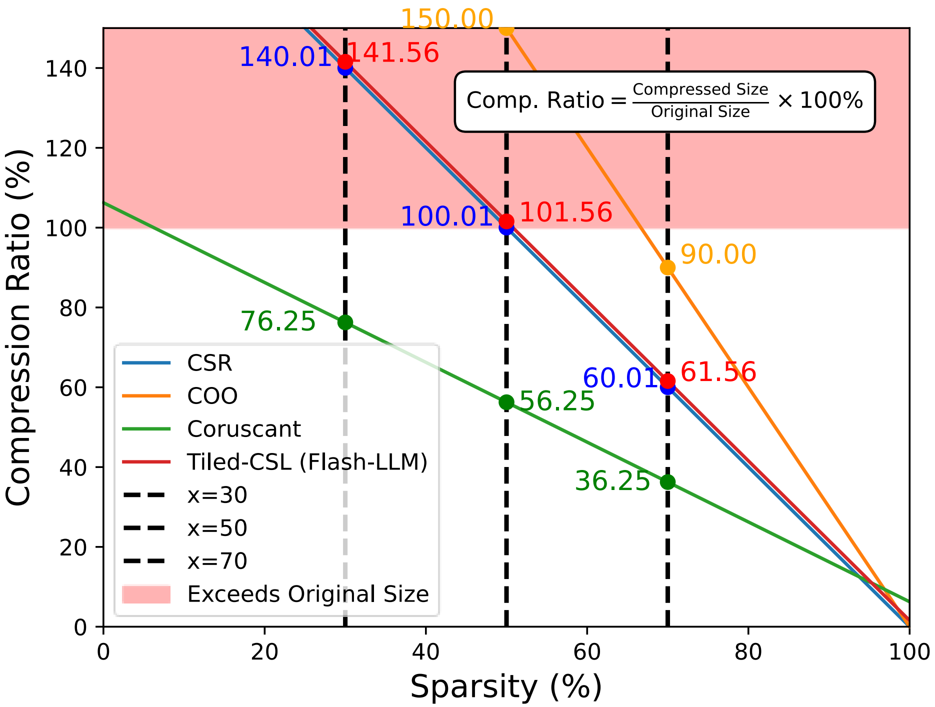
Why Unstructured Sparsity is Ignored Current State?

- GPUs only support structured sparsity
- Forces pruning methods to adopt this restrictive pattern
- Results in significant accuracy degradation

The Problem with Current Solutions

1. Existing sparse formats are not efficient at 30-70% sparsity (our target range)
2. No GPU support for other sparsity patterns

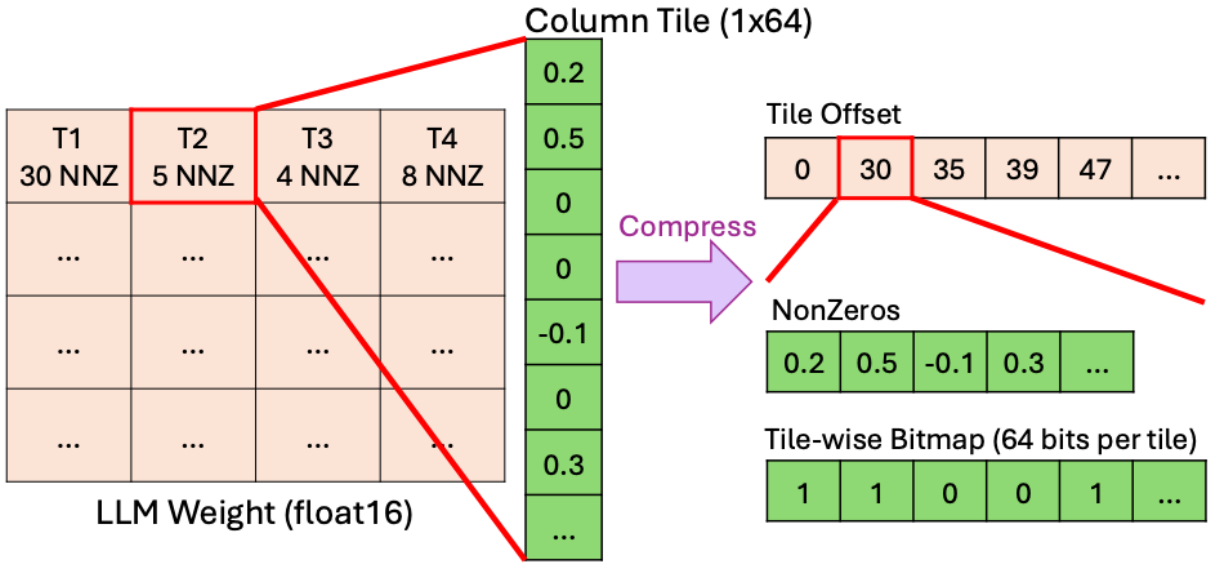
The figure shows the compression ratio comparison showing the inefficiency of existing formats



Our Solution: Let’s start from compression format

1. Coruscant Sparse Format

- Bitmap-based representation
- Efficient for 30-70% range



T1 30 NNZ	T2 5 NNZ	T3 4 NNZ	T4 8 NNZ
...	...	...	...
...	...	...	...
...	...	...	...

LLM Weight (float16)

Column Tile (1x64)

0.2
0.5
0
0
-0.1
0
0.3
...

Compress

Tile Offset

0	30	35	39	47	...

NonZeros

0.2	0.5	-0.1	0.3	...

Tile-wise Bitmap (64 bits per tile)

1	1	0	0	1	...

Our Solution: What else can be co-designed with compression format?

2. Coruscant GPU Kernel

- Reduces data transfer
- Column-wise tiling to avoid bank conflicts

3. Coruscant Sparse Tensor Core

- Operates directly on compressed format
- Hardware support for bitmap format

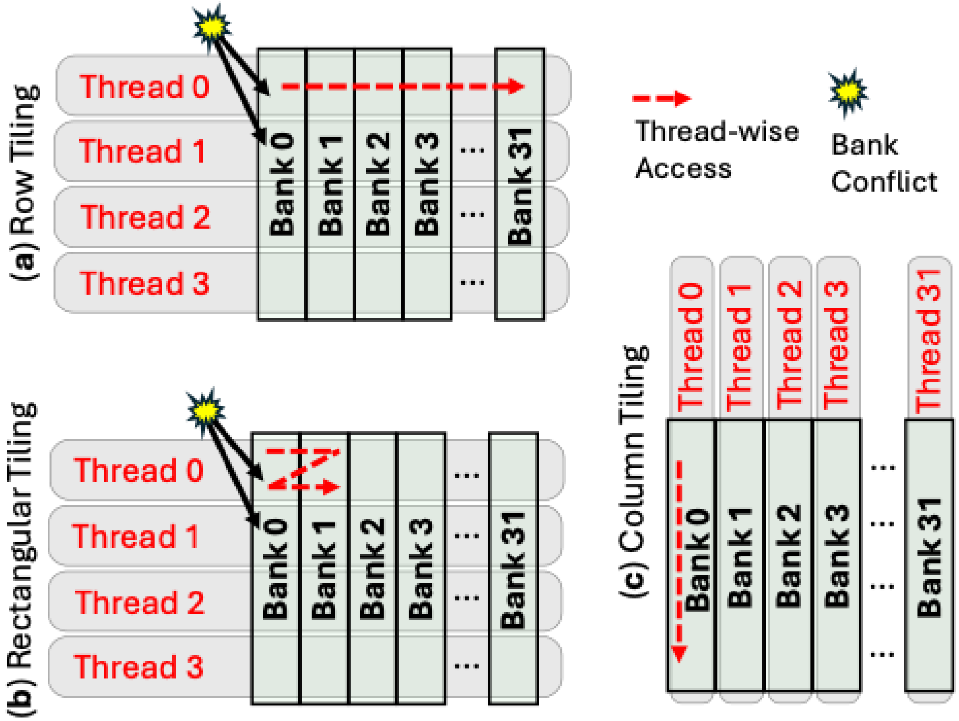
What can be optimized in a GPU kernel?

Smart Memory Management:

- Load-as-sparse, compute-as-dense
- Column-wise tiling prevents bank conflicts
- Bitmap decompression in registers

Why Column-wise Tiling?

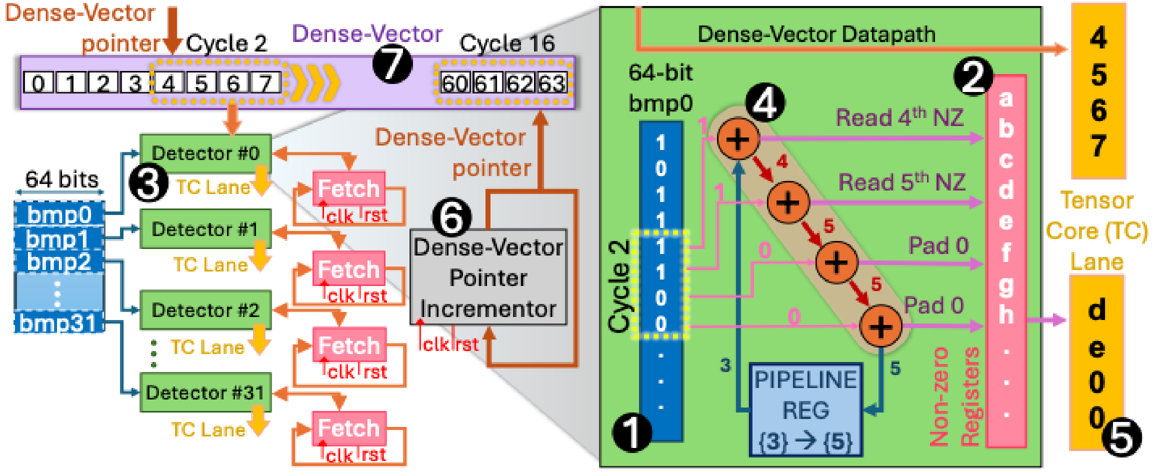
- Each thread writes to different memory banks
- Zero bank conflicts during decompression
- Critical for performance



Can we improve it further?
Yes, by customizing hardware!

The Bitmap Decoder (show in figure):

- Parses bitmap without decompression
- Minimal hardware overhead



Benefits:

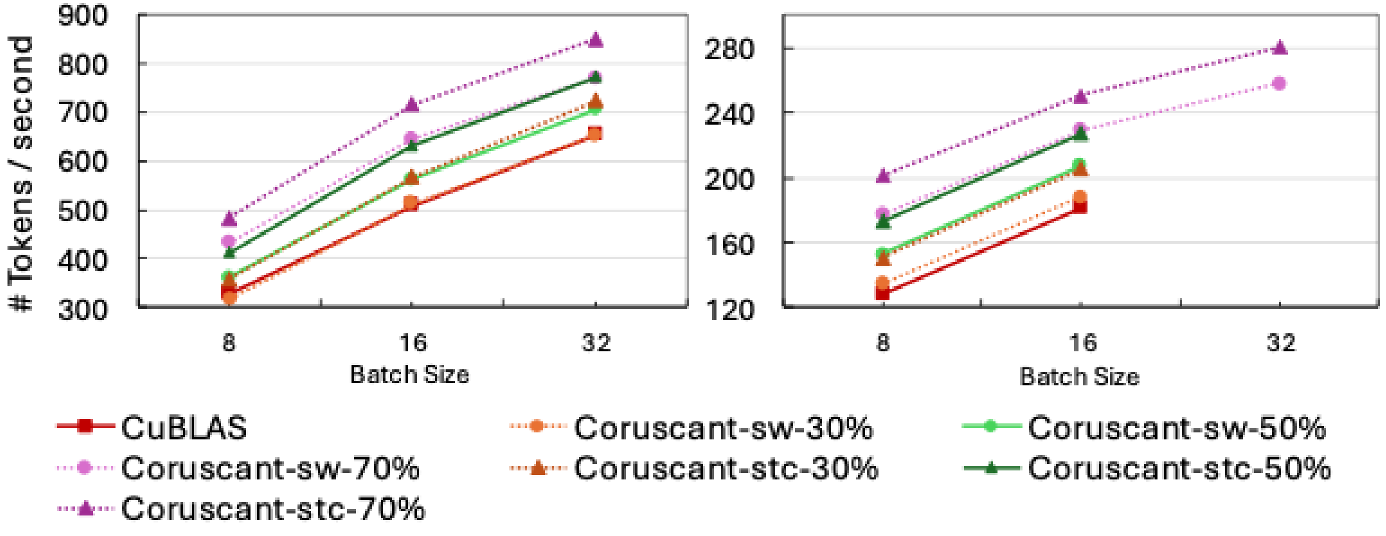
- Removes decompression overhead
- Enables true sparse computation

Speedups Across the Board Kernel Performance:

- Up to 2.02× at 70% sparsity vs cuBLAS
- Up to 1.48× at 50% sparsity vs Flash-LLM
- Up to 2.75× total With Sparse Tensor Core (STC) vs cuBLAS

End-to-End Speedup

- Software kernel: 135 tokens/sec (+26%)
- With sparse tensor core: 206 tokens/sec (+40%)



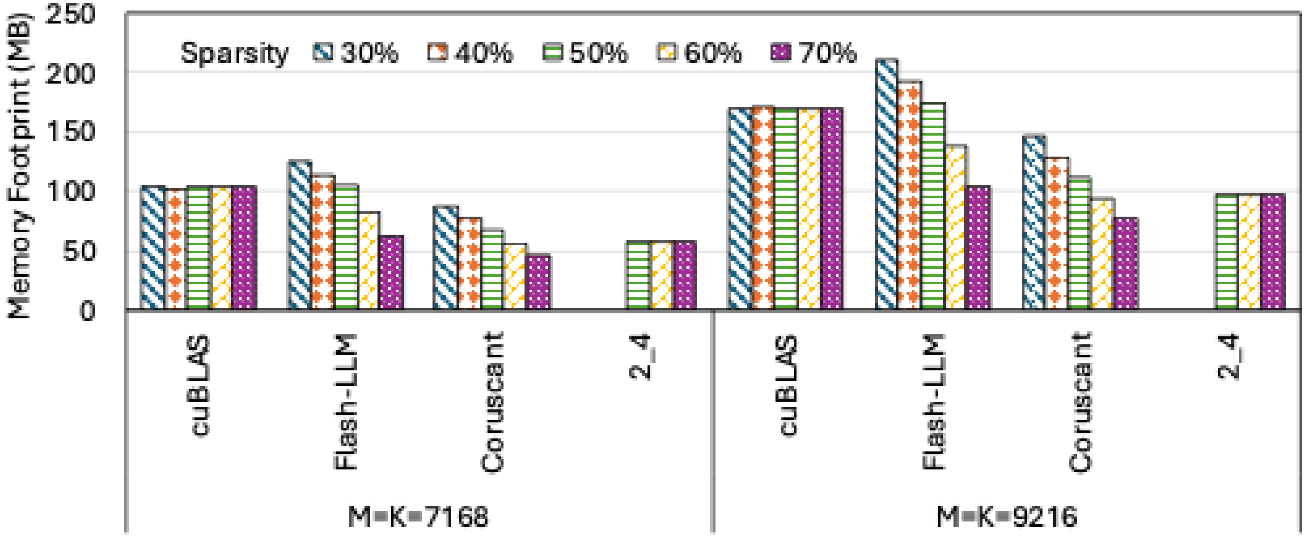
(a) Llama2 7B

(b) Llama2 13B

Beyond Just Performance

Memory Footprint Reduction:

- 55% reduction at 70% sparsity
- More space for KV cache
- Enables larger batch sizes



Energy Efficiency:

- Lower data movement = less energy
- Critical for deployment costs

Hardware Efficiency:

- Only 0.51mm^2 area overhead on V100
- 0.018% of total GPU area

Workloads are changing so quickly

- Example: Google’s TPU Workloads

DNN Model	2016	2019	2020	2022
MLP/DLRM	61%	27%	25%	24%
RNN	29%	21%	29%	2%
CNN	5%	24%	18%	12%
Transformers	0%	21%	28%	57%

Reference: Jouppi, Norm, et al. "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings." Proceedings of the 50th Annual International Symposium on Computer Architecture. 2023.

Research Question 2:

How to make decisions based on sparsity?

Misam: Machine Learning Assisted Configuration Selection in Accelerators for Sparse Matrix Multiplication

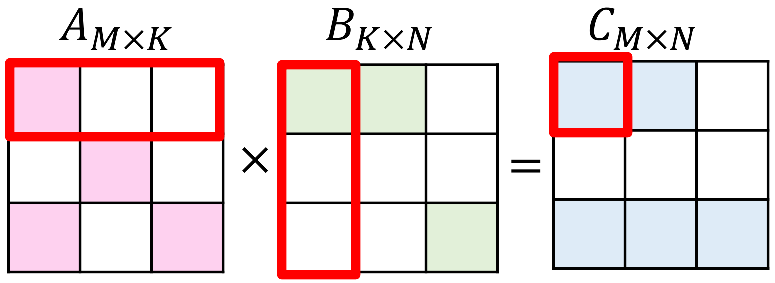
- Led by Sanjali Yadav, a 2nd year PhD student at CASL
- Also, fresh from the oven! **MICRO 2025**
- **Authors:** Sanjali Yadav, Amirmahdi Namjoo, and Bahar Asgari

SpGEMM Accelerators

Modern accelerators employ different dataflows for kernel implementation.

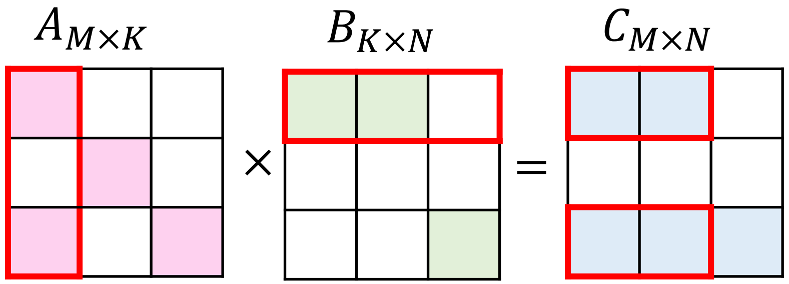
Inner Product (IP)

```
for m=0 to M
 for n=0 to N
  for k=0 to K
   C[m][n]+= A[m][k]*B[k][n]
```



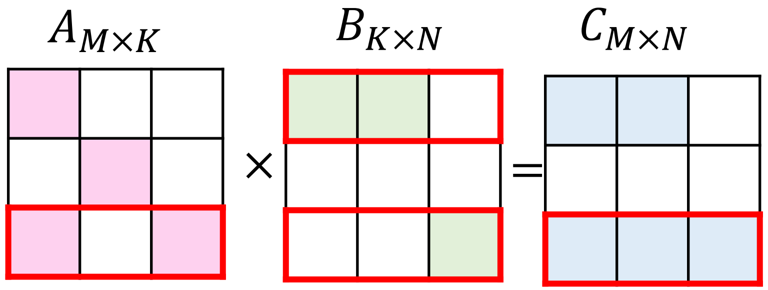
Outer Product (OP)

```
for k=0 to K
 for m=0 to M
  for n=0 to N
   C[m][n]+= A[m][k]*B[k][n]
```



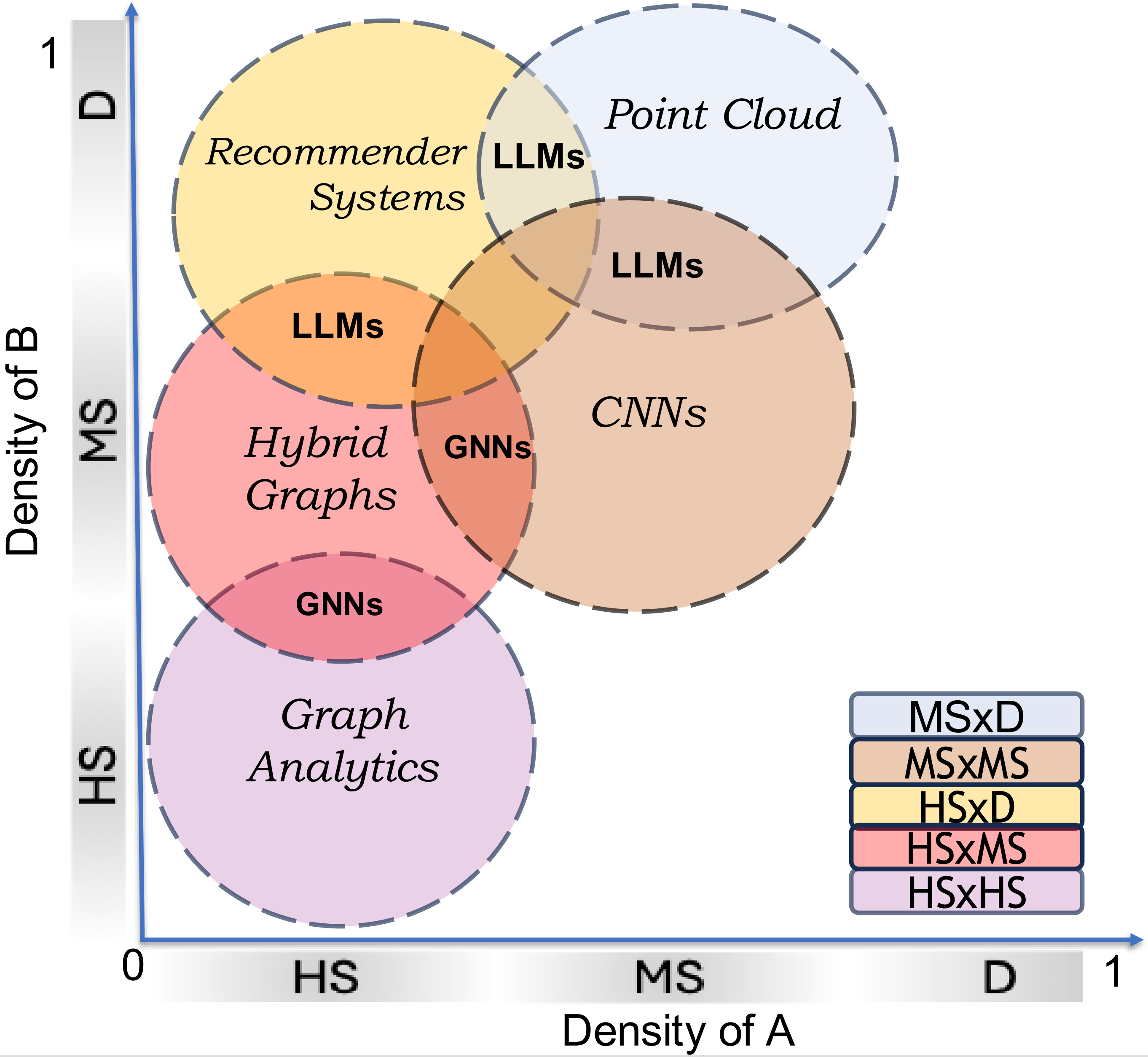
Row-Wise Product (RW)

```
for m=0 to M
 for k=0 to K
  for n=0 to N
   C[m][n]+= A[m][k]*B[k][n]
```



Sparsity…

- Refers to the presence of ineffectual zeroes in the data
- Occurs in various applications
- Unfolds across a range:
  - Highly sparse (HS)
  - Mildly sparse (MS)
  - Dense (D)

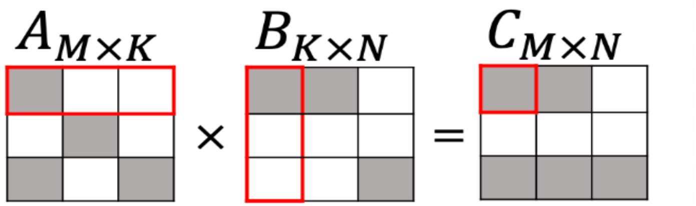


Hardware Accelerators for Sparsity

Modern sparse matrix–multiplication (SpMM/SpGEMM) accelerators handle varying sparsity by using fixed dataflow schemes*, **inner product**, **outer product**, **and row-wise product**, each tuned for a sparsity range.

**Dataflow Scheme: combination of the compression format of the sparse inputs and output and the computation schedule.*

Dataflow for Sparse Accelerators

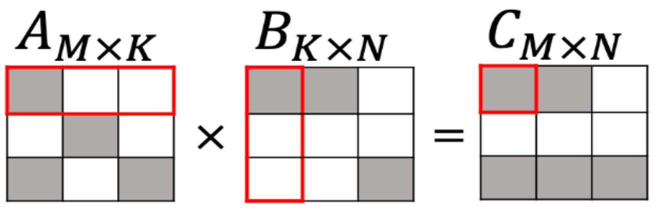


Inner Product (IP)

```
for m=0 to M
 for n=0 to N
  for k=0 to K
   C[m][n]+= A[m][k]*B[k][n]
```

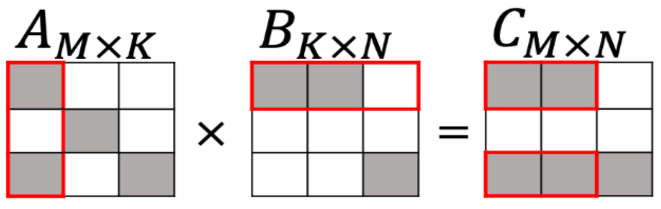
DESIGN ASPECTS	IP
Partial Sum Granularity	✓
Index Intersection	✗
Input Reuse of B	✗
Output Reuse	✓

Dataflow for Sparse Accelerators



Inner Product (IP)

```
for m=0 to M
 for n=0 to N
  for k=0 to K
   C[m][n]+= A[m][k]*B[k][n]
```

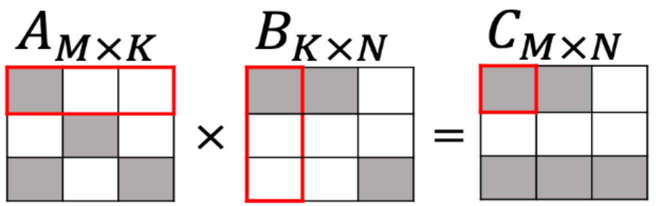


Outer Product (OP)

```
for k=0 to K
 for m=0 to M
  for n=0 to N
   C[m][n]+= A[m][k]*B[k][n]
```

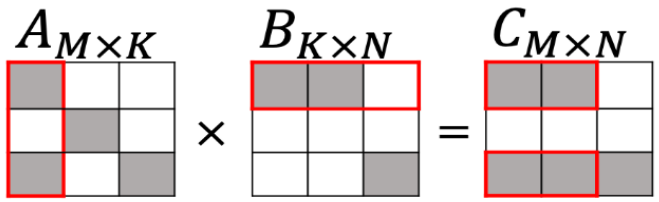
DESIGN ASPECTS	IP	OP
Partial Sum Granularity	✓	✗
Index Intersection	✗	✓
Input Reuse of B	✗	✓
Output Reuse	✓	✗

Dataflow for Sparse Accelerators



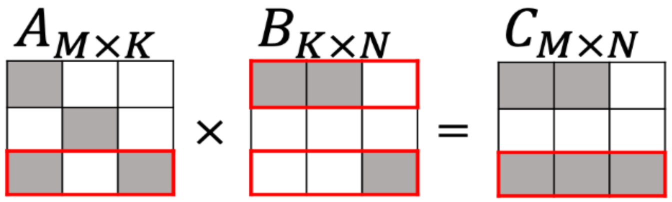
Inner Product (IP)

```
for m=0 to M
 for n=0 to N
  for k=0 to K
   C[m][n]+= A[m][k]*B[k][n]
```



Outer Product (OP)

```
for k=0 to K
 for m=0 to M
  for n=0 to N
   C[m][n]+= A[m][k]*B[k][n]
```



Row-Wise Product (RW)

```
for m=0 to M
 for k=0 to K
  for n=0 to N
   C[m][n]+= A[m][k]*B[k][n]
```

DESIGN ASPECTS	IP	OP	RW
Partial Sum Granularity	✓	☒	✓
Index Intersection	☒	✓	✓
Input Reuse of B	☒	✓	☒
Output Reuse	✓	☒	✓

Flexible Accelerators for Varying Sparsity

Recent accelerators support multiple dataflows for varied sparsity patterns. However, there are still challenges:

Trapezoid[1]: Has no built-in method to select optimal dataflow.

Flexagon[2]: Uses an offline profiling approach.

Challenge 1: Lack of a fast, accurate and generalizable dataflow selection method.

[1] Y. Yang, J. S. Emer and D. Sanchez, "Trapezoid: A Versatile Accelerator for Dense and Sparse Matrix Multiplications," *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, Buenos Aires, Argentina, 2024, pp. 931-945, doi: 10.1109/ISCA59077.2024.00072.

[2] Francisco Muñoz-Martínez, Raveesh Garg, Michael Pellauer, José L. Abellán, Manuel E. Acacio, and Tushar Krishna. 2023. 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 (ASPLOS 2023). https://doi.org/10.1145/3582016.3582069

Cost of Flexibility

A second challenge faced by these versatile accelerators is the cost of flexibility.

MORE COMPUTE UNITS

COMPLEX DISTRIBUTION NETWORK

MORE ON-CHIP STORAGE STRUCTURE

Harnessing Reconfigurability of FPGAs

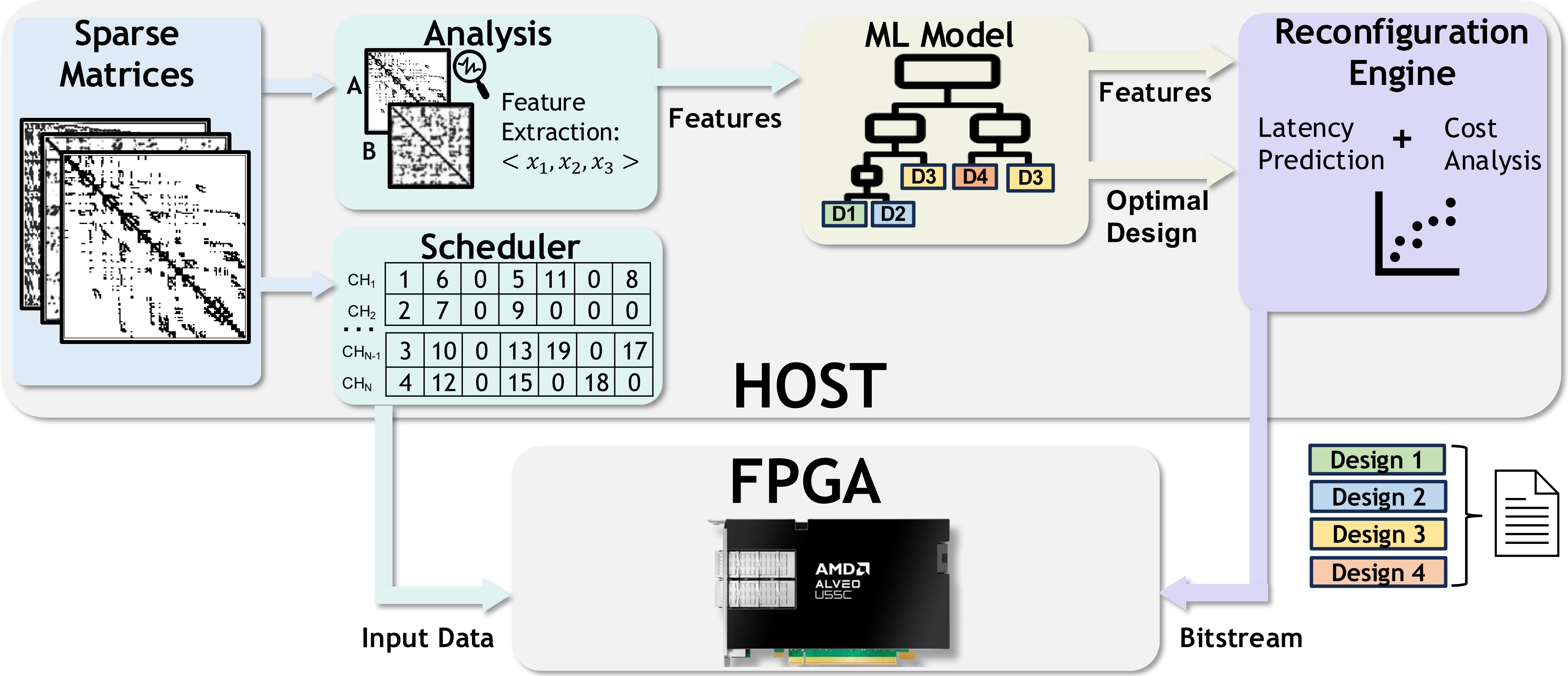
- FPGAs have emerged as popular platform for accelerating sparse linear algebra.
- FPGAs offer fine-grained parallelism and reconfigurable fabric for deeply pipelined, custom datapaths.

Challenge 2: Harnessing reconfigurability of FPGAs to create a generalizable, sparsity-aware accelerator remains a largely untapped opportunity.

Misam solves these challenges!

- Formulates dataflow scheme selection as an ML classification problem
- Develops a reconfiguration engine.
- Introduces a set of FPGA-based dataflow scheme implementations for SpMM and SpGEMM

Overview of Misam



Sparse Matrices

Analysis

Feature Extraction: $< x_1, x_2, x_3 >$

Features

ML Model

Features

Optimal Design

Reconfiguration Engine

Latency Prediction + Cost Analysis

Scheduler

CH_1	1	6	0	5	11	0	8
CH_2	2	7	0	9	0	0	0
⋯							
CH_{N-1}	3	10	0	13	19	0	17
CH_N	4	12	0	15	0	18	0

HOST

FPGA

Input Data

Bitstream

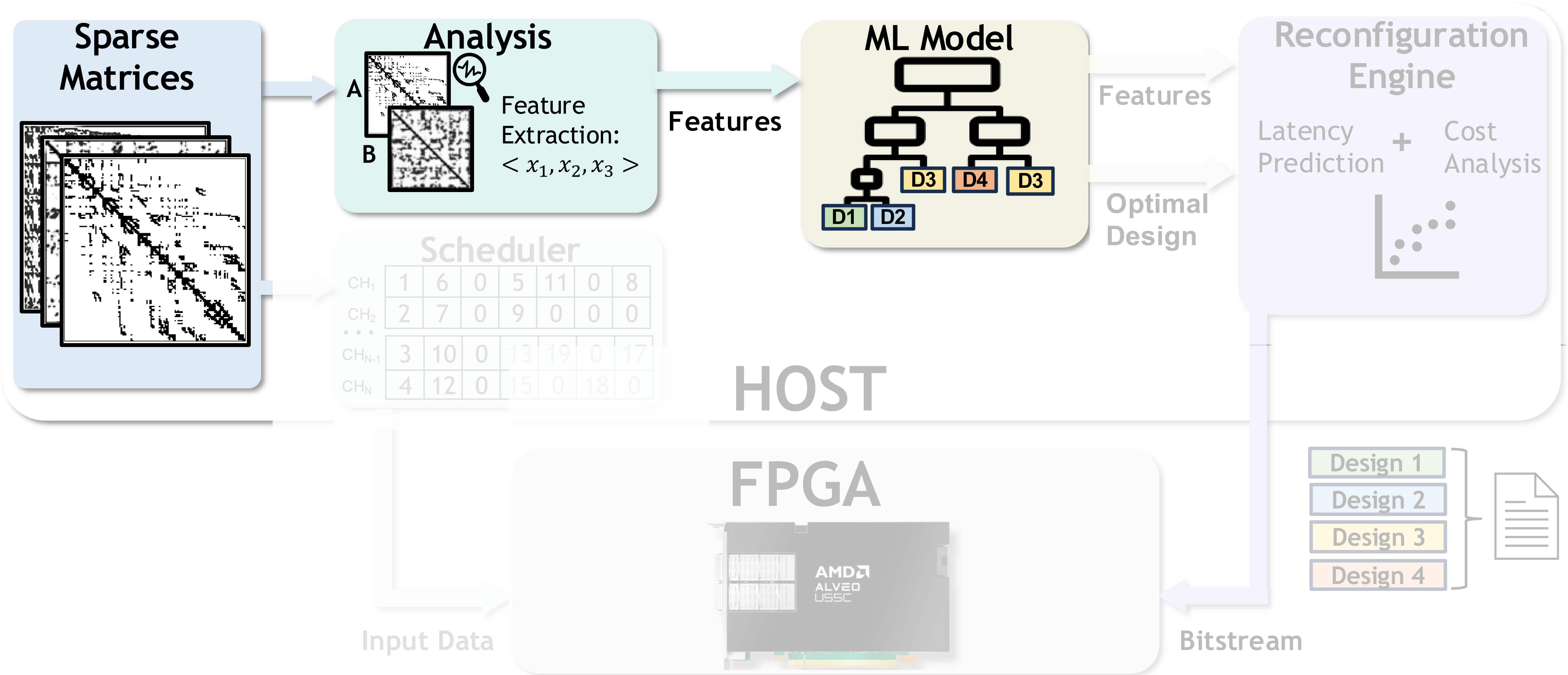
Design 1

Design 2

Design 3

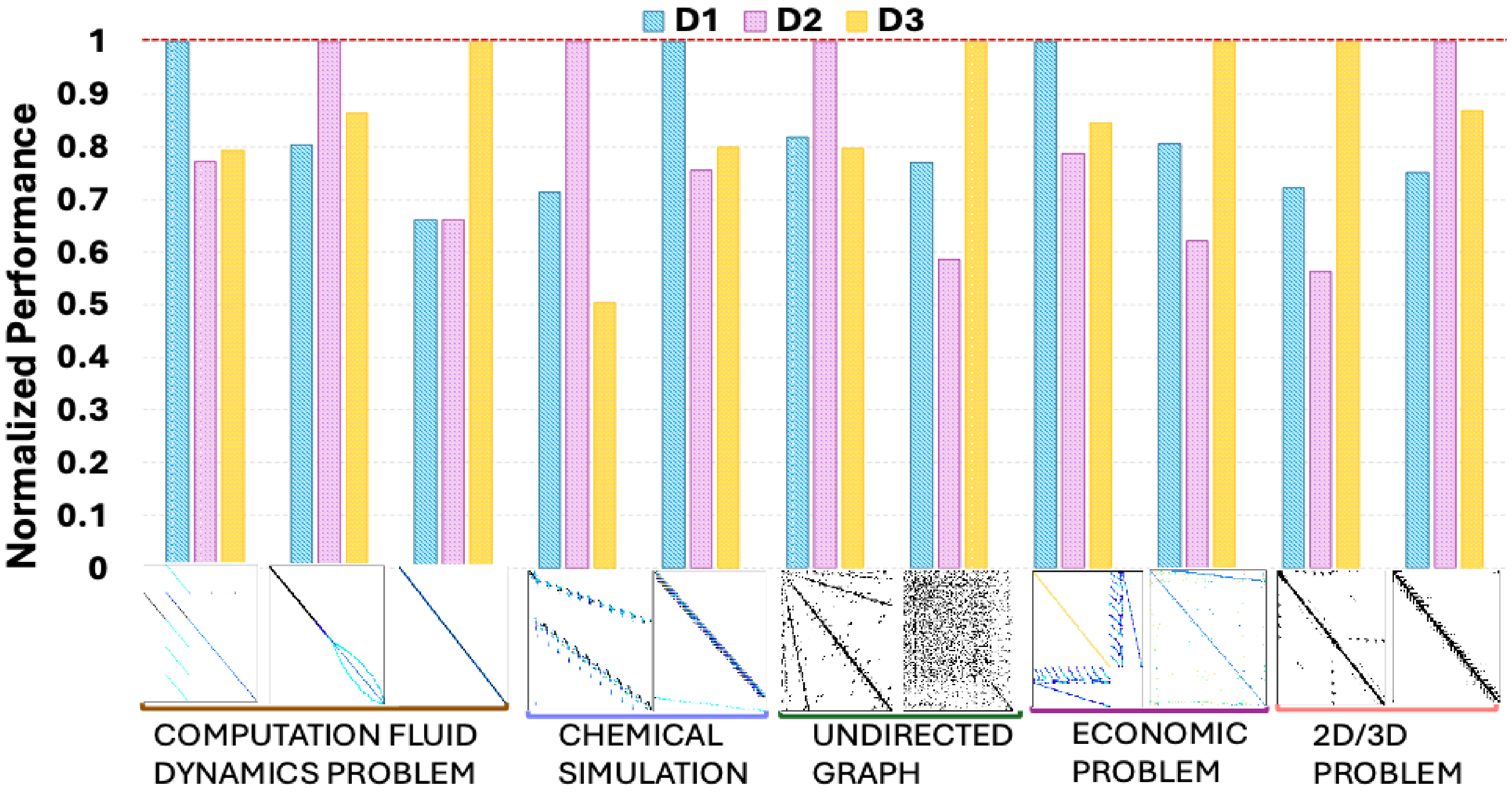
Design 4

Dataflow Design Selection



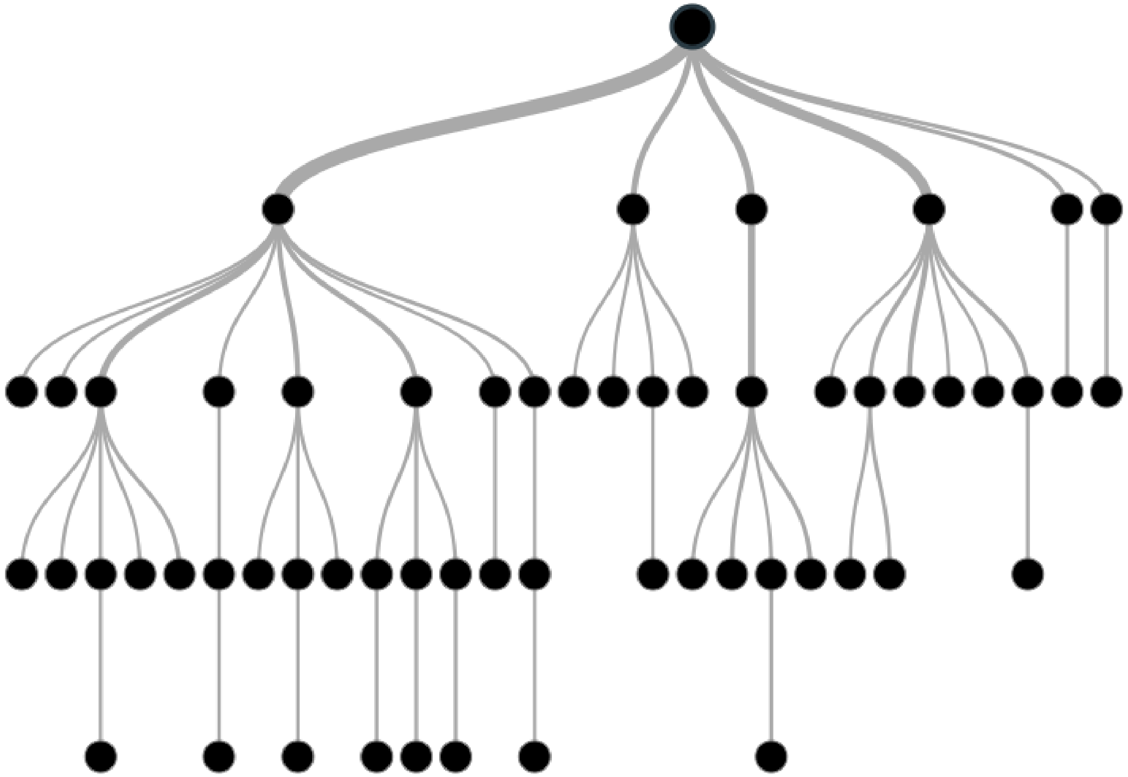
CH_1	1	6	0	5	11	0	8
CH_2	2	7	0	9	0	0	0
. . .							
CH_{N-1}	3	10	0	13	19	0	17
CH_N	4	12	0	15	0	18	0

Why Do We Need an ML Model?



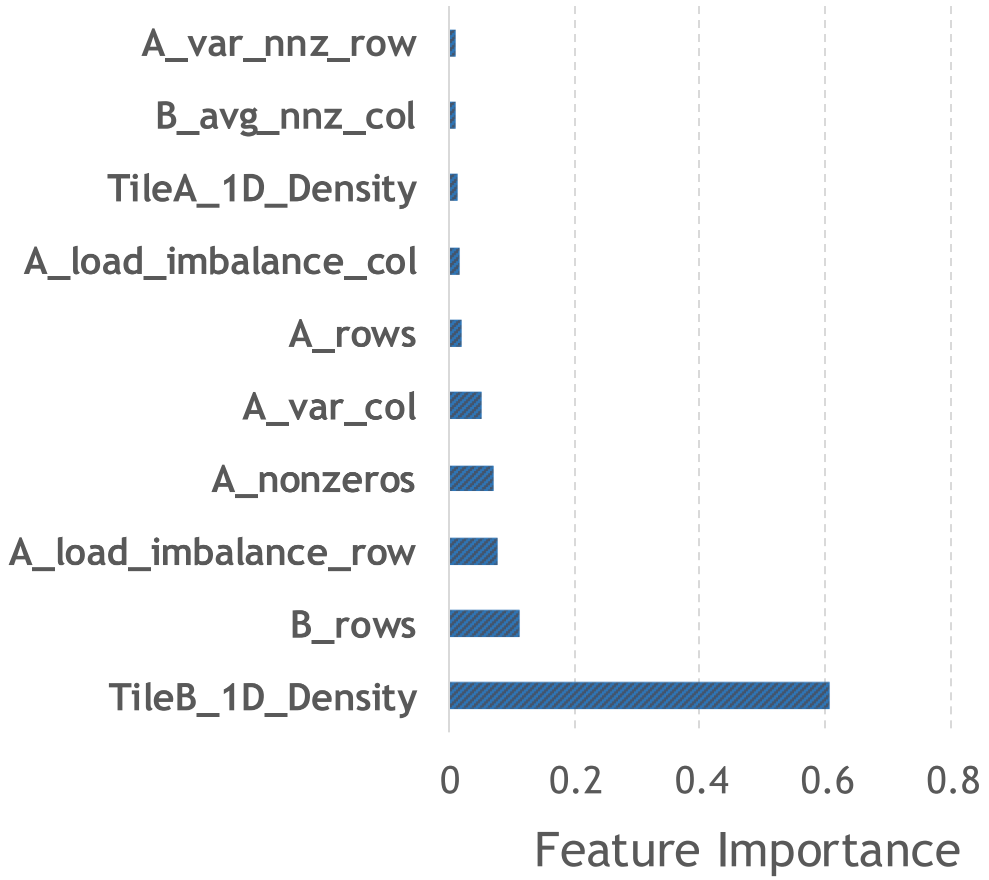
Dataflow Selection Model

- ❑ Light-weight
- ❑ Fast
- ❑ Accurate
- ❑ Generalizable

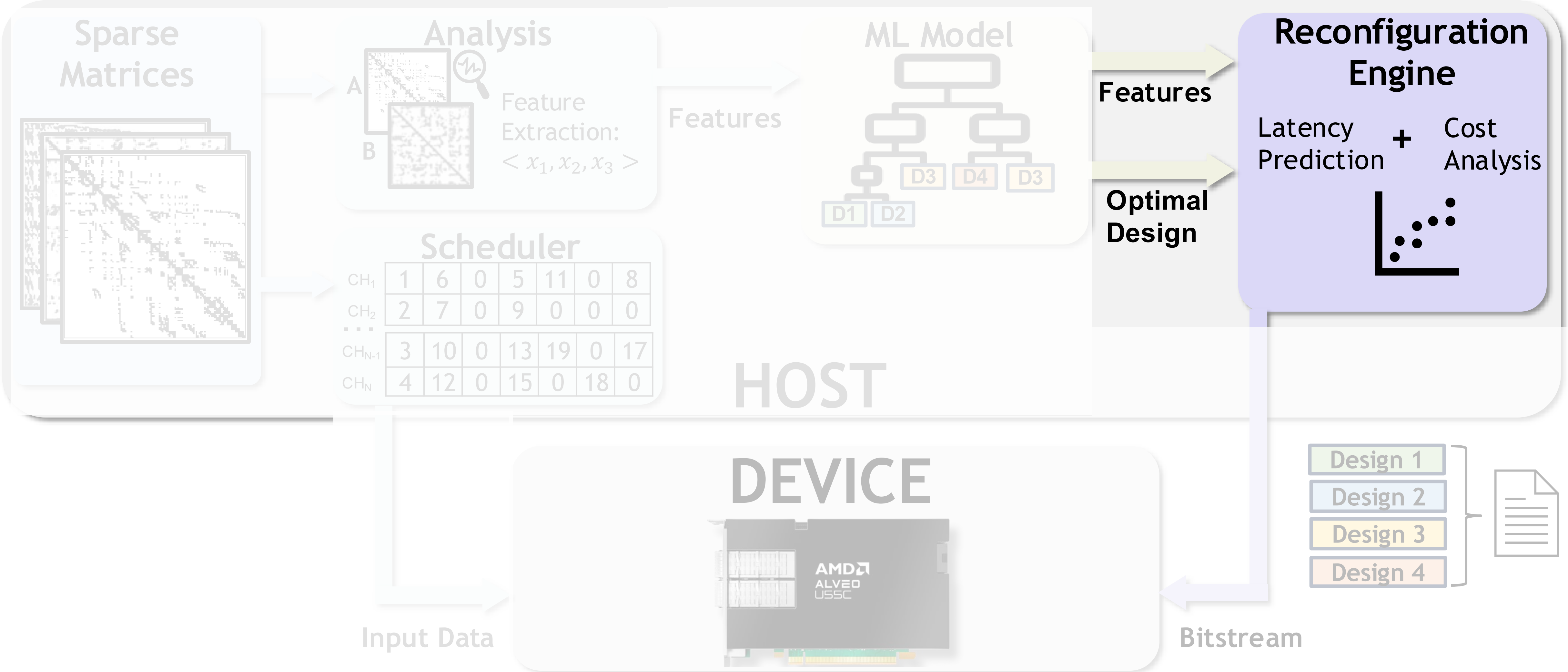


Feature Importance

- Comprehensive list of candidate features extracted from input matrices A and B
- Most influential features are extracted to create a light-weight decision tree model.



Reconfiguration Engine



Sparse Matrices

Analysis

A

B

Feature Extraction: $< x_1, x_2, x_3 >$

Features

ML Model

D3 D4 D3

D1 D2

Features

Optimal Design

Reconfiguration Engine

Latency Prediction + Cost Analysis

Scheduler

CH_1	1	6	0	5	11	0	8
CH_2	2	7	0	9	0	0	0
. . .							
CH_{N-1}	3	10	0	13	19	0	17
CH_N	4	12	0	15	0	18	0

HOST

DEVICE

Input Data

Bitstream

Design 1

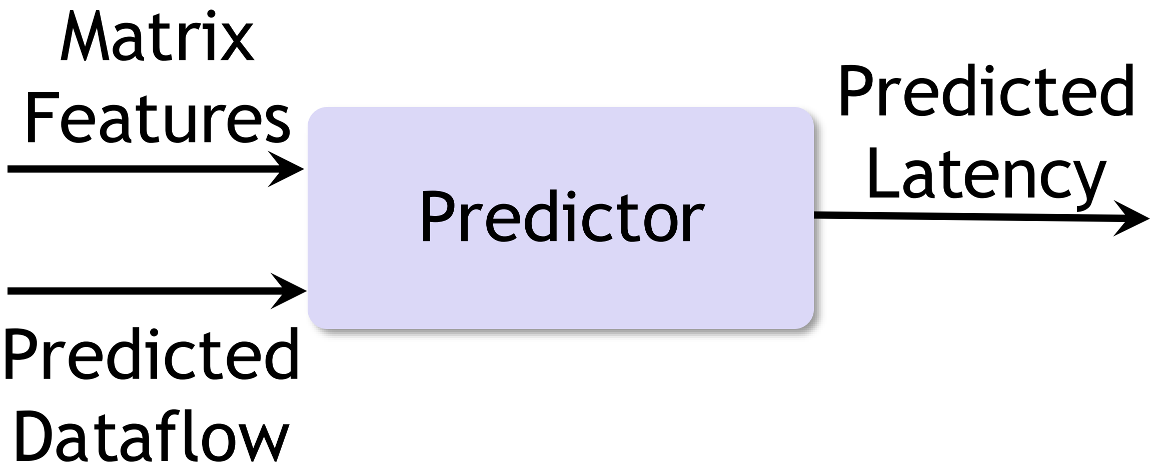
Design 2

Design 3

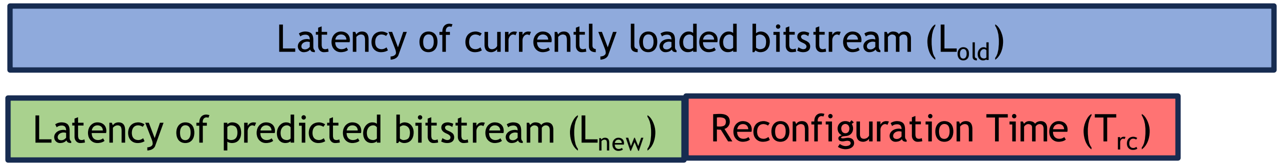
Design 4

When should FPGA be reconfigured?

- Latency predictor model



- Cost Analysis

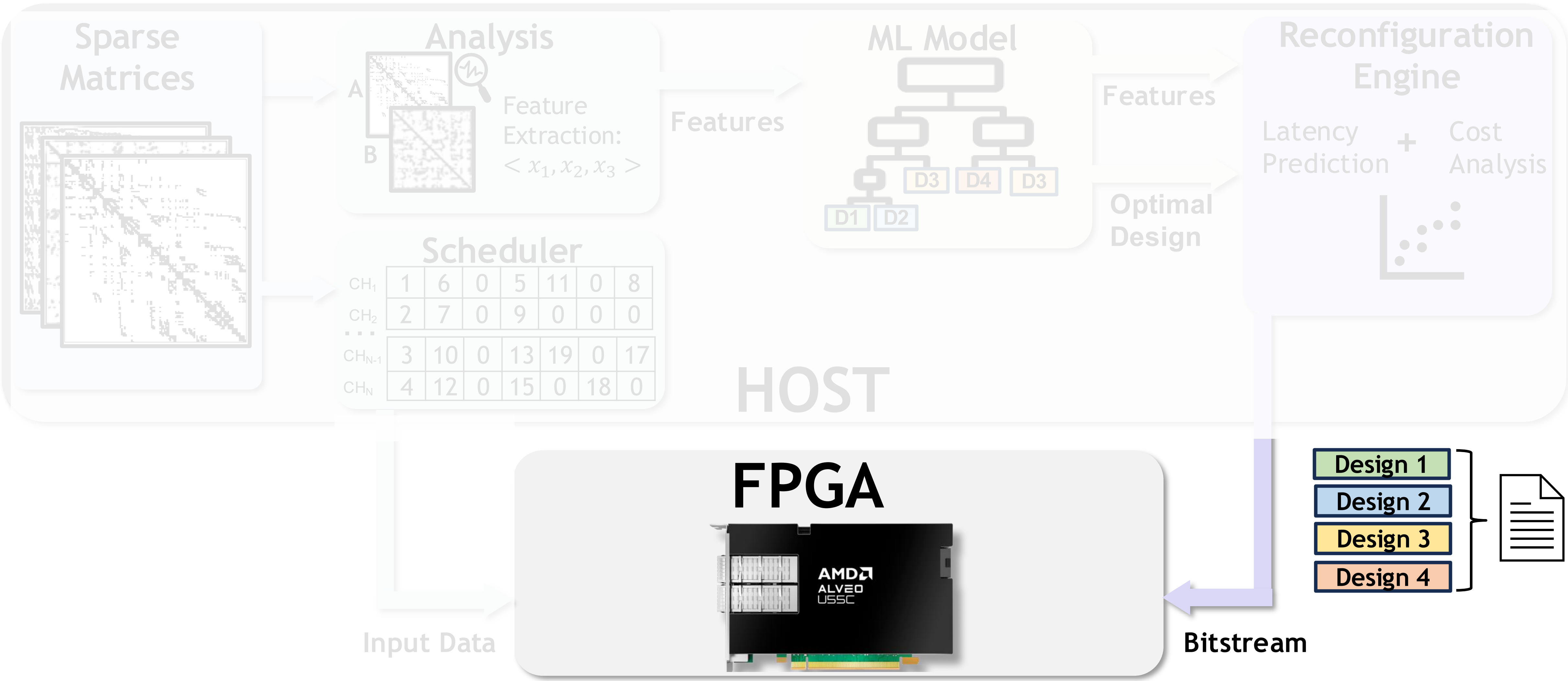

Latency of currently loaded bitstream (L_{old})
Latency of predicted bitstream (L_{new})
Reconfiguration Time (T_{rc})

t is user defined threshold

Reconfigure iff:

$$\boxed{L_{new} < L_{old} \text{ and } T_{rc} <= t\ (L_{old} - L_{new})}$$

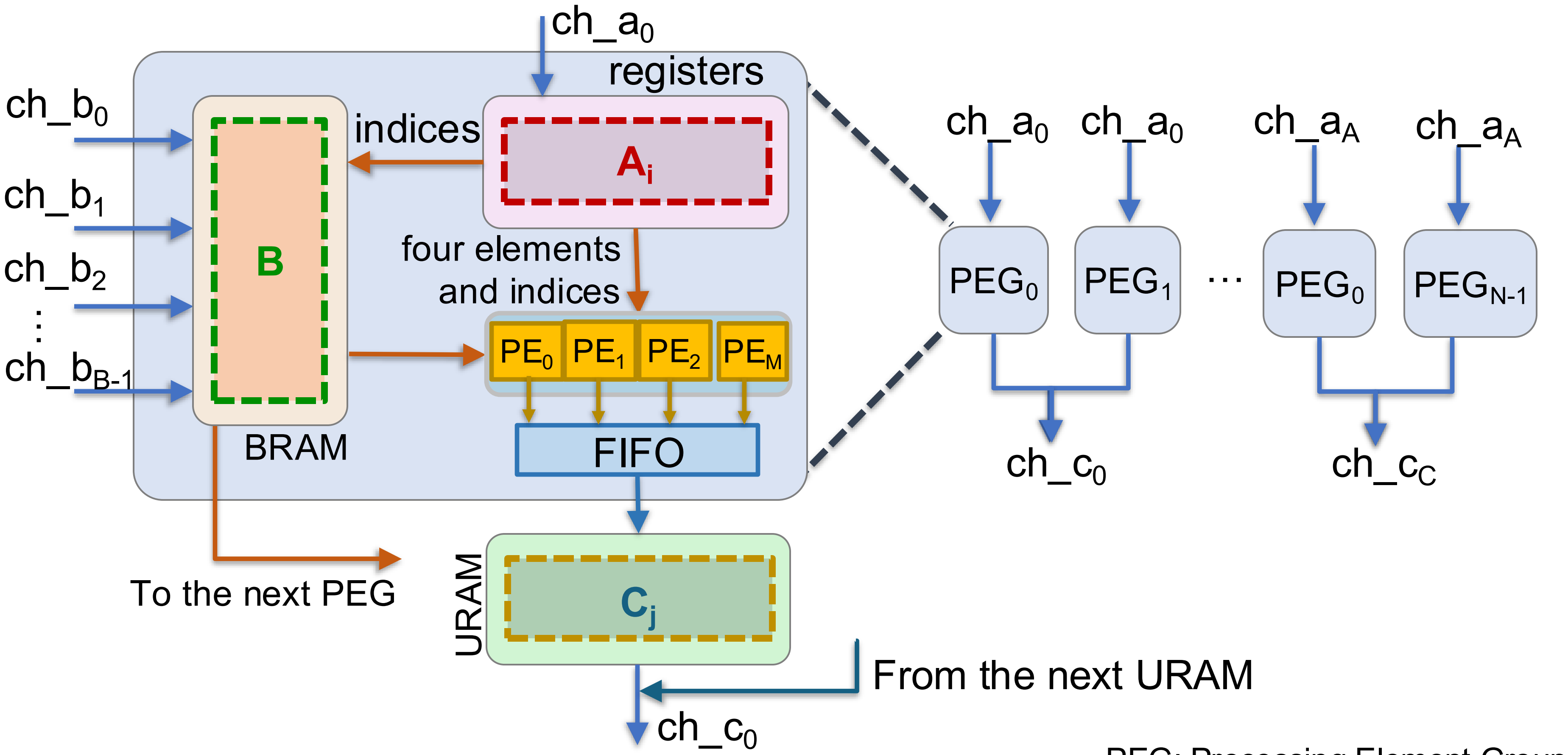
FPGA-Based SpMM & SpGEMM Dataflow Schemes



CH_1	1	6	0	5	11	0	8
CH_2	2	7	0	9	0	0	0
. . .							
CH_{N-1}	3	10	0	13	19	0	17
CH_N	4	12	0	15	0	18	0

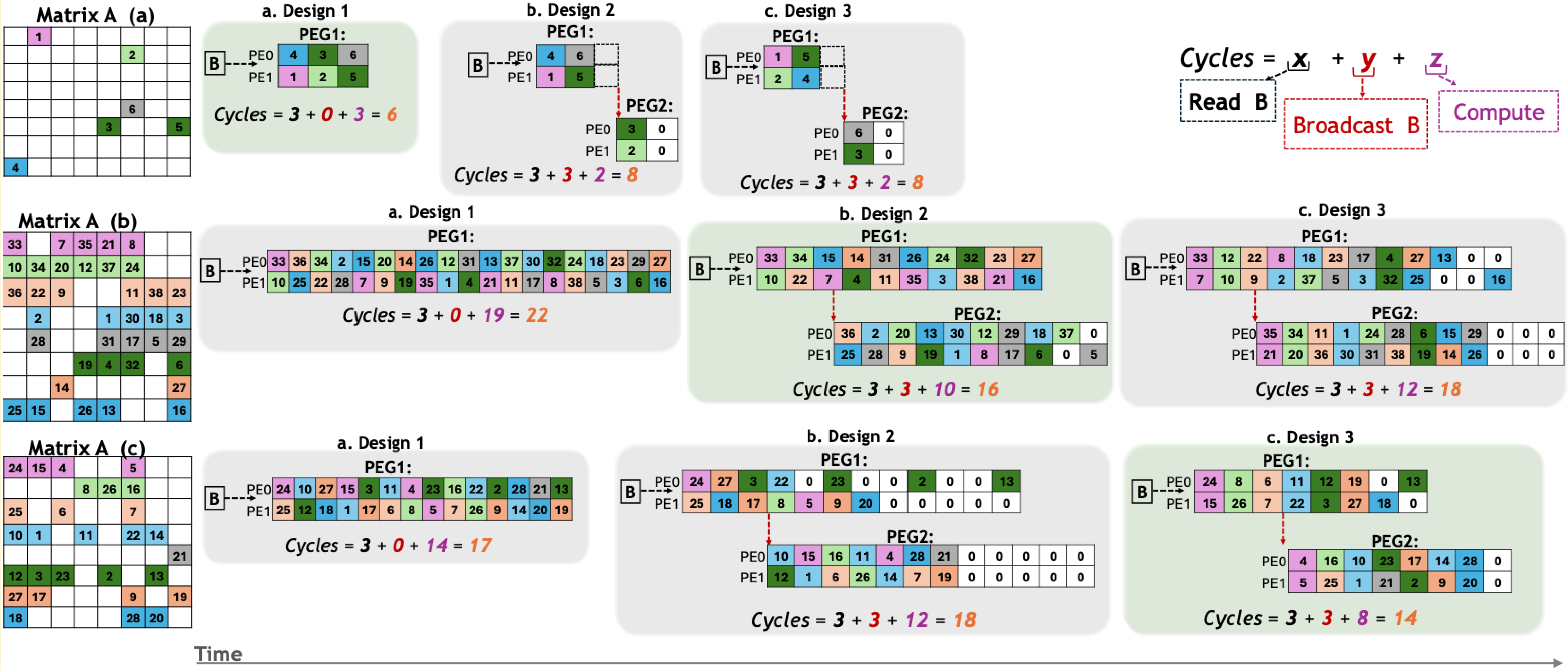
Misam Architecture

ID	Description
A	No. of input A channels
B	No. of input B channels
C	No. of output C channels
N	No. of processing element groups (PEG)
M	No. of processing elements (PE)



PEG: Processing Element Group

Details about FPGA configurations in paper.



Cycles = $\underbrace{x}_{\text{Read B}}$ + $\underbrace{y}_{\text{Broadcast B}}$ + $\underbrace{z}_{\text{Compute}}$

Matrix A (a)

	1						
					2		
					6		
				3			5
4							

a. Design 1

PEG1:

PE0	4	3	6
PE1	1	2	5

Cycles = *3* + *0* + *3* = *6*

b. Design 2

PEG1:

PE0	4	6
PE1	1	5

PEG2:

PE0	3	0
PE1	2	0

Cycles = *3* + *3* + *2* = *8*

c. Design 3

PEG1:

PE0	1	5
PE1	2	4

PEG2:

PE0	6	0
PE1	3	0

Cycles = *3* + *3* + *2* = *8*

Matrix A (b)

33		7	35	21	8		
10	34	20	12	37	24		
36	22	9			11	38	23
	2			1	30	18	3
	28			31	17	5	29
			19	4	32		6
		14					27
25	15		26	13			16

a. Design 1

PEG1:

PE0	33	36	34	2	15	20	14	26	12	31	13	37	30	32	24	18	23	29	27
PE1	10	25	22	28	7	9	19	35	1	4	21	11	17	8	38	5	3	6	16

Cycles = *3* + *0* + *19* = *22*

b. Design 2

PEG1:

PE0	33	34	15	14	31	26	24	32	23	27
PE1	10	22	7	4	11	35	3	38	21	16

PEG2:

PE0	36	2	20	13	30	12	29	18	37	0
PE1	25	28	9	19	1	8	17	6	0	5

Cycles = *3* + *3* + *10* = *16*

c. Design 3

PEG1:

PE0	33	12	22	8	18	23	17	4	27	13	0	0
PE1	7	10	9	2	37	5	3	32	25	0	0	16

PEG2:

PE0	35	34	11	1	24	28	6	15	29	0	0	0
PE1	21	20	36	30	31	38	19	14	26	0	0	0

Cycles = *3* + *3* + *12* = *18*

Matrix A (c)

24	15	4			5		
			8	26	16		
25		6			7		
10	1		11		22	14	
							21
12	3	23		2		13	
27	17				9		19
18					28	20	

a. Design 1

PEG1:

PE0	24	10	27	15	3	11	4	23	16	22	2	28	21	13
PE1	25	12	18	1	17	6	8	5	7	26	9	14	20	19

Cycles = *3* + *0* + *14* = *17*

b. Design 2

PEG1:

PE0	24	27	3	22	0	23	0	0	2	0	0	13
PE1	25	18	17	8	5	9	20	0	0	0	0	0

PEG2:

PE0	10	15	16	11	4	28	21	0	0	0	0	0
PE1	12	1	6	26	14	7	19	0	0	0	0	0

Cycles = *3* + *3* + *12* = *18*

c. Design 3

PEG1:

PE0	24	8	6	11	12	19	0	13
PE1	15	26	7	22	3	27	18	0

PEG2:

PE0	4	16	10	23	17	14	28	0
PE1	5	25	1	21	2	9	20	0

Cycles = *3* + *3* + *8* = *14*

Experimental Setup

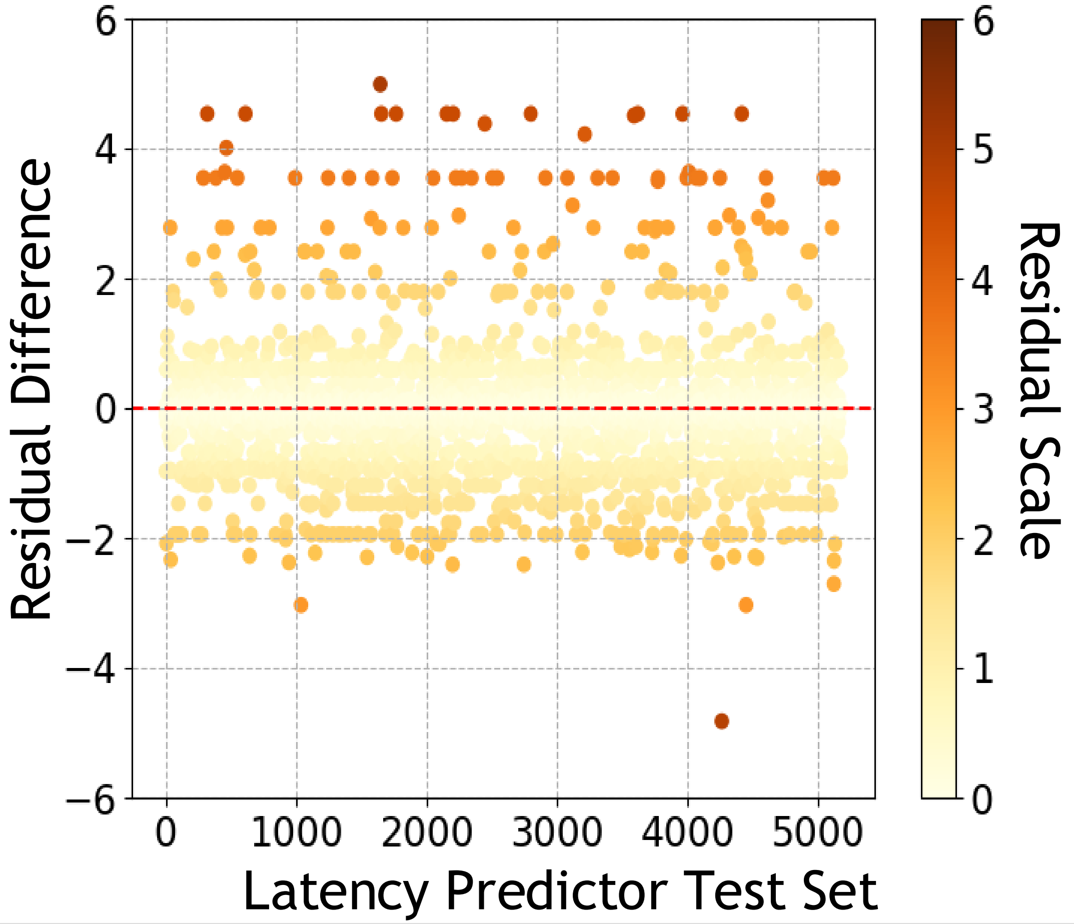
- **Target FPGA**: AMD Alveo U55C
- **Baselines:**
  - cuSPARSE executed on an NVIDIA RTX A6000 GPU
  - Intel Math Kernel Library (MKL) run on Intel Core i9-11980HK CPU
  - Trapezoid (ISCA 24)
- **Dataset:** SuiteSparse, DNN workloads
- **Metrics:** Latency, Energy Consumption

Decision Tree Model

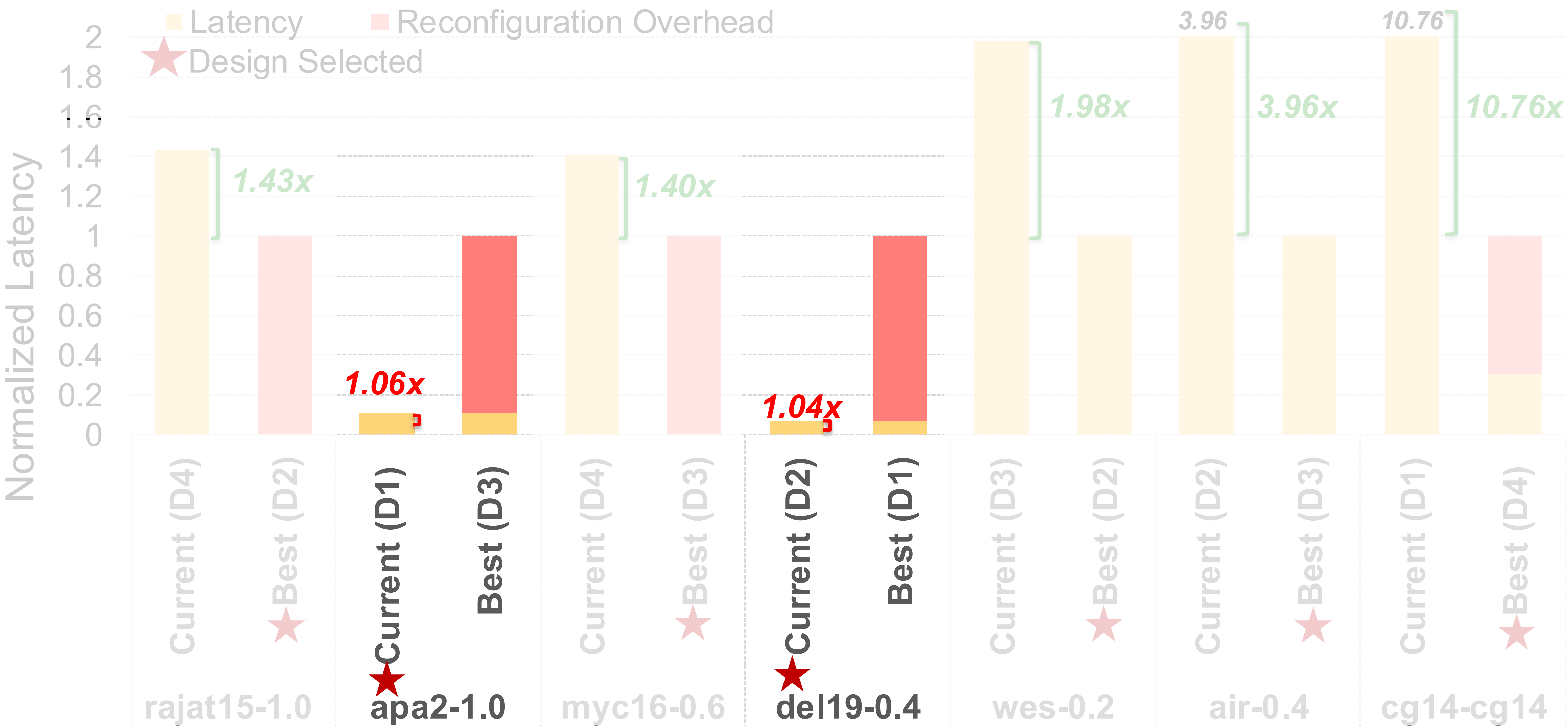
- ☑ Light-weight → 6KB storage on host
- ☑ Fast → Inference is 0.1% of end-to-end runtime
- ☑ Accurate → 90% accuracy
- ☑ Generalizable → Retrainable for different scenarios

Latency Predictor Model

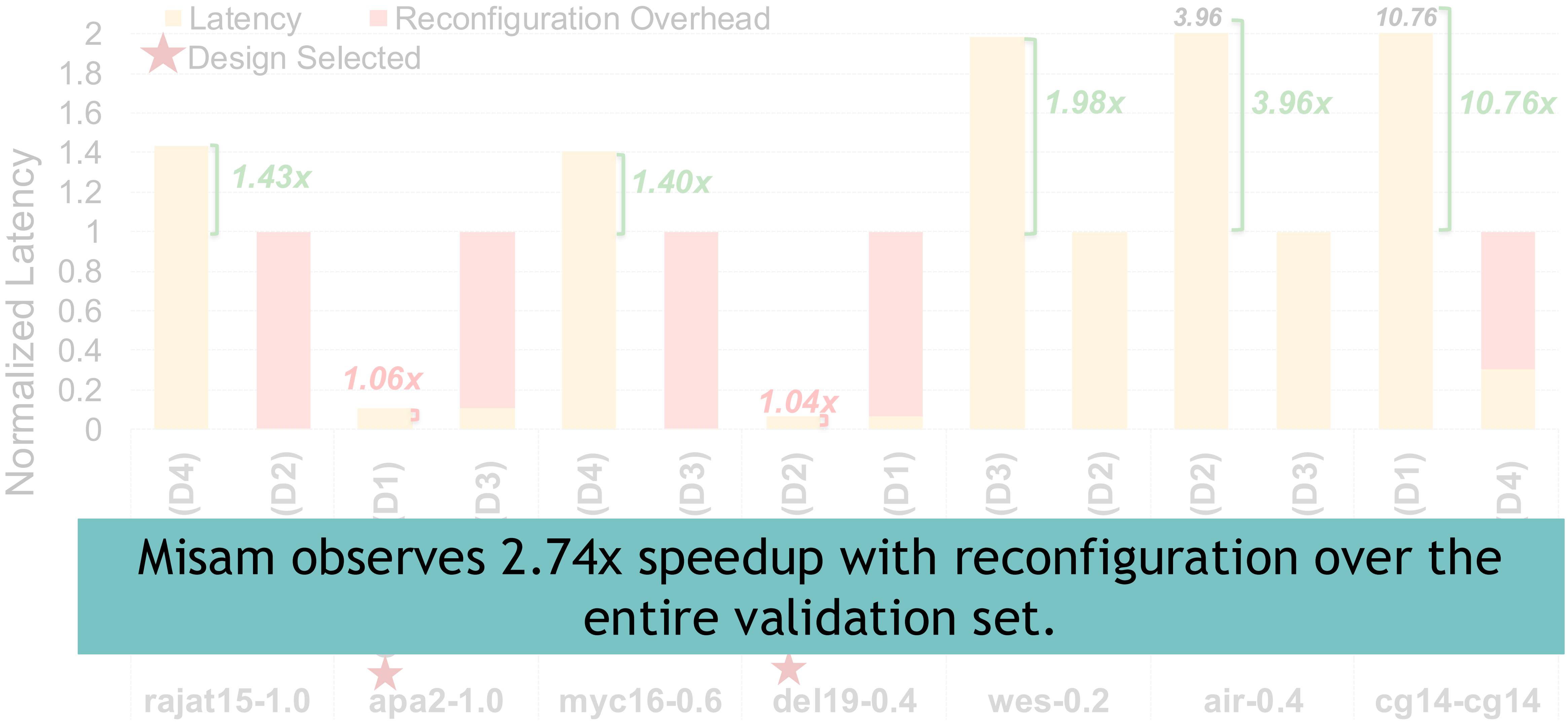
- Dataset comprising of 19,000 matrices
- $R^2 = 0.978$



Reconfiguration Engine

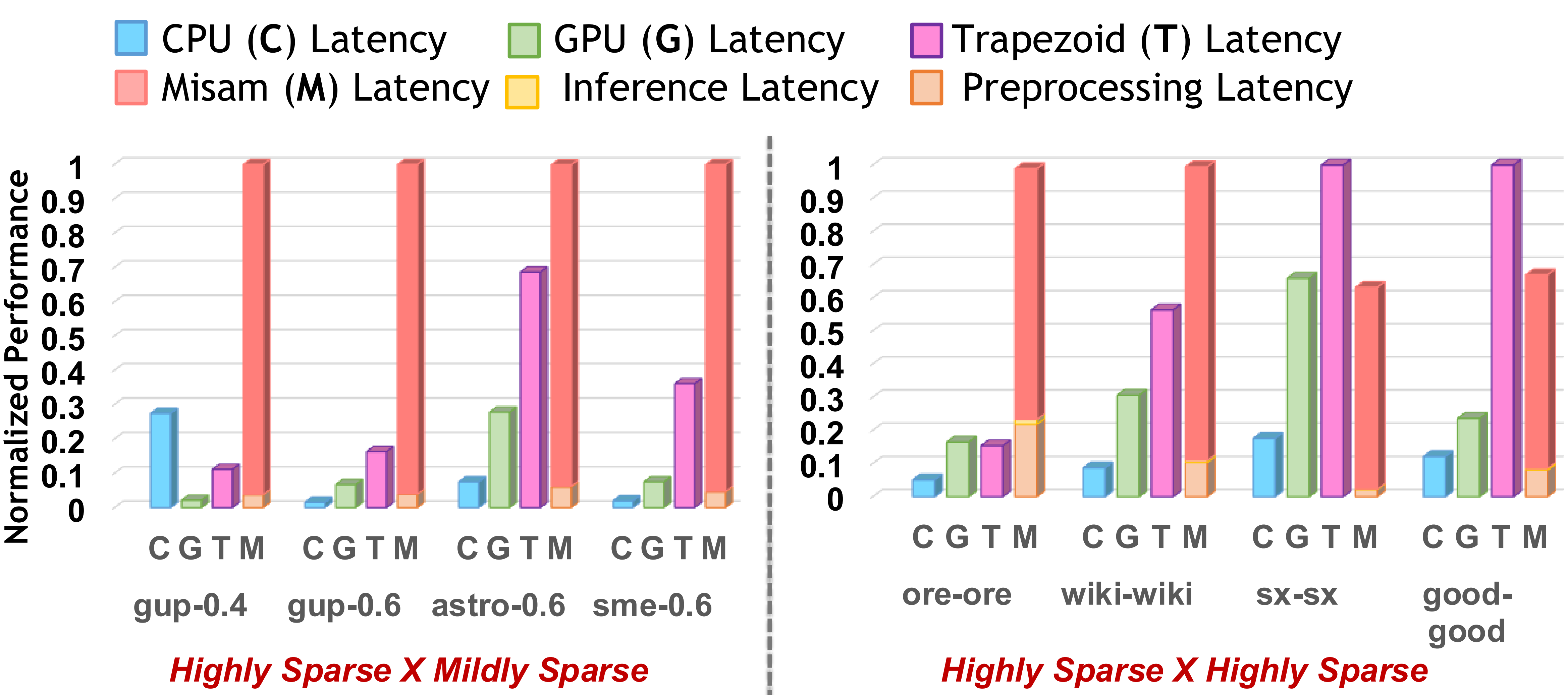


Reconfiguration Engine



Misam observes 2.74x speedup with reconfiguration over the entire validation set.

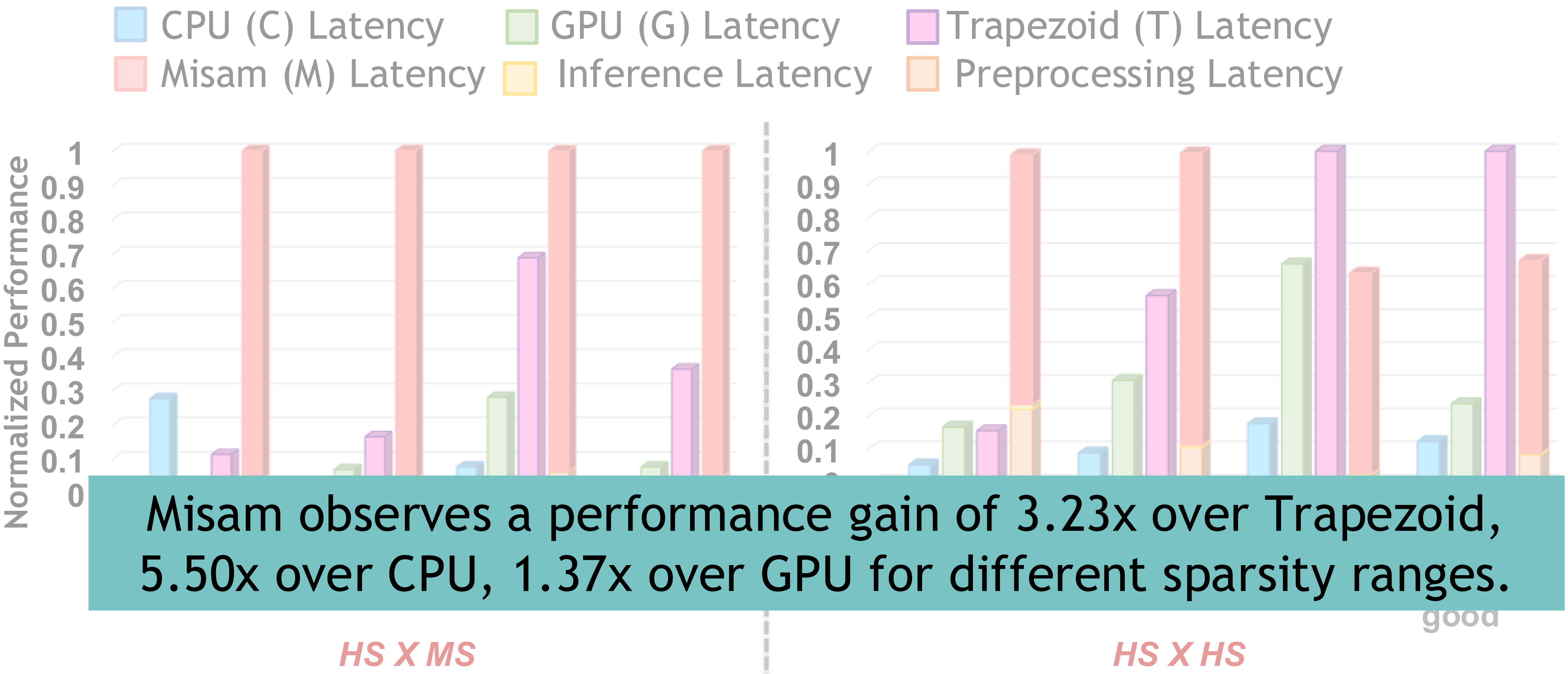
End-to-End Performance Comparison



Highly Sparse X Mildly Sparse

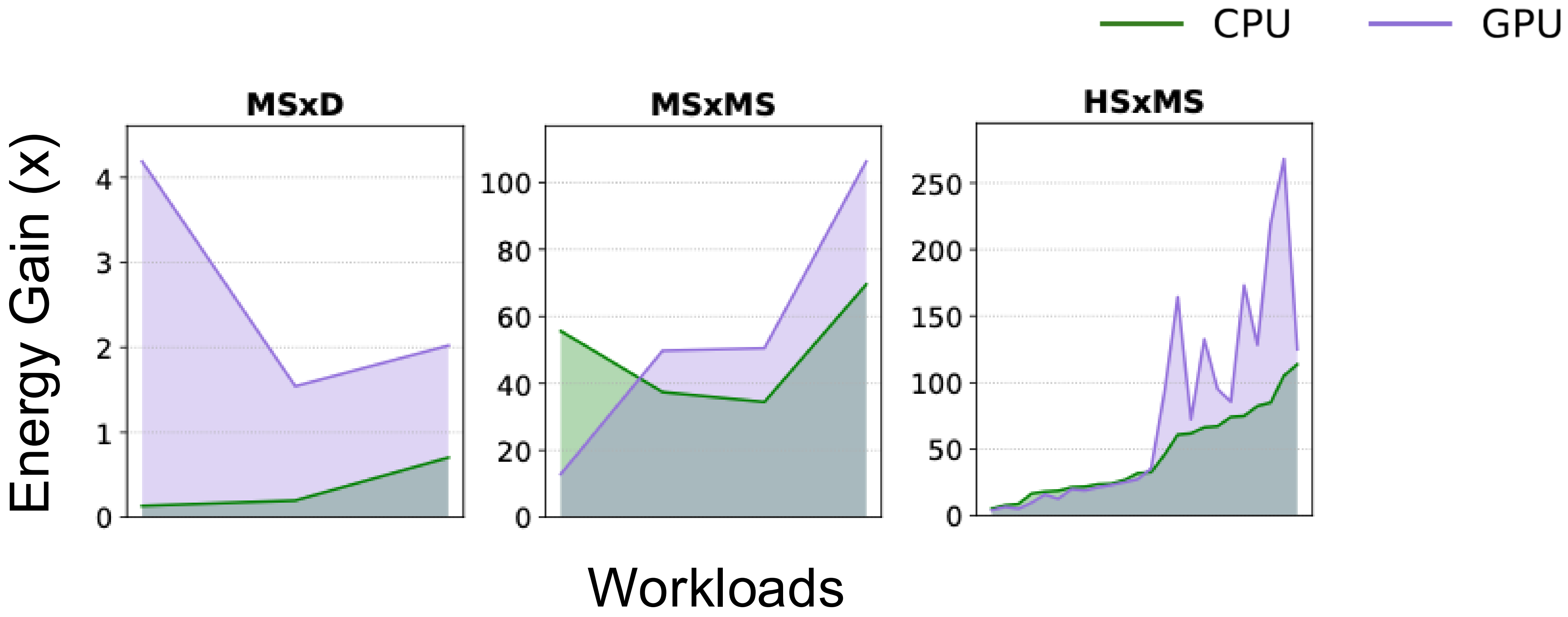
Highly Sparse X Highly Sparse

End-to-End Performance Comparison

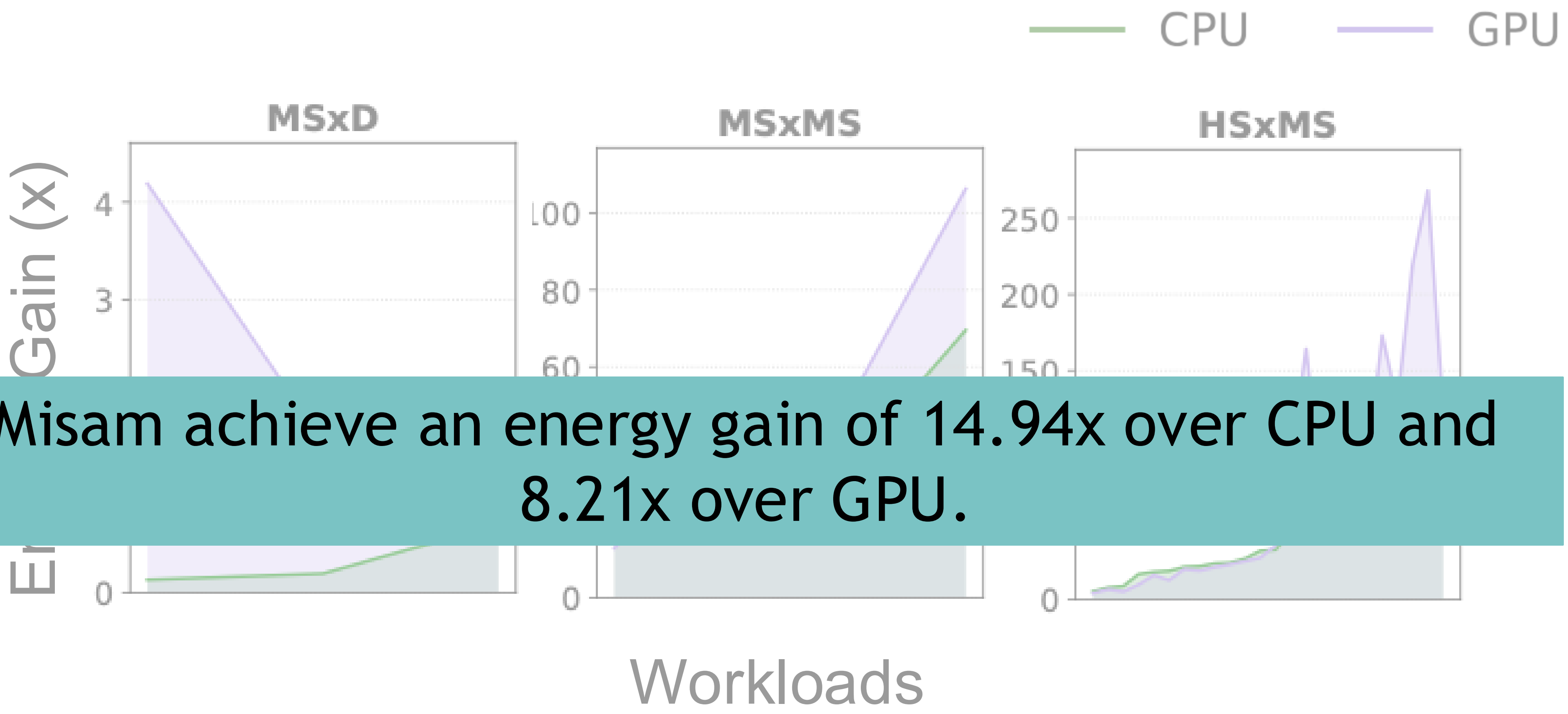


Misam observes a performance gain of 3.23x over Trapezoid, 5.50x over CPU, 1.37x over GPU for different sparsity ranges.

Energy Gains of Misam over CPU and GPU

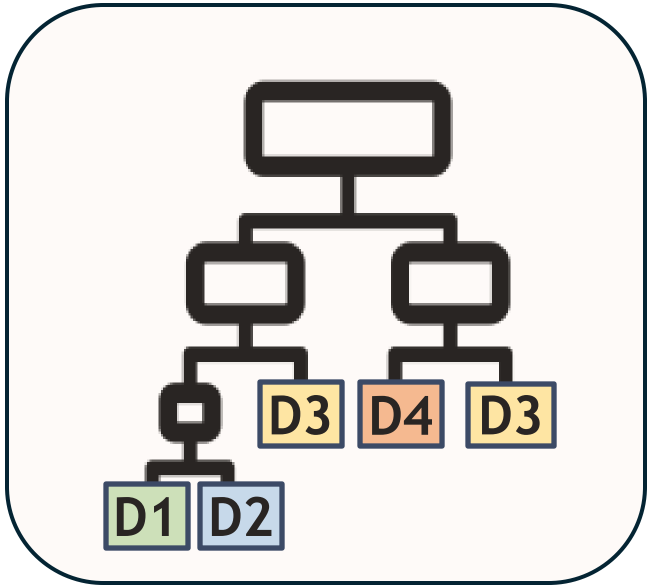


Energy Gains of Misam over CPU and GPU



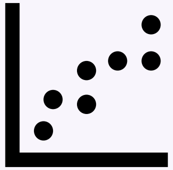
Misam achieve an energy gain of 14.94x over CPU and 8.21x over GPU.

Summary of Misam

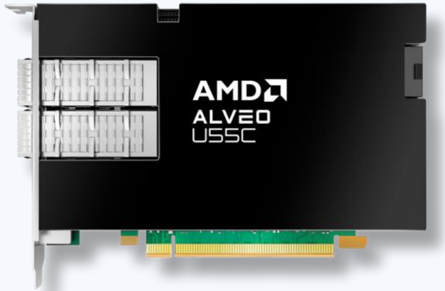


Dataflow Selection Model

Latency Predictor + Cost Analysis



Reconfiguration Engine



FPGA-Base kernels

Research Question 3:

Where in DSAs a sparsity-aware decision can help?

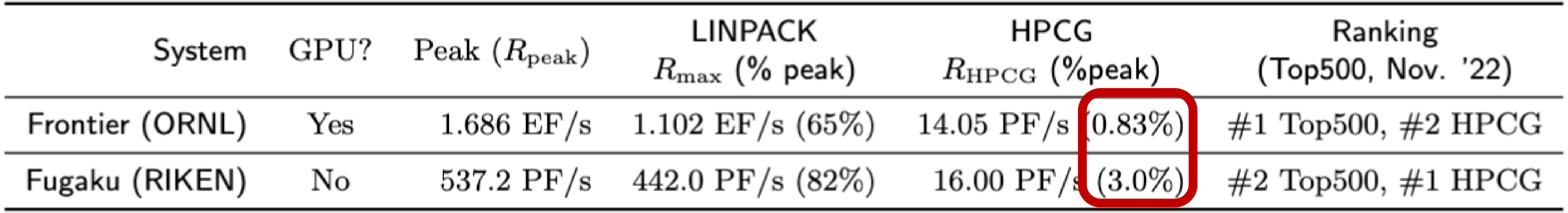
1. Algorithm selection
2. Improving resource underutilization

Acamar: A Dynamically Reconfigurable Scientific Computing Accelerator for Robust Convergence and Minimal Resource Utilization

- Led by Ubaid Bakhtiar, a 3rd year PhD student at CASL
- **MICRO 2024**
- **Authors:** Ubaid Bakhtiar, Helia Hosseini, and Bahar Asgari

We don’t want DSAs to face the same issue

HPCG benchmark, which is dominated by distributed sparse matrix-vector multiplication



System	GPU?	Peak (R_{peak})	LINPACK $R_{\max}$ (% peak)	HPCG R_{HPCG} (%peak)	Ranking (Top500, Nov. ’22)
Frontier (ORNL)	Yes	1.686 EF/s	1.102 EF/s (65%)	14.05 PF/s (0.83%)	#1 Top500, #2 HPCG
Fugaku (RIKEN)	No	537.2 PF/s	442.0 PF/s (82%)	16.00 PF/s (3.0%)	#2 Top500, #1 HPCG

Motivation - The Efficiency Crisis

Modern supercomputers achieve <5% peak performance on real workloads

- Fugaku: 3.6% efficiency on HPCG benchmark
- Frontier: 1.1% efficiency on HPCG benchmark

Scientific computing is critical for:

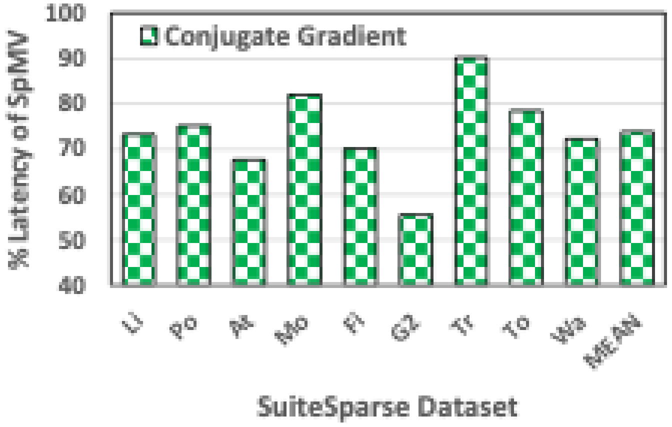
- Fluid dynamics simulations
- Electromagnetics modeling
- Aeronautical calculations

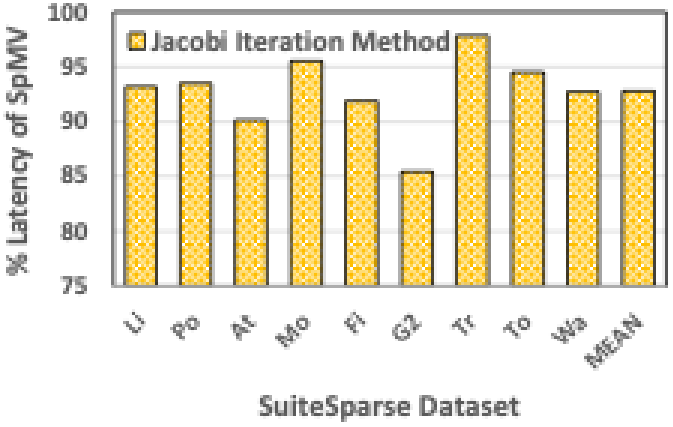
Solving Ax = b in Scientific Computing

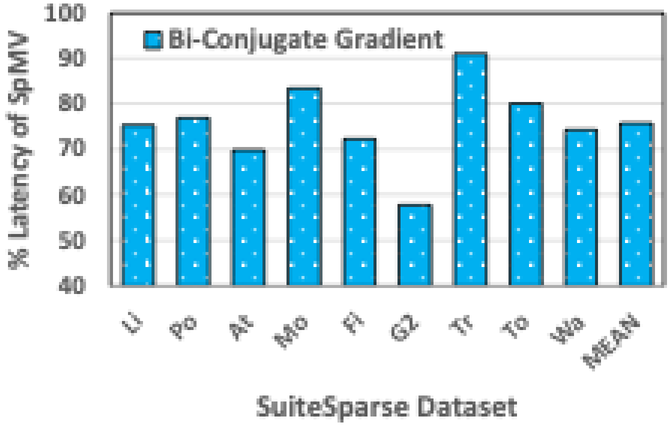
Most scientific problems reduce to solving Ax = b

- A: Sparse coefficient matrix
- b: Constant vector
- x: Solution vector

Sparse Matrix-Vector multiplication (SpMV) consumes 75-95% of solver execution time







Two Major Challenges

1. Solution Divergence

- Different solvers require specific matrix properties:
  - Jacobi (JB): Strictly diagonally dominant matrices
  - Conjugate Gradient (CG): Symmetric, positive definite
  - BiCG-STAB: Non-symmetric matrices
- No single solver works for all matrices!

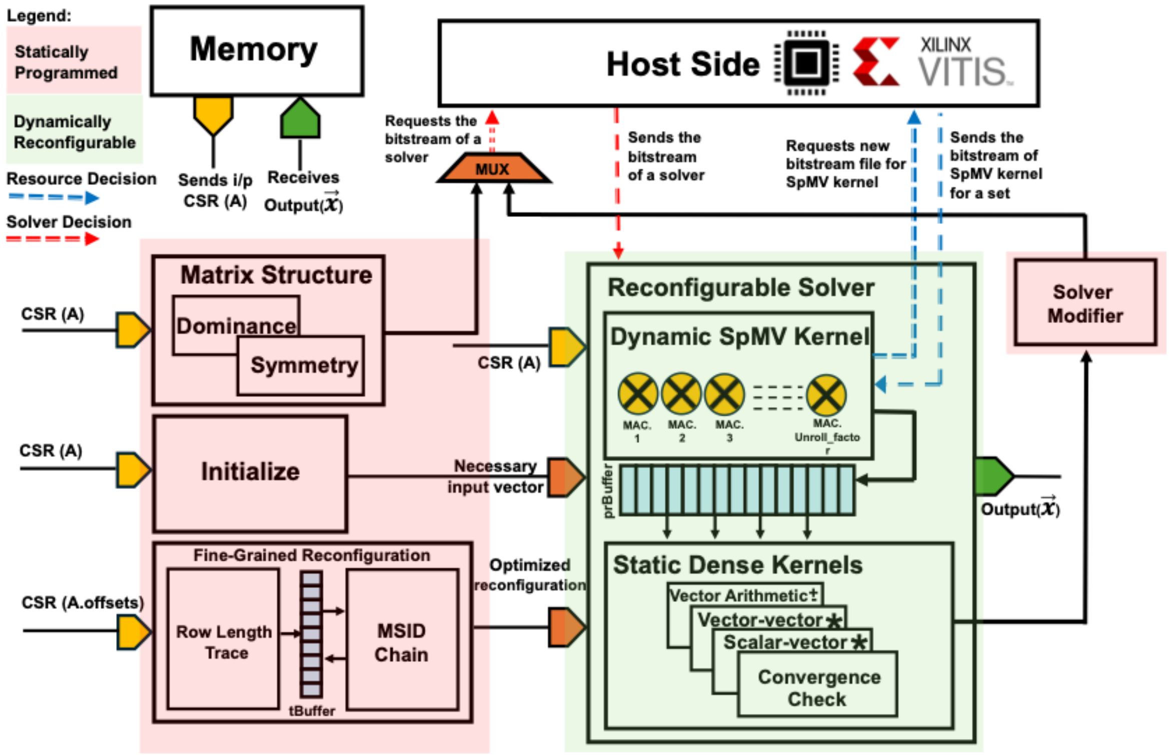
Two Major Challenges

2. Resource Underutilization

- Static designs waste resources due to varying sparsity patterns
- Uneven distribution of non-zeros leads to inefficiency

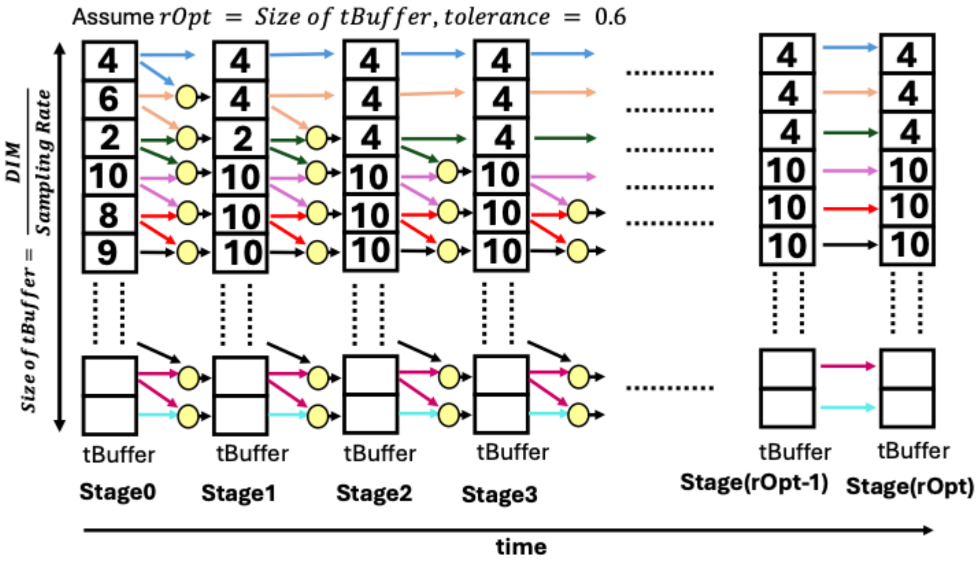
Acamar's Solution: Dynamic Reconfiguration at Two Levels

1. **Solver-Level Reconfiguration**
   - Dynamically switch between JB, CG, and BiCG-STAB
   - Ensures convergence for any matrix structure
2. **Fine-Grained SpMV Reconfiguration**
   - Adapt resource allocation based on sparsity pattern
   - Optimize unroll factor per matrix section



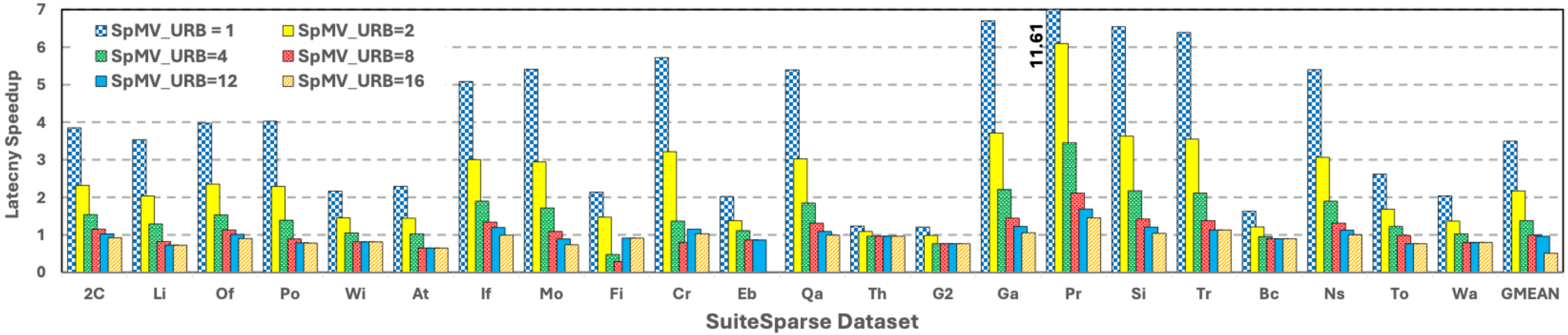
Further Optimization

- Minimizes reconfiguration overhead
- Balances between:
  - Resource utilization improvement
  - Reconfiguration cost
- Maintains performance while reducing reconfigurations by up to 80%



Experimental Results: Speedup

- Up to 11.61× speedup vs. unoptimized baseline
- Average 6× speedup across benchmarks
- Consistent improvements across diverse matrix structures

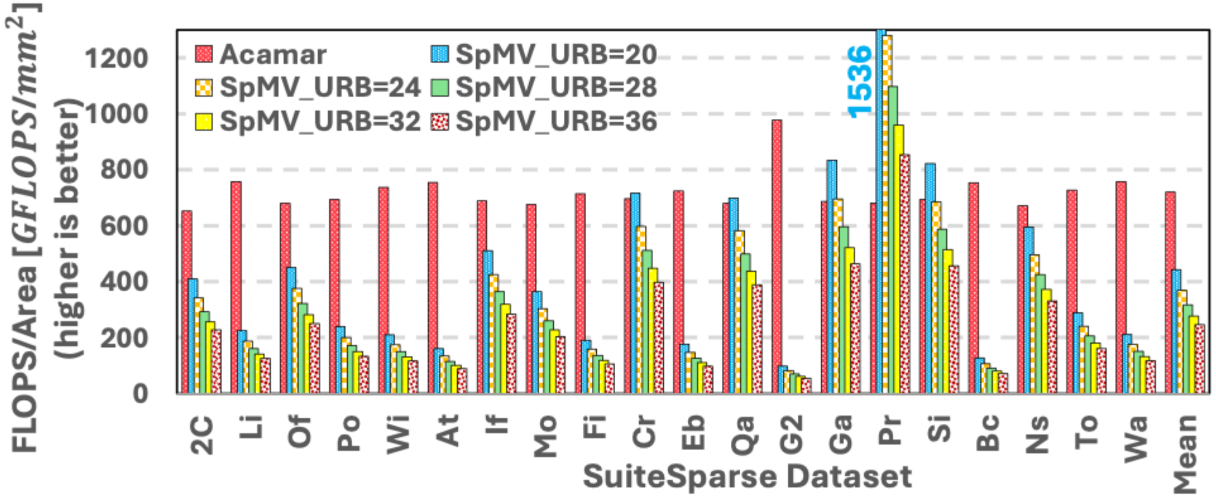


Performance Efficiency: FLOPS per area

- Acamar achieves 720 GFLOPS/mm2 average
- 2× more area efficient than static designs
- Enables multi-tenant execution on same FPGA

Key Benefits:

- Better resource sharing
- Higher computational density
- Lower cost per operation



Impact of Acamar

- Addresses inefficiency in modern supercomputers
- Enables more efficient scientific simulations
- Opens new research directions in adaptive computing

CASL @UMD CS

- Learn more about our team:
  casl.umd.edu.cs
- Recent updates on LinkedIn:

https://www.linkedin.com/company/**casl-research-group**/

Sponsors:

DoE ASCR
ECRP2023

NSF
PPoSS