IISWC 2022

Nov 8, 2022

Demystifying Map Space Exploration for NPUs

Sheng-Chun Kao[1], Angshuman Parashar[2], Po-An Tsai[2], **Tushar Krishna**[1]




Contact: tushar@ece.gatech.edu

1 Georgia Tech School of Electrical and Computer Engineering College of Engineering

2 NVIDIA

https://github.com/maestro-project/gamma-timeloop

Outline

- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

Outline

- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

DNN Applications

“AI is the new electricity” – Andrew Ng

Object Detection

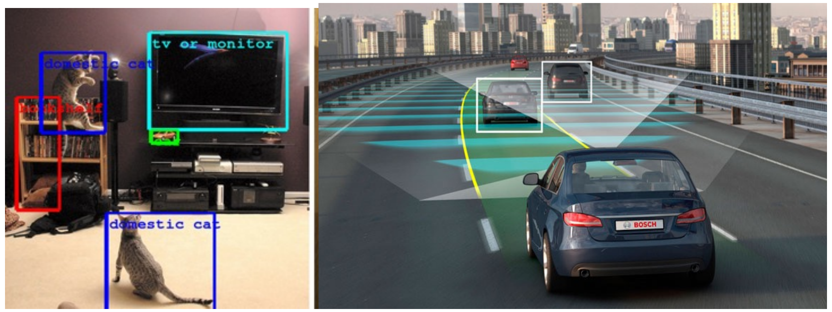
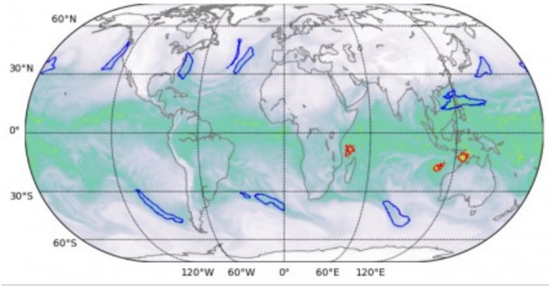
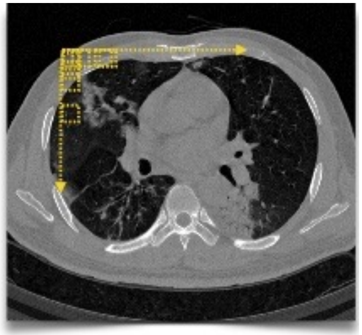


Image Segmentation

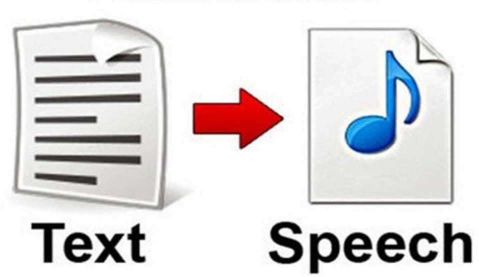


Medical Imaging



Speech Recognition

Text to Speech



Recommendations

Why do we need DNN accelerators (NPUs)?

- **Millions of Parameters (i.e., weights)**
  - Billions of computations → **Need lots of parallel compute**

This makes CPUs inefficient

DNN Topology	Number of Weights
AlexNet (2012)	3.98M
VGGnet-16 (2014)	28.25M
GoogleNet (2015)	6.77M
Resnet-50 (2016)	23M
DLRM (2019)	540M
Megatron (2019)	8.3B

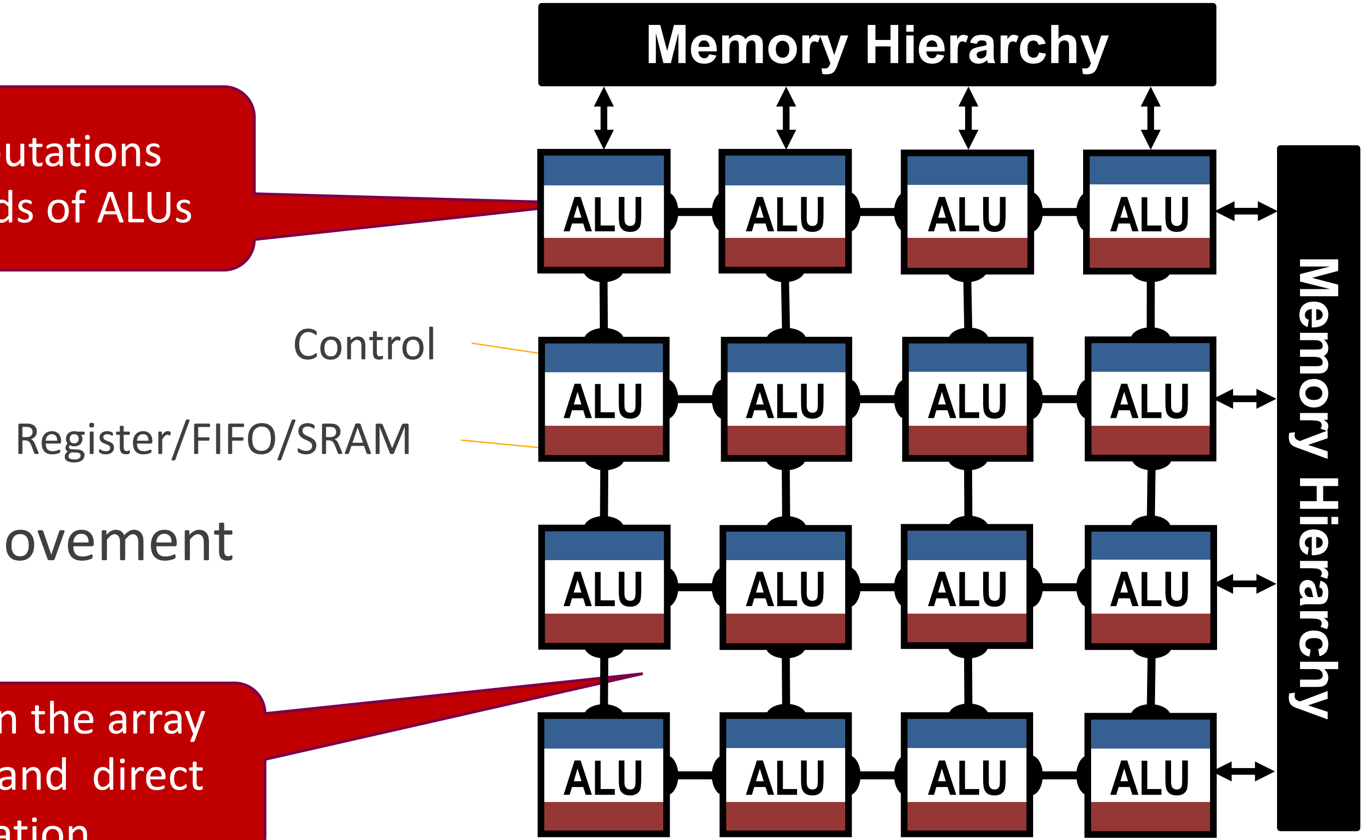
  - Heavy data movement → **Need to reduce energy**

This makes GPUs inefficient

Spatial (or Dataflow) NPUs

- **Millions of Parameters (i.e., weights)**
  - Billions of computations

Spread computations across hundreds of ALUs



  - Heavy data movement

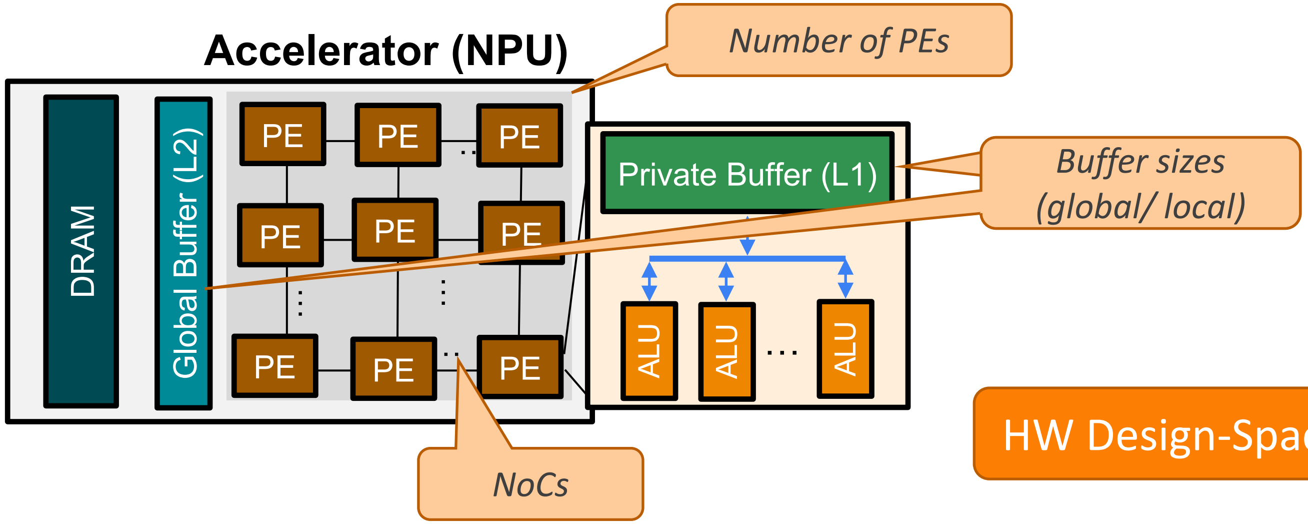
Reuse data within the array via local storage and direct communication

Examples: Google TPU, MIT Eyeriss, …

Outline

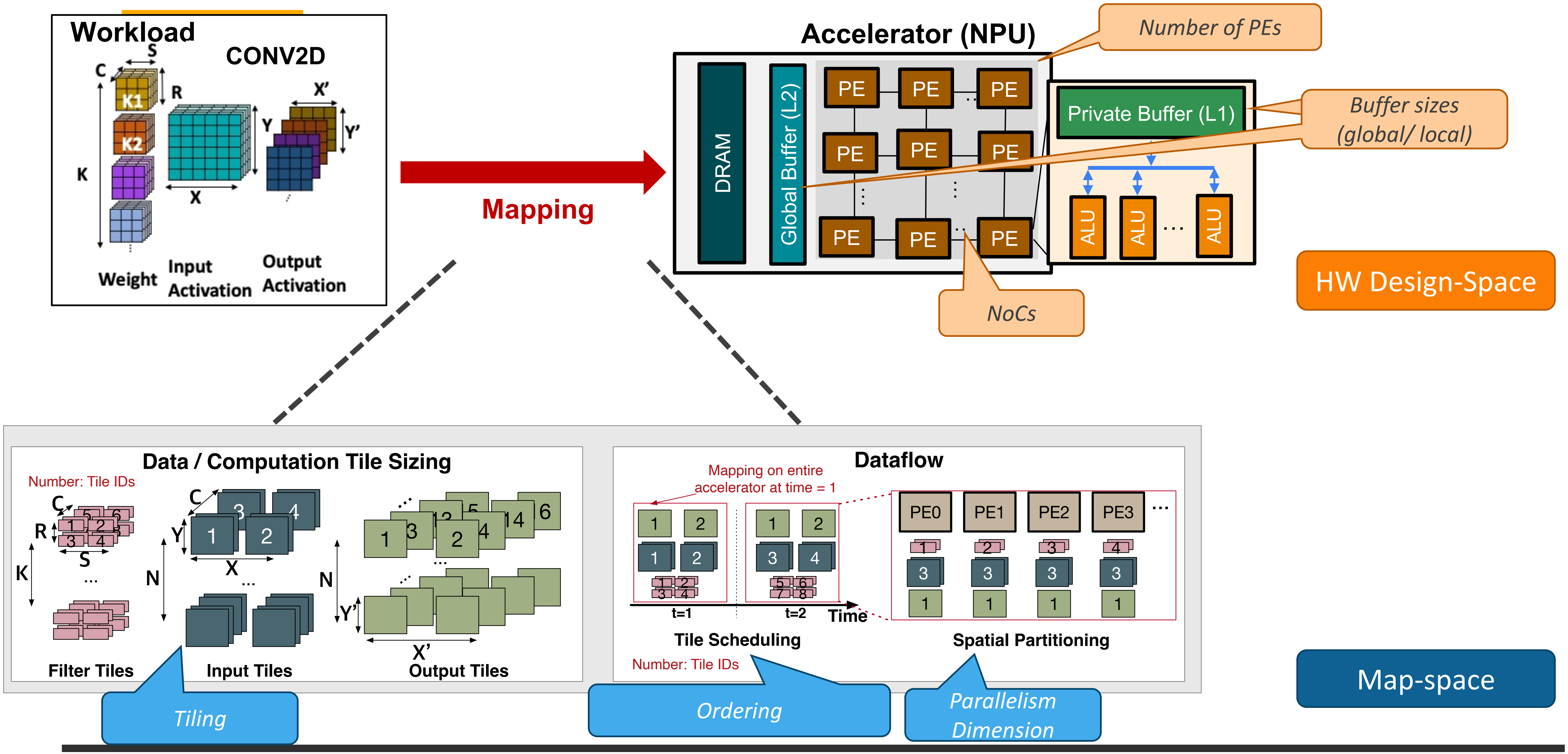
- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

Architectural Components of a DNN Accelerator

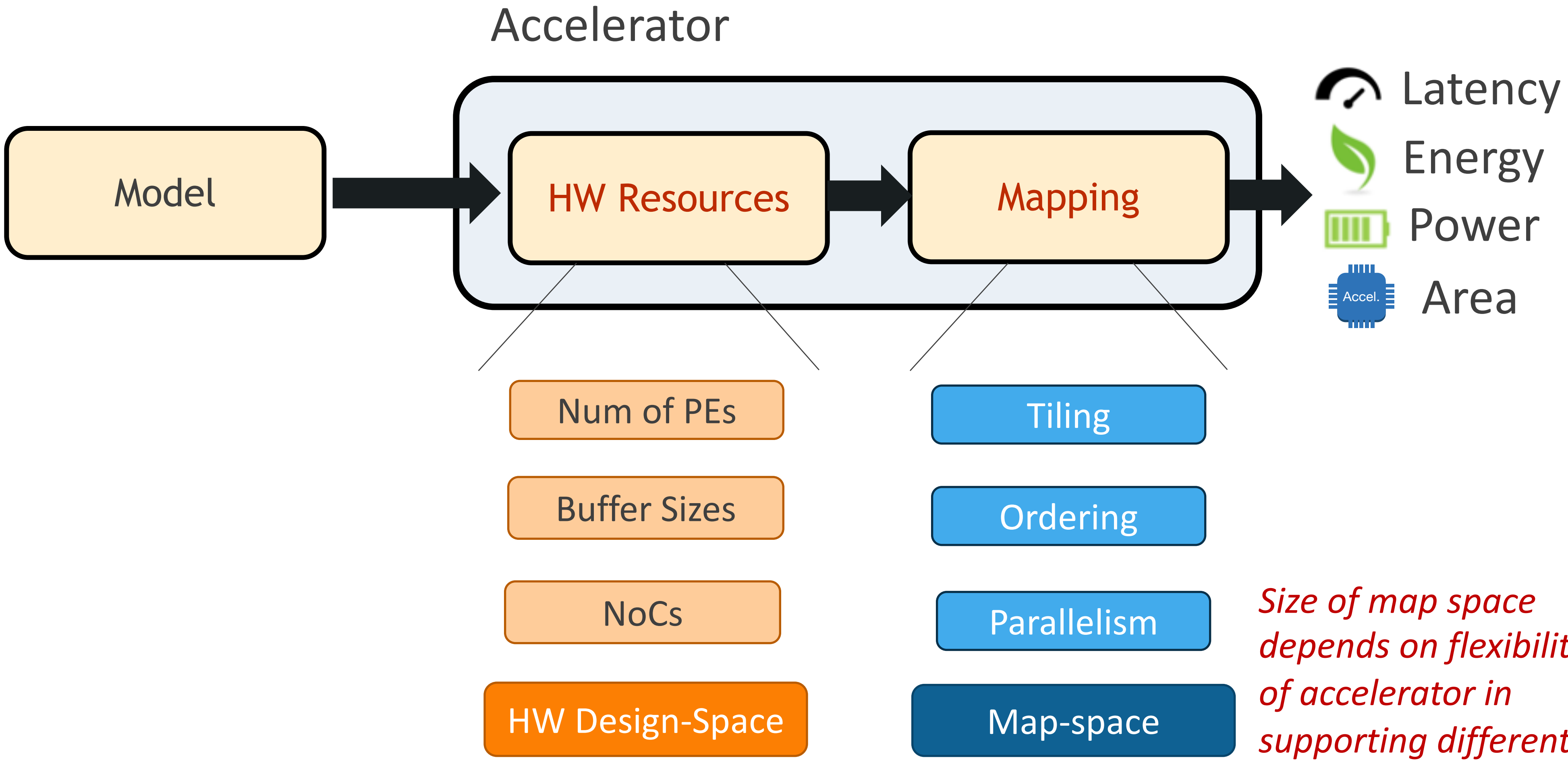


HW Design-Space

Architectural Components of a DNN Accelerator



Design Space of An Accelerator



Size of map space depends on flexibility of accelerator in supporting different ordering, parallelism and tile sizes

Representation of a Mapping

Loop Nest

DRAM Mapping

```
....
 for x= [0, 128, xt):
```

DRAM Tile
DRAM Order
DRAM Par.

L2 Mapping

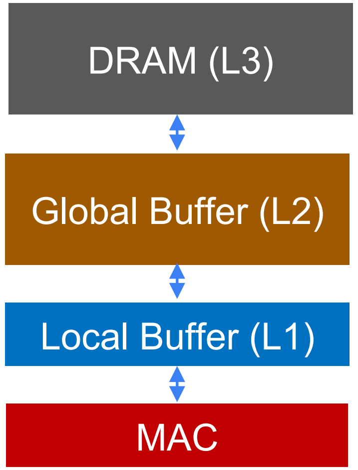
```
for bt=[0, 1, 1)
 for kt=[0, 16, 1):
  for ct=[0, 64, 1):
   for rt=[0, 3, 1):
    for st=[0, 3, 1):
     for yt=[0, 4, 1):
      for xt= [0, 4, 1):
       par_for kp=[0, 16, 1):
        par_for cp=[0, 64, 1):
```

L2 Tile
L2 Order
L2 Par.

L1 Mapping

```
...
 for rtt= [0, 1, 1):
 ...
```

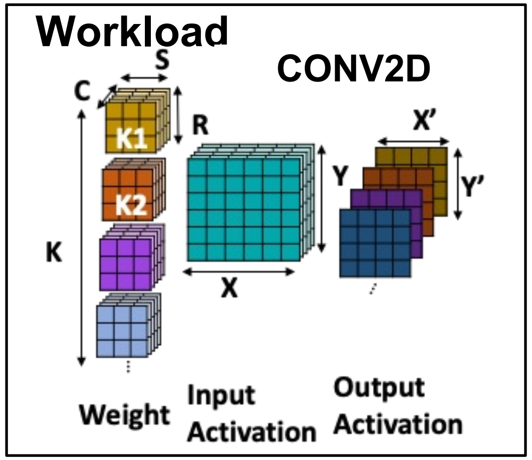
L1 Tile
L1 Order
L1 Par.



Alternate representations also exist:
- data-centric (Kwon et al MICRO 2019)
- memory-centric (Mei et al., IEEE Trans. Comp. 2021)

Example of Mapping run by NVDLA

Focus: Intra-layer / Intra-operator Mappings



Map

L3 Mapping

```
...
for kt=[0, 16, 1):
  for ct=[0, 64, 1):
    for rt=[0, 3, 1):
      for st=[0, 3, 1):
        for yt=[0, 4, 1):
          for xt= [0, 4, 1):
            par_for kp=[0, 16, 1):
              par_for cp= [0, 64, 1):
```

Tile Sizes

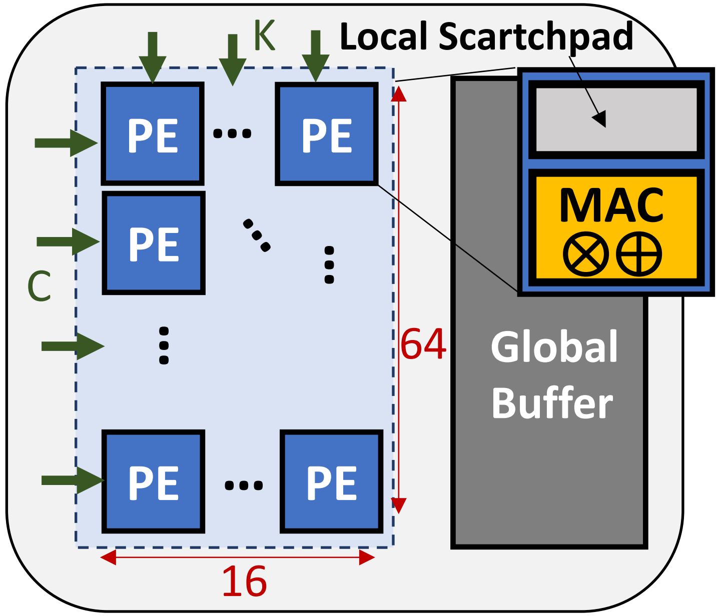
L2 Mapping

Loop Order

Loop Parallelization

...

L1 Mapping



Mapping - Orchestrating Data Movement

L3 Mapping

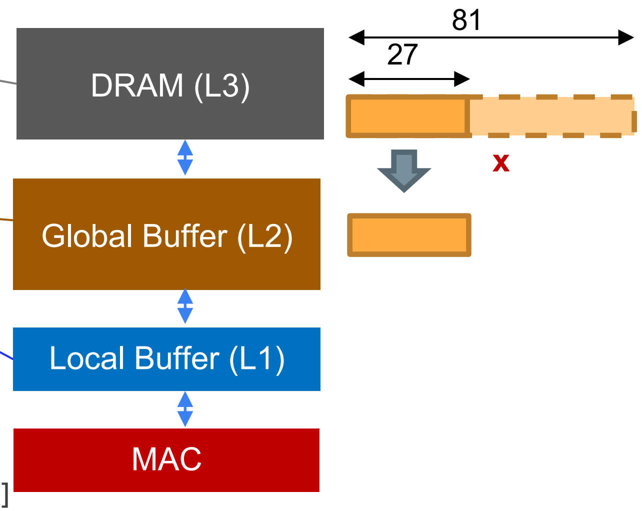
```
for s2= [0, 9, 9):
  for x2= [0, 81, 27):
```

L2 Mapping

```
for s1= [0, 9, 3):
  for x1= [0, 27, 3):
```

L1 Mapping

```
for s0= [0, 3, 1):
  for x0= [0, 3, 1):
    x = x2+x1+x0
    s = s2+s1+s0
    PartialSum[x][s] = Weight[s]*Input[x+s]
    Output[x] += PartialSum[x][s]
```



L3 Mapping

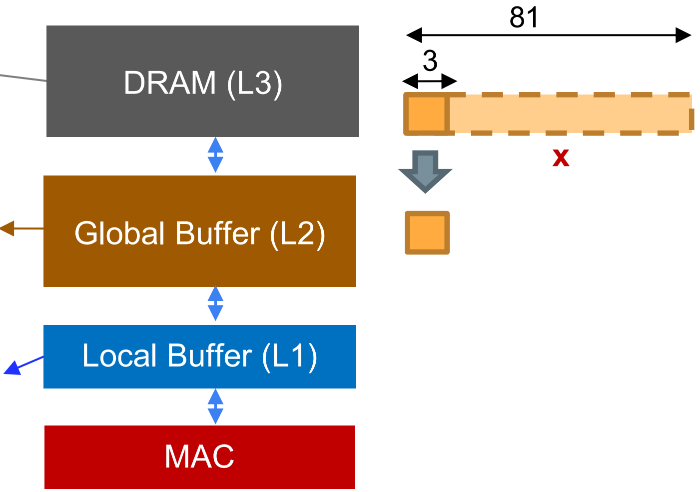
```
for s2= [0, 9, 9):
  for x2= [0, 81, 3):
```

L2 Mapping

```
for s1= [0, 9, 3):
  for x1= [0, 3, 3):
```

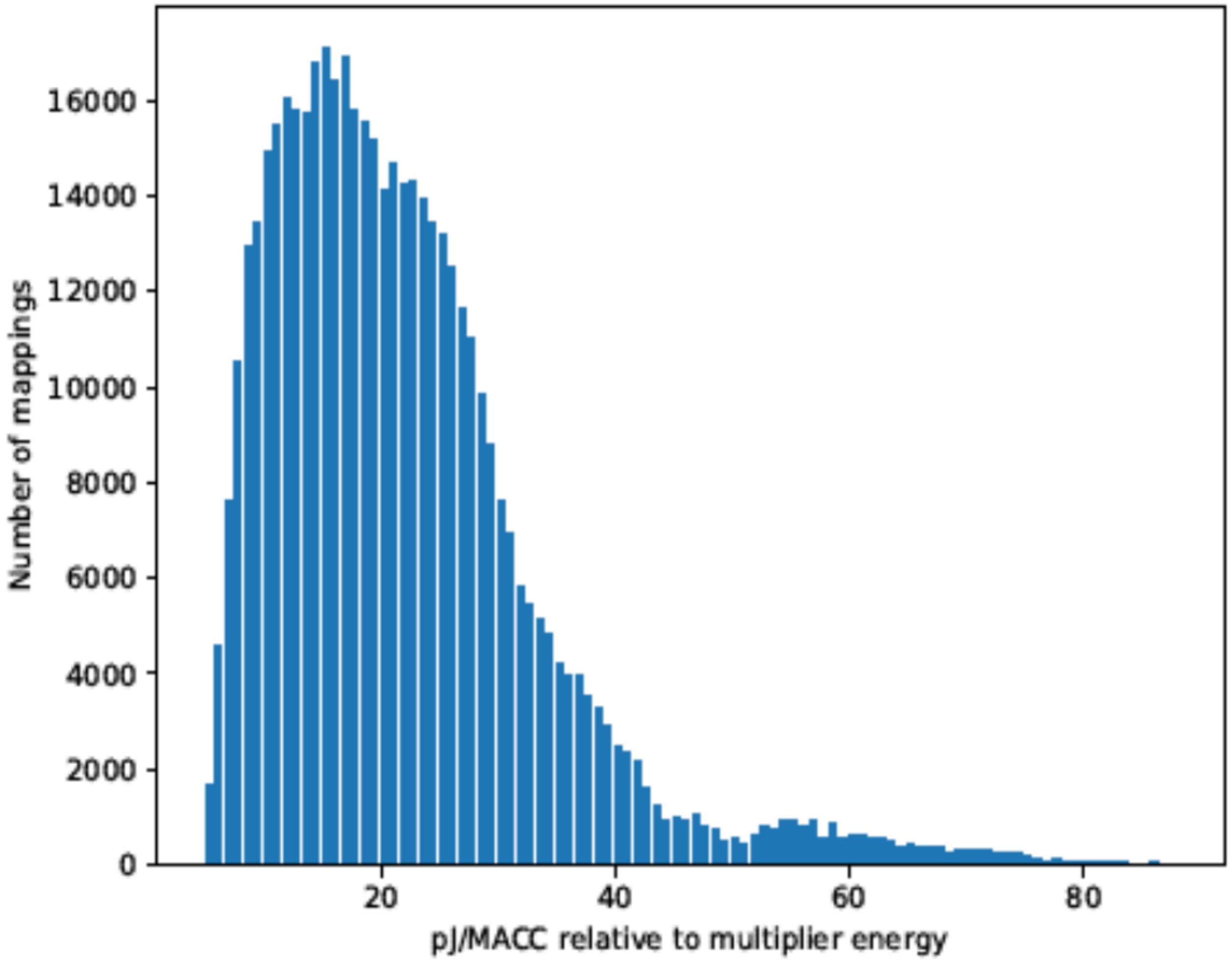
L1 Mapping

```
for s0= [0, 3, 1):
  for x0= [0, 3, 1):
    x = x2+x1+x0
    s = s2+s1+s0
    PartialSum[x][s] = Weight[s]*Input[x+s]
    Output[x] += PartialSum[x][s]
```



Why Mappings Matter?

VGG conv3 2 Layer. Source: Timeloop



480,000 mappings shown

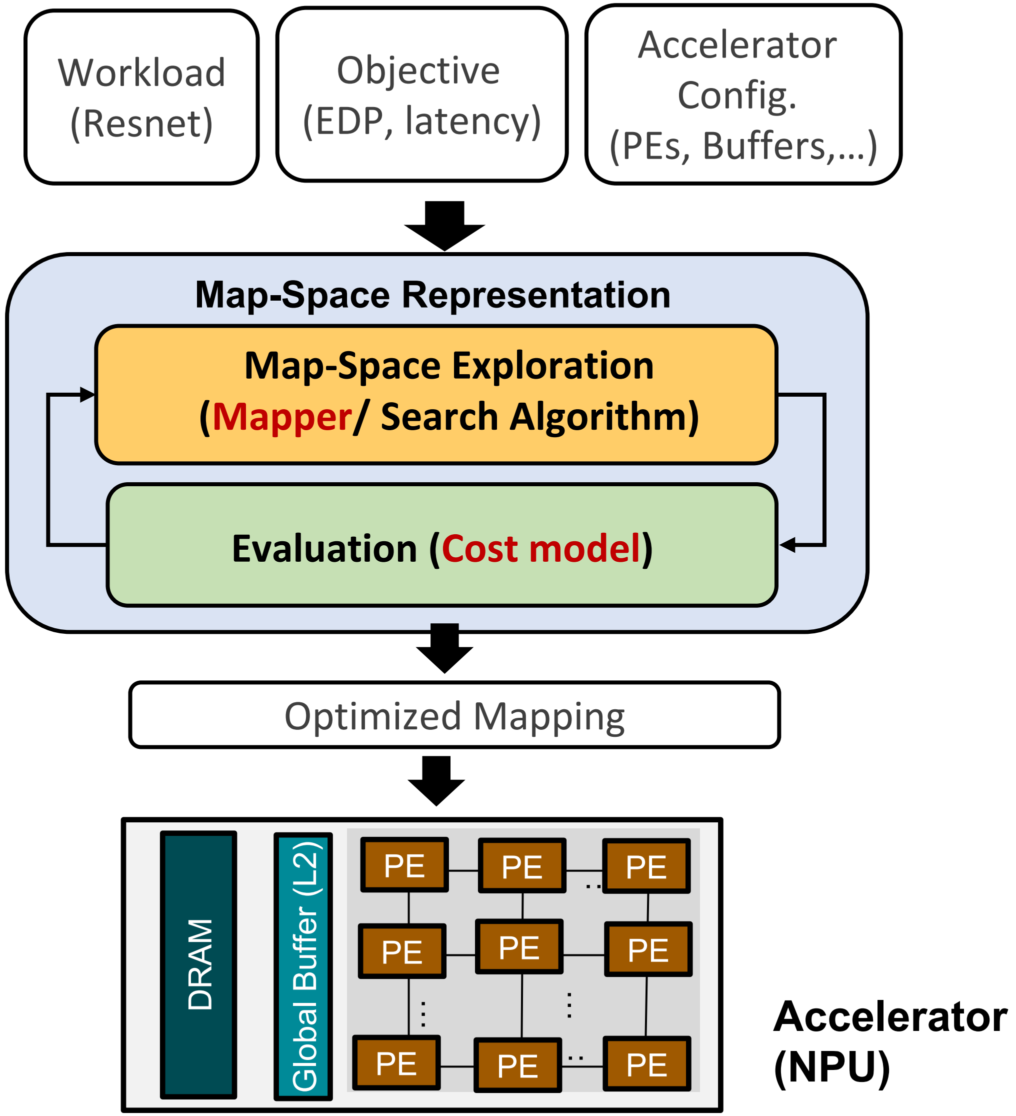
Spread: 19x in energy efficiency

Only 1 is optimal, 9 others within 1%

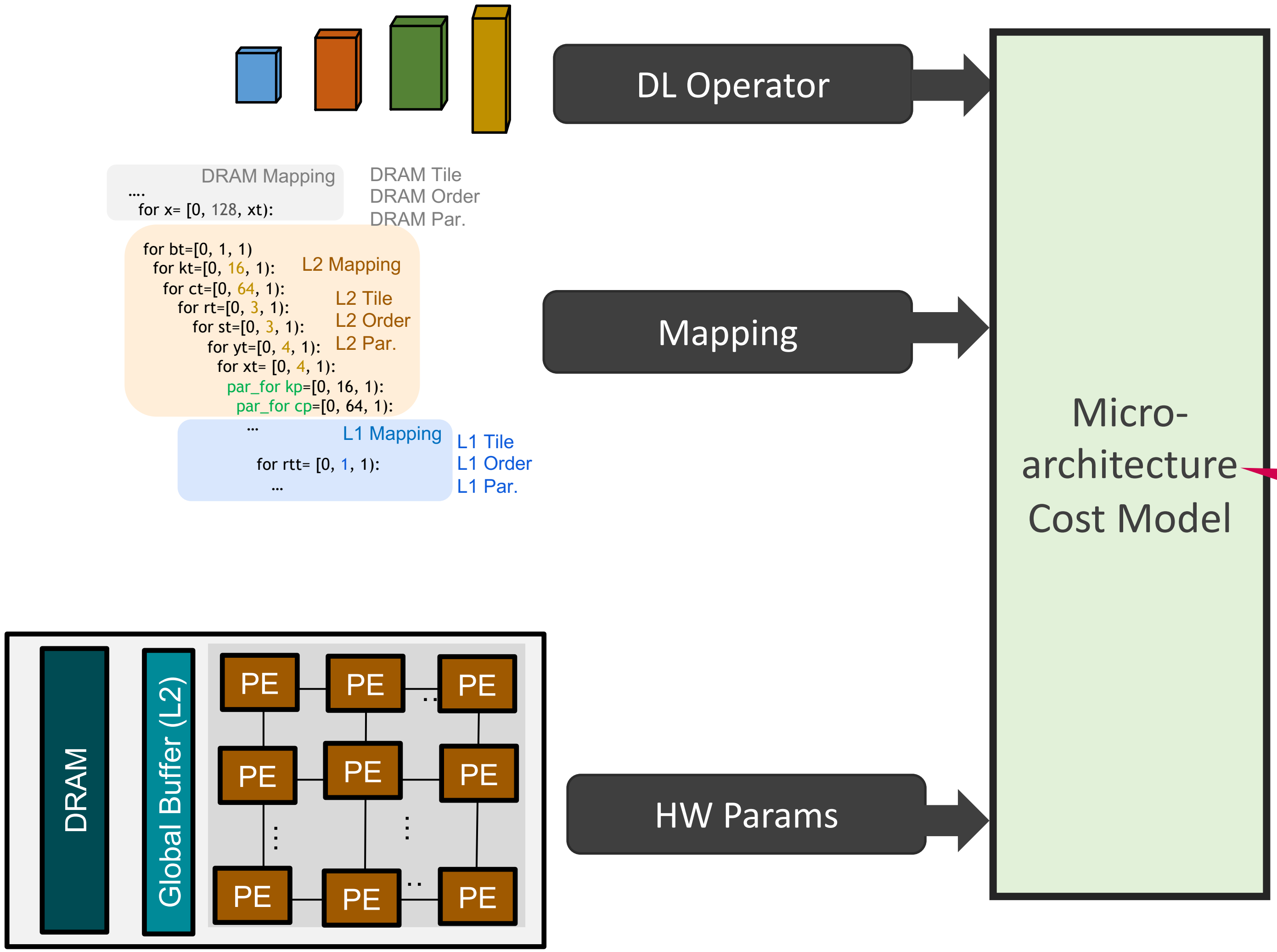
6,582 mappings have min. DRAM accesses but vary 11x in energy efficiency

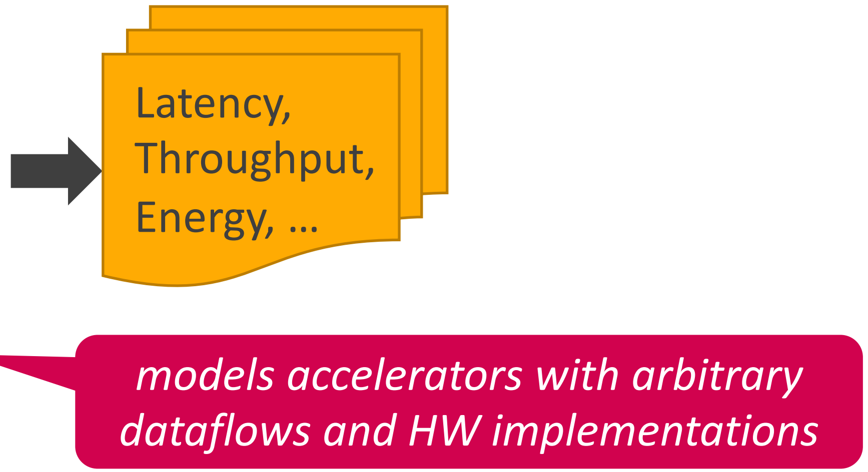
Map Space Exploration (MSE) is crucial!

Map Space Exploration (MSE)



Cost Model





Popular *Analytical* Cost Models

Timeloop (Parashar et al, ISPASS 2019)
MAESTRO (Kwon et al, MICRO 2020)
SCALE-sim (Samajdar et al, ISPASS 2020)
Interstellar (Yaang et al, ASPLOS 2020)
ZigZag (Mei et al., IEEE Trans. Computers 202
…

Challenge(s) with MSE

Immense design space: ~O(10^24) per DNN layer on a 2-level memory hierarchy accelerator

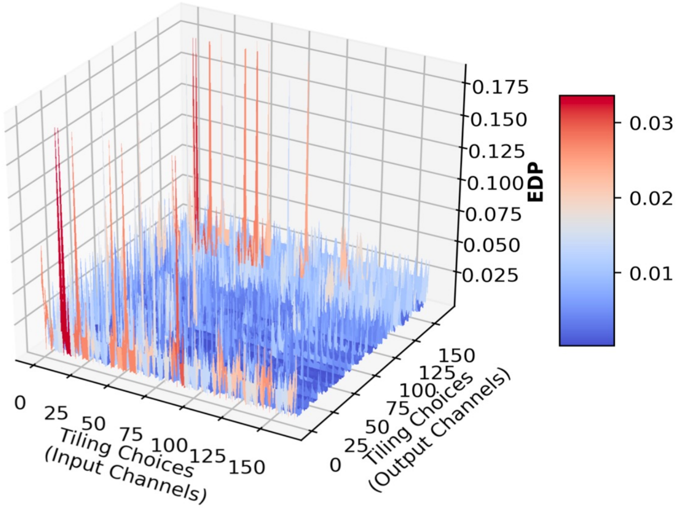
- Not amenable to exhaustive search
  - Need ~ 10^{25} years assuming 1 msec per sample
    - 3x longer than the age of Earth!

    → Sample efficiency is crucial

Sample efficiency: *The performance improvement under limited number of samples.*

E.g., VGG-16 (K=64, C=64, X=224, Y=224, R=3, S=3)
Tile Search Space = 64x64x224x224x3x3 = 10^9
Parallelism Search Space = 6
Loop Order Search Space = 6!
Total = 10^{12} (per tiling level)

- **Discrete design space (X) and non-convex performance (i.e., reward) space (f(X))**
  - Not directly amenable to gradient-descent



Mindmapping [ASPLOS'21]

MSE is an active area of research

Work	Search Technique
AutoTVM (OSDI’18)	Simulated Anneal
Timeloop (ISPASS’19)	Pruned Search
dMazeRunner (TECS’19)	Pruned Search
Simba (MICRO’19)	Random Search
FlexTensor (ASPLOS’20)	RL
CoSA (ISCA’21)	Mixed-Integer Programming
MindMapping (ASPLOS’21)	Gradient-based
HASCO (ISCA’21)	Bayesian Opt + RL
Gamma (ICCAD’20)	Specialized GA

Heuristics

Learning-based methods

MSE is solved? What’s next?

Goal of this work

Demystifying MSE

Challenge 1: Dozens of mappers are proposed. It is hard to compare them systematically

Challenge 2: It is hard to explain and understand why and how the mapper works

Improving MSE

Challenge 1: The run time of MSE become bottlenecks for large DNN models with complex tensor shape

Challenge 2: DNN workloads has sparsity, how MSE cope with sparsity is still an open question

Outline

- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

Goal of this work

Demystifying MSE

Challenge 1: Dozens of mappers are proposed. It is hard to compare them systematically

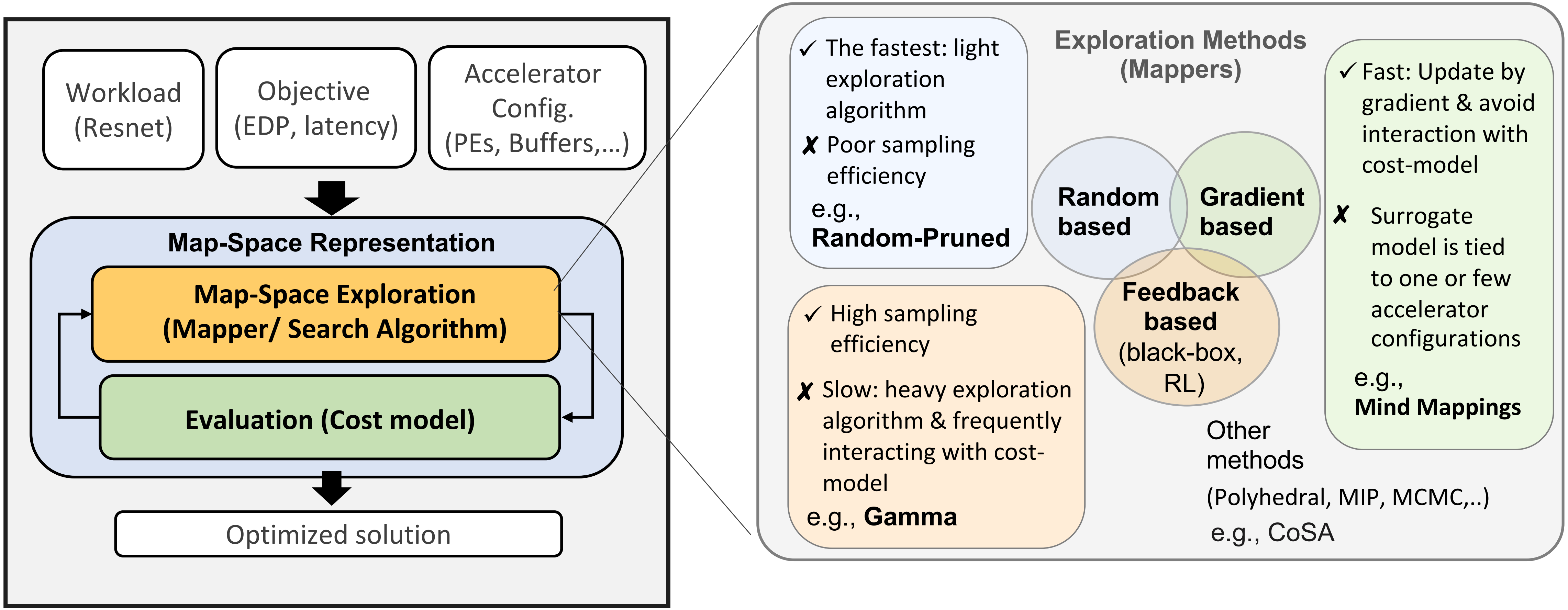
Challenge 2: It is hard to explain and understand why and how the mapper works

Improving MSE

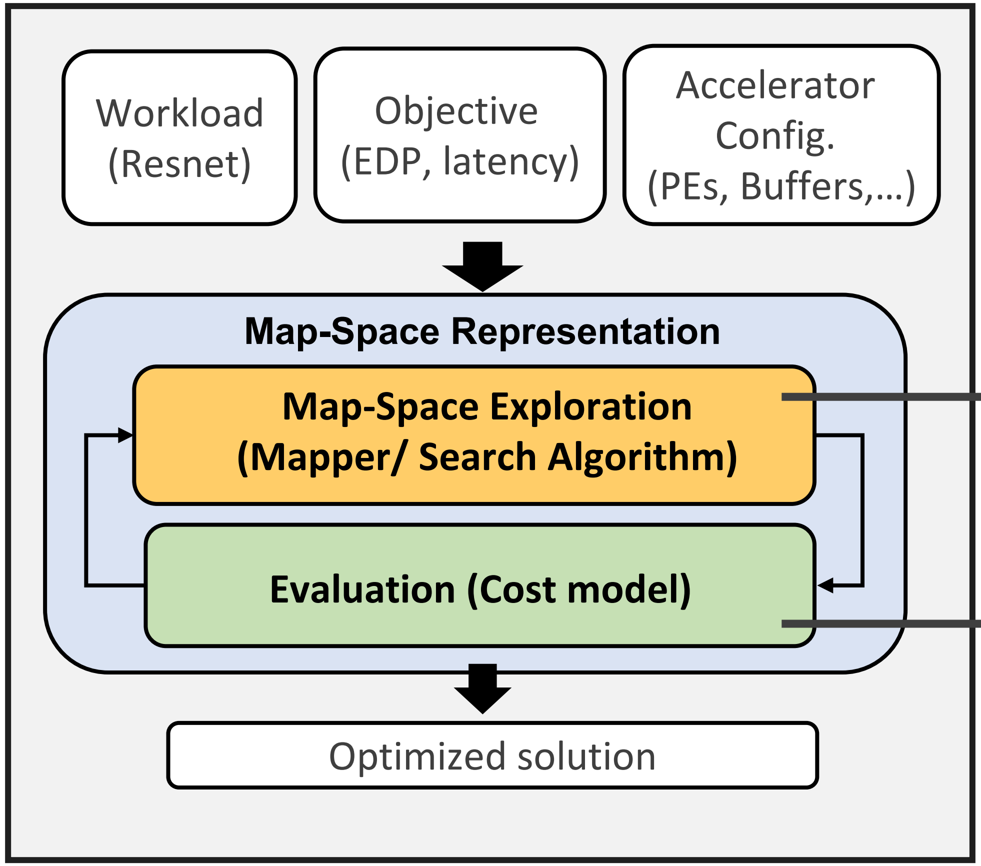
Challenge 1: The run time of MSE become bottlenecks for large DNN models with complex tensor shape

Challenge 2: DNN workloads has sparsity, how MSE cope with sparsity is still an open question

Categorizing Existing DNN Mappers



Target Mappers



We select the SOTA of each category and compare their behaviors

Random-pruned[1] (Timeloop's native mapper)
MindMappings[2] (Trained surrogate model)
Gamma[3] (Genetic Algorithm with custom operators)

Timeloop[4]

1, 4 https://github.com/nvlabs/timeloop
2 https://github.com/kartik-hegde/mindMappings
3 : https://github.com/maestro-project/gamma-timeloop

Evaluation Setup

Accelerator Configuration

- **Accel-A:** A seen and trained accelerator configuration for the surrogate model in gradient base method
- **Accel-B:** A new unseen accelerator configuration

	Accelerator Configuration
Accel A	512 KB shared buffer, 64 KB private buffer per PE, 256 PEs, 1 ALUs per PE
Accel B	64 KB shared buffer, 256 B private buffer per PE, 256 PEs, 4 ALUs per PE

DNN Workloads

A workload is a DNN layer

- We use some selected workloads for the following experiments

CONV2D Workload	(B,K,C,Y,X,R,S)
Resnet Conv_3	(16,128,128,28,28,3,3)
Resnet Conv_4	(16,256,256,14,14,3,3)
Inception Conv_2	(16,192,192,27,27,5,5)

CONV2D Notation	
B	Batch size
K	Output channel
C	Input channel
Y	Input Height
X	Input Width
R	Weight Height
S	Weight Width

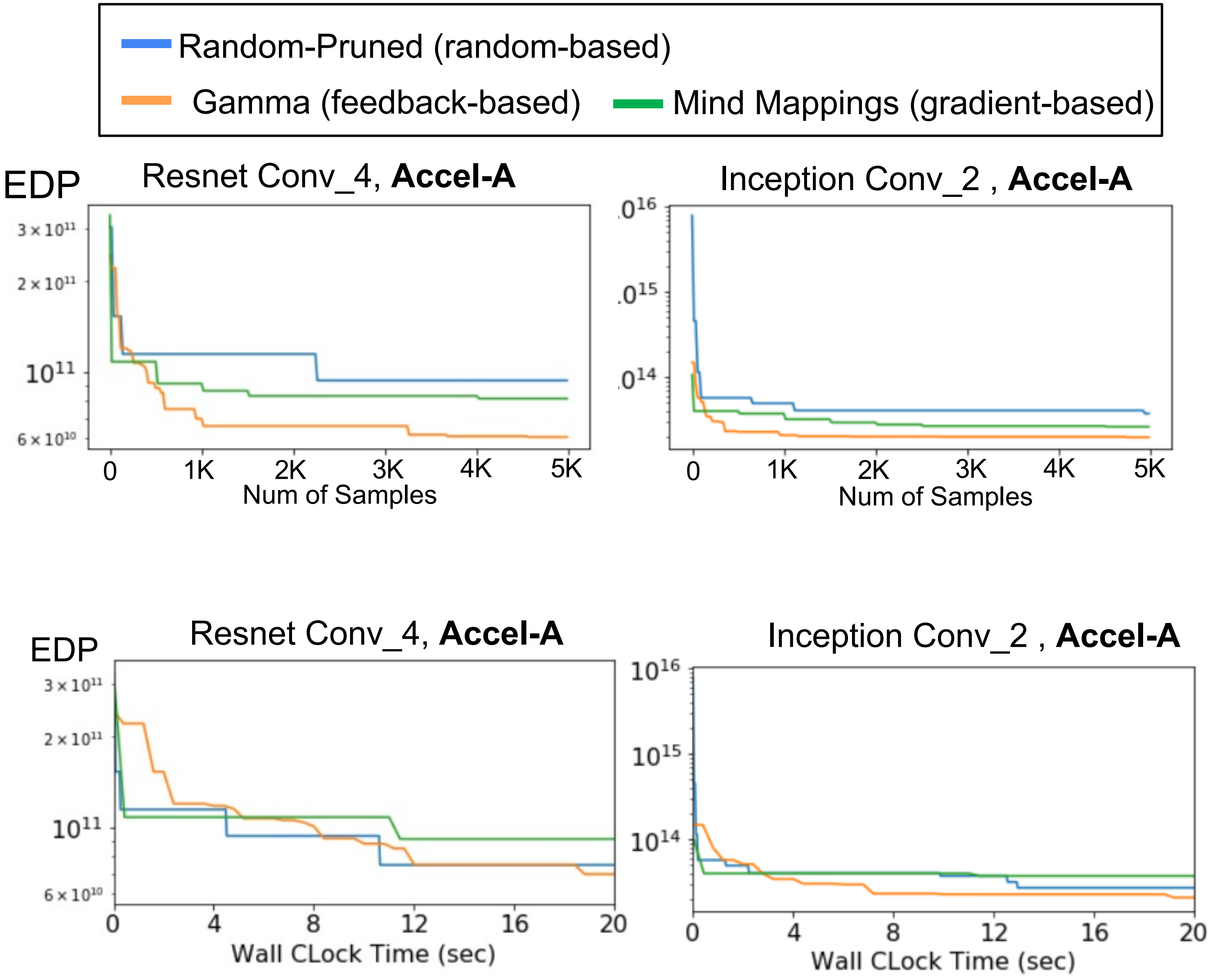
GEMM Workload	(B,M,K,N)
Bert-Large KQV	(16,1024,1024,512)
Bert-Large Attn	(16,512,1024,512)
Bert-Large FF	(16,4096,1024,512)

GEMM Notation	
M	Matrix-A Rows
N	Matrix-B Rows
K	Contraction sizes

Objective

- We use Energy-Delay-Product (**EDP**) as objective
- Other objectives can also be used

Comparisons of Mapper Algorithms



***Accel-A:** Accelerator configuration MindMappings is trained on*

Sampling Efficiency

- Gradient-based > Feedback-based > Random-Pruned
  - Direct gradient access faster than collecting more data samples to learn

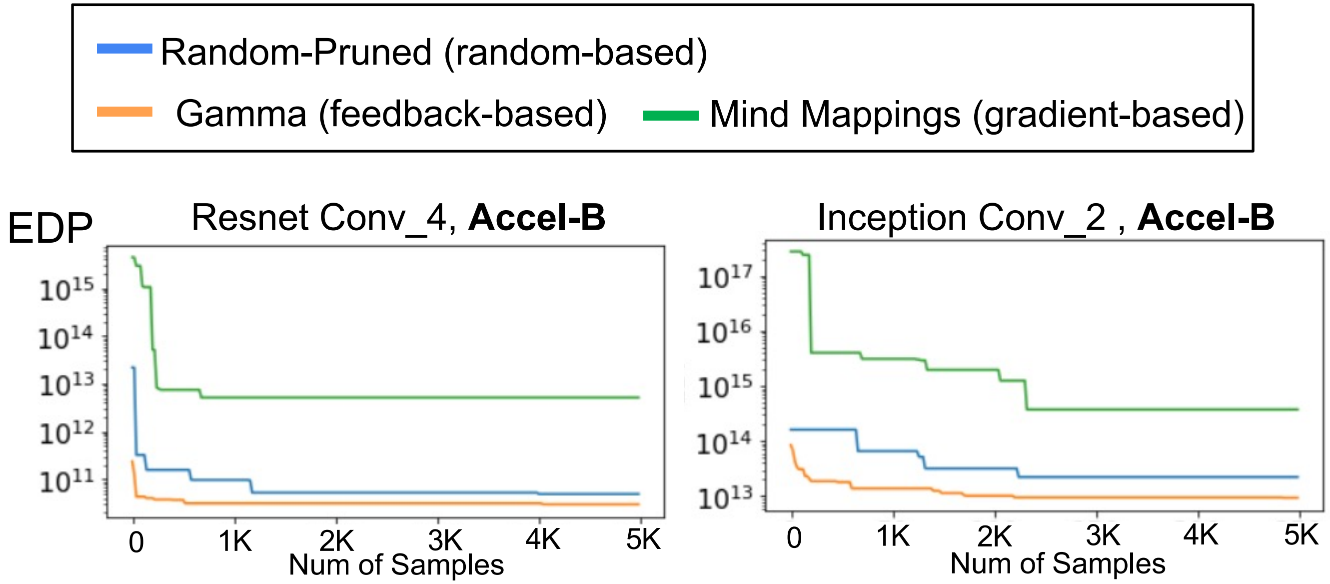
Optimality

- Feedback-based > Gradient-based > Random-Pruned
  - Gradient-based can get stuck in local minima

Wall-clock Search Time

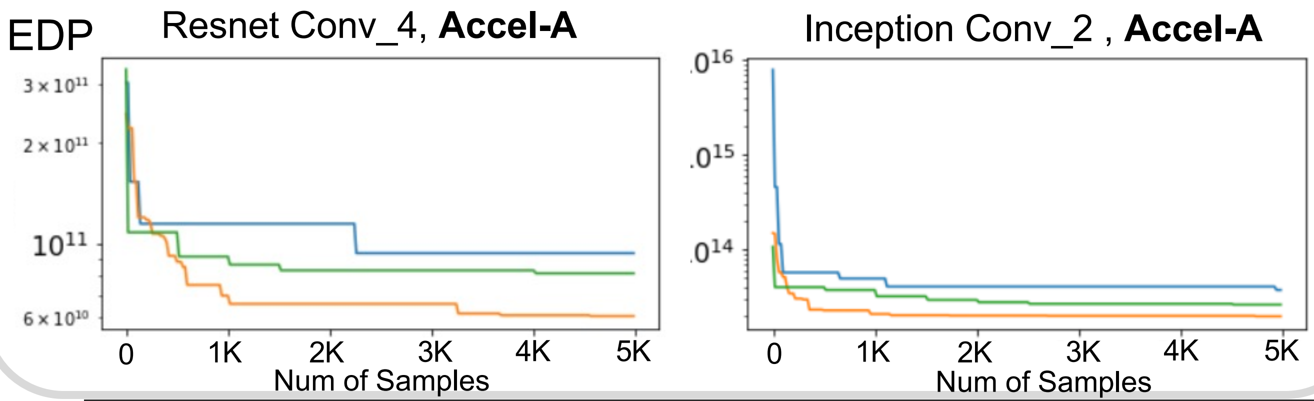
- Random-based > Gradient-based > Feedback-based
  - Runtime cost per sample is about 10x higher for Learning methods than Random-based
  - *When time constraint is strictly tight, Learning method cannot yet gather adequate data to improve their sampling function*

Comparisons of Mapper Algorithms



__Accel-A:__ Accelerator configuration MindMappings is trained on
__Accel-B :__ An unseen accelerator configurations

Recap of performance on Accel-A



- Gradient-based method is tied to one or few seen accelerator configurations in the training dataset
- Gradient-based method can recover the performance by
  - Collecting new data (1M-5M data points for quality result)
  - Re-train the surrogate model

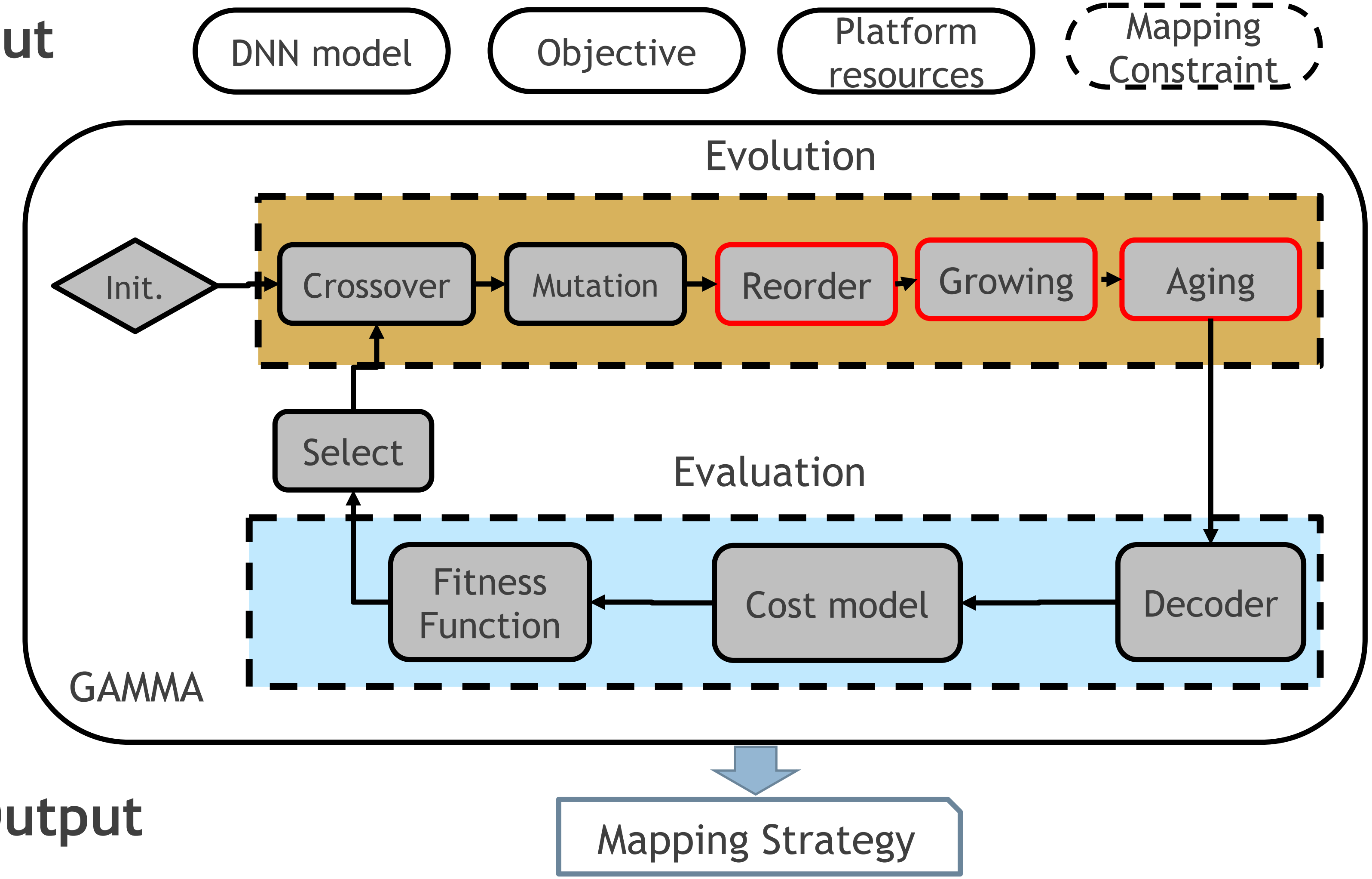
Deeper Look at Gamma

S-C. Kao and T. Krishna, **"GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm",** *In Proc of the IEEE/ACM International Conference on Computer-Aided Design* ***(ICCAD)****, Nov 2020*

Features

- Mapping represented as "genes"
- Custom operators
    - crossover and mutation to maintain valid mappings
    - 3 additional operators

Input



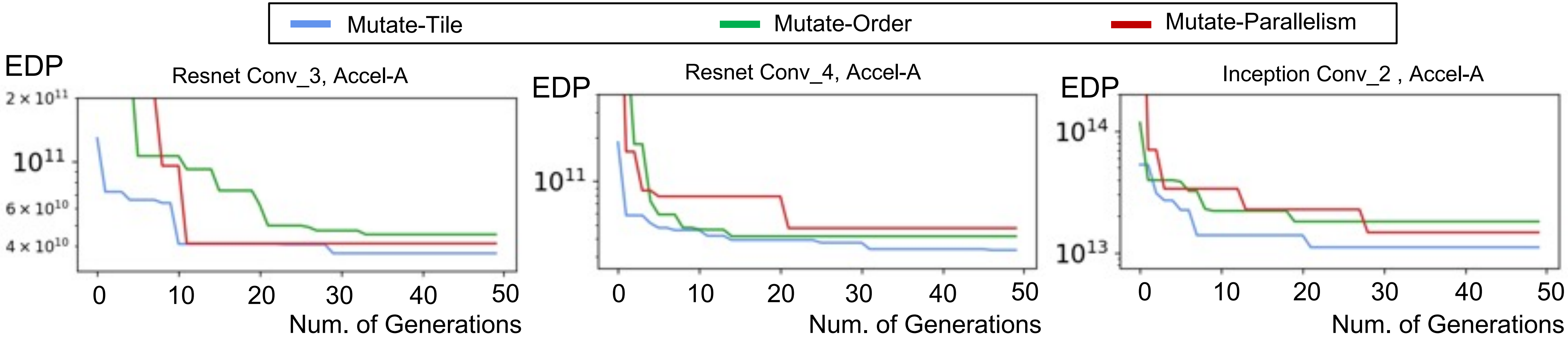
Output

P	C	R	S	X	K	Y	P	X	Y	S	R	K	C
K	20	3	3	15	64	10	C	7	5	1	1	8	1

Sensitivity Analysis of Mapper's Operators

*We use **Gamma** for this analysis*

- Gamma has separate mutation operator for different mapping axes
- We use only single mutation operator for exploration

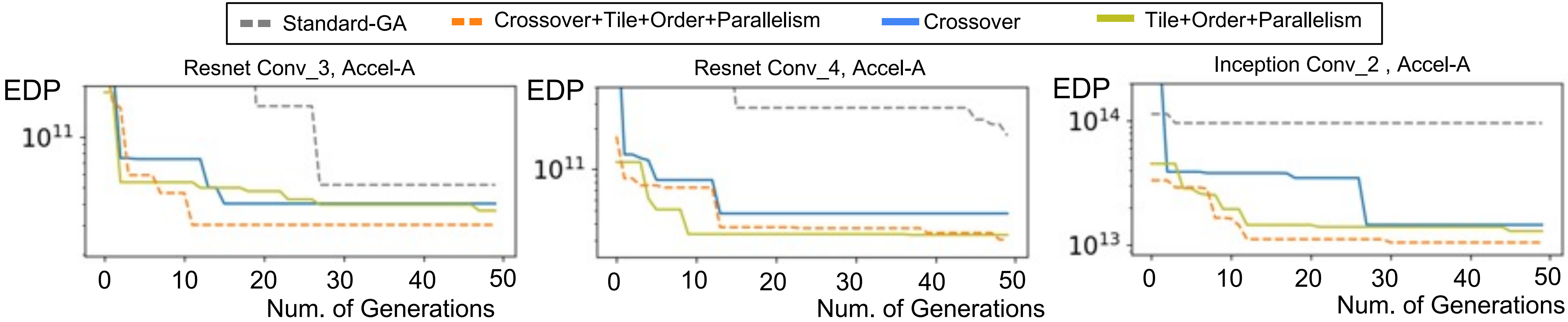


- Mutation-Tile is the most impactful
  - Providing tile size flexibility crucial for accelerator performance
- Many order + parallelism permutations lead to similar latency or energy.
  - Various loop orders can be placed into large "stationarity" buckets (such as weight/ input/ output/ row)

Sensitivity Analysis of Mapper’s Operators

*We use **Gamma** for this analysis*

- GA is special for its crossover operator
- We perform sensitivity analysis on crossover operator



- standard-GA is not efficient due to lack of specialized mutation and crossover operators
- Blending two high-performance mappings (*crossover*) can effectively create another high-performance mapping (from MutateT+O+P to Crossover+MutateT+O+P)
- Crossover by itself is not as efficient without specialized mutations

Outline

- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

Goal of this work

Demystifying MSE

Challenge 1: Dozens of mappers are proposed. It is hard to compare them systematically

Challenge 2: It is hard to explain and understand why and how the mapper works

Improving MSE

Challenge 1: The run time of MSE become bottlenecks for large DNN models with complex tensor shape

Challenge 2: DNN workloads has sparsity, how MSE cope with sparsity is still an open question

Warm-start: techniques for Improving MSE Speed

Observation:

- DNN workloads have similarity
- Solution for mapping axes on Order and Parallelism can often be re-used

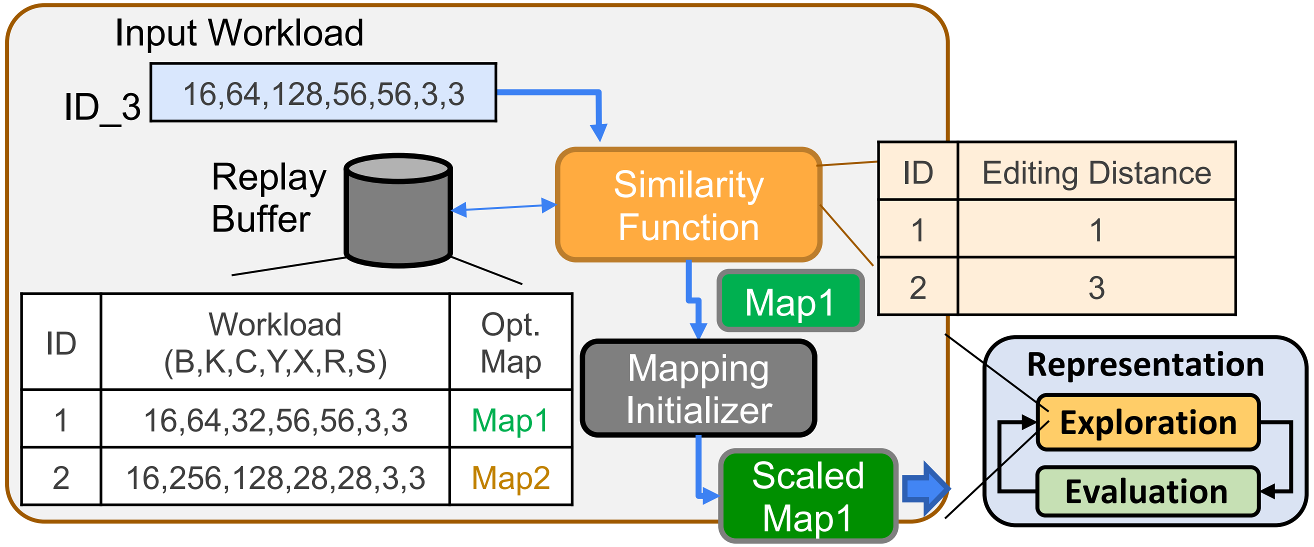
Warm-start

Initialize mapping search by the previous mapping solutions

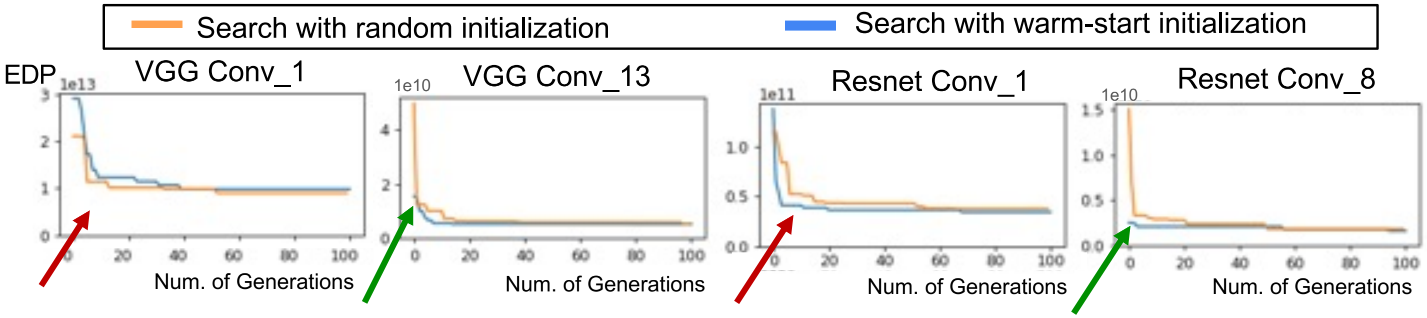
- Store found solutions in the replay buffer

Warm-start initialization steps

1. Compare workload similarity in the replay buffer
2. Inherent Order, Parallelism axes
3. Scale tile axis to match the tensor shape of the new workload
4. Run the optimization loop as usual



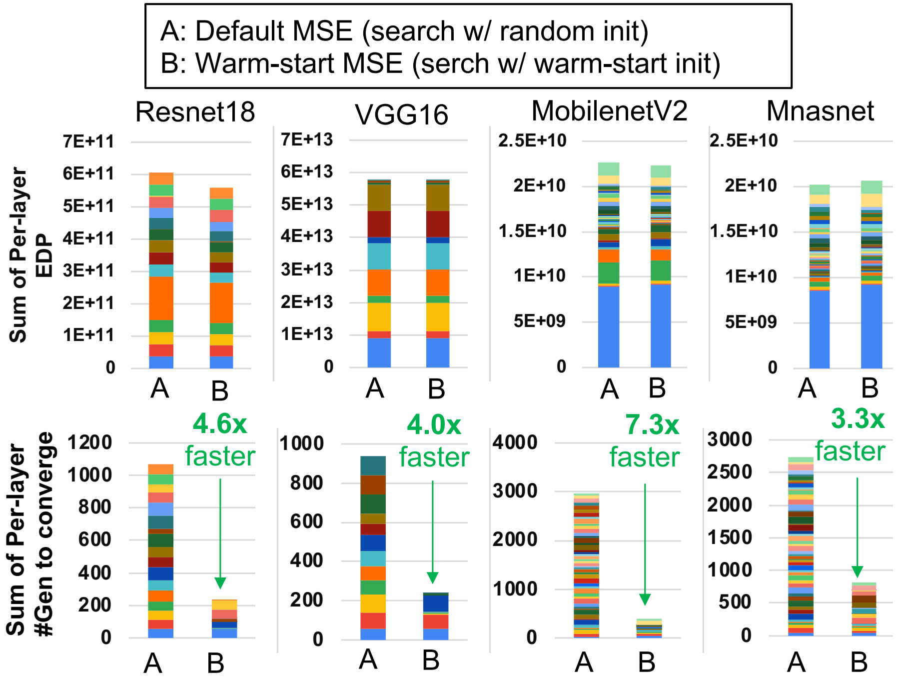
The Effect of Warm-start



First layer of DNN → No difference

Later layers of DNN → can leverage previous solutions and start at better points

Configuration: Accel-B



- Warm-start can help start at better points and converge faster
- Warm-start can reduce the time-to-converge by 3.3x-7.3x for searching entire model

Goal of this work

Demystifying MSE

Challenge 1: Dozens of mappers are proposed. It is hard to compare them systematically

Challenge 2: It is hard to explain and understand why and how the mapper works

Improving MSE

Challenge 1: The run time of MSE become bottlenecks for large DNN models with complex tensor shape

Challenge 2: DNN workloads has sparsity, how MSE cope with sparsity is still an open question

Does Sparsity Matter for MSE?

Motivation:
- DNN model weights are often trained to be sparse
- Does an optimal mapping for dense workload still perform well in sparse one?

Experiment Methodology:
- We find optimal mapping for workloads with different sparsity
- We evaluate the found optimal mapping one across different sparsity levels
- Green-text represents best of each row

Configuration: Accel-B

- Green-text overlaps with blue-cell (optimized mapping for specific density level)
  - → MSE needs to consider sparsity to pursue best performance
- Weight sparsity can be supported by simple extension of the current MSE framework, since weight sparsity is often fixed after model is trained.
  - But what about activation sparsity?

		EDP (cycles uJ)			
		Weight Density of the Workload			
Test the found mapping across different density	Density	1.0	0.5	0.1	0.01
	Density	Resnet Conv_3			
	1.0	3.7E+10	3.9E+10	5.8E+10	1.6E+12
	0.5	1.0E+10	4.9E+09	9.1E+09	3.9E+11
	0.1	8.0E+08	6.6E+07	6.4E+07	8.3E+08
	0.01	5.0E+07	3.1E+04	4.8E+04	1.6E+04
	Density	Resnet Conv_4			
	1.0	3.1E+10	3.6E+10	1.0E+11	4.3E+11
	0.5	8.3E+09	4.9E+09	1.4E+10	9.6E+10
	0.1	5.5E+08	9.1E+07	2.3E+07	3.7E+08
	0.01	3.0E+07	7.0E+05	6.4E+03	5.4E+03
	Density	Inception Conv_2			
	1.0	1.1E+13	1.3E+13	1.5E+13	5.9E+14
	0.5	3.4E+12	2.0E+12	2.3E+12	1.5E+14
	0.1	3.5E+11	1.3E+10	5.1E+09	4.0E+10
	0.01	3.3E+09	9.4E+06	3.3E+06	6.2E+05

Sparsity-aware: Technique to Support Dynamic Sparsity

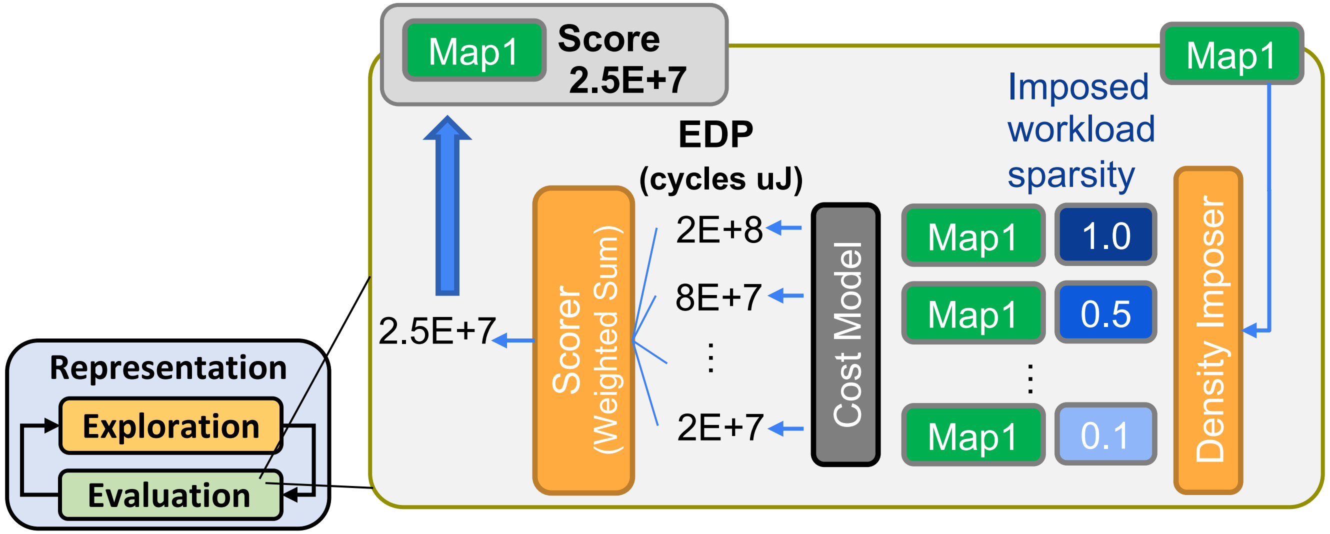
Motivation:
- Activation sparsity is dynamic
- Fresh searches for new mapping for each input-activation is not practical

Sparsity-aware
- Learn a mapping that generalizes across different sparsity levels

Sparsity-aware Methods

At evaluation phase

1. We ignore the workload sparsity and impose different pre-defined sparsity to the workloads
   - E.g., 1.0, 0.5, 0.1 → 3 different sparsity
2. Scores the mapping by the weighted sum of the performance of this workload across different assumed sparsity
3. The mappings will be ranked and selected by the scores



The Effect of Sparsity-aware for Activation Sparsity

Input-activation is sparse

Experiment Methodology:

- Sparsity-aware is trained (searched) under the assumption of 1.0, 0.8, 0.5, 0.2, 0.1, five sparsity levels
- Static-density only optimized for specific density
- We evaluate the found optimal mapping one across different sparsity 1.0 – 0.05
- Green-text represents best of each row

- **[Static density]** Green-text overlaps with blue-cell (optimized mapping for specific sparsity level)
  - Different sparsity levels do require different mapping
- **[Static-density]** Mapping found for specific sparsity level perform poorly for other sparsity levels
- **[Sparsity-aware]** The found general mapping can
  - Perform comparably to optimal mappings optimized for different sparsity levels
  - Perform relatively well across a range of sparsity (1.0 – 0.05)

Workload Density	EDP (Energy uJ) Sparsity-aware	Static density 1.0	Static density 0.5	Static density 0.1
Resnet Conv_3, Accel-B				
1.0	2.40E+13	2.39E+13	2.41E+13	2.46E+13
0.9	1.75E+13	1.94E+13	1.76E+13	1.79E+13
0.8	1.23E+13	1.54E+13	1.24E+13	1.26E+13
0.7	8.26E+12	1.18E+13	8.30E+12	8.46E+12
0.6	5.21E+12	8.69E+12	5.24E+12	5.34E+12
0.5	3.02E+12	6.06E+12	3.02E+12	3.10E+12
0.4	1.55E+12	3.90E+12	1.56E+12	1.59E+12
0.3	6.59E+11	2.21E+12	6.63E+11	6.77E+11
0.2	1.98E+11	1.00E+12	1.99E+11	2.04E+11
0.1	4.78E+10	2.65E+11	4.81E+10	4.78E+10
0.05	1.28E+10	7.34E+10	1.29E+10	2.67E+10
Inception Conv_2, Accel-B				
1.0	7.77E+15	7.77E+15	7.93E+15	7.83E+15
0.9	5.67E+15	6.33E+15	5.79E+15	5.71E+15
0.8	3.99E+15	5.00E+15	4.08E+15	4.02E+15
0.7	2.67E+15	3.84E+15	2.74E+15	2.69E+15
0.6	1.69E+15	2.82E+15	1.73E+15	1.70E+15
0.5	9.78E+14	1.97E+15	9.78E+14	9.83E+14
0.4	5.02E+14	1.26E+15	5.21E+14	5.05E+14
0.3	2.13E+14	7.16E+14	2.23E+14	2.14E+14
0.2	6.39E+13	3.22E+14	8.64E+13	6.38E+13
0.1	1.55E+13	8.37E+13	4.49E+13	1.53E+13
0.05	4.12E+12	2.25E+13	2.53E+13	3.98E+12

Outline

- Background on NPUs
- Map-Space and Map-Space Exploration
- Quantitative Comparison of Mappers
- Improving Map-Space Exploration
- Lessons Learnt

Conclusion

Thank you!

- **Motivation and Problem Statement**
  - MSE is a key component of DNN accelerator design/deployment
  - It is computationally challenging and has spawned research in heuristics and learning-based methods, each claiming to be better than the other
  - We characterize three classes of DNN accelerator mappers: Timeloop's native Random-Pruned, MindMappings (gradient-based) and Gamma (feedback-based)
- **Key Findings**
  - Learning-based mappers have higher sampling efficiency than random-pruned by constantly improving their sampling function. However, they have the higher wall-clock time to acquire one sample.
  - Tile is the most critical mapping axis to explore.
  - Creating new mappings from high-performance mappings (i.e., *crossover*) improves sample-efficiency
  - MSE needs to consider sparsity
- **Proposed Optimizations**
  - **Warm-Start:** Leveraging DNN workload similarity can bootstrap Mapper from better points
  - **Sparsity-aware:** To tackle dynamic sparsity in activation, it is possible to find a generic mapping works generally well across a range of sparsity level

Code available: https://github.com/maestro-project/gamma-timeloop