

清華大學 交叉信息研究院
Institute for Interdisciplinary Information Sciences, Tsinghua University

上海人工智能实验室
Shanghai Artificial Intelligence Laboratory

上海期智研究院
SHANGHAI QI ZHI INSTITUTE

KAPLA: Scalable NN Accelerator Dataflow Design Space Structuring and Fast Exploring

Zhiyao Li and **Mingyu Gao**

Tsinghua University

Shanghai Artificial Intelligence Lab

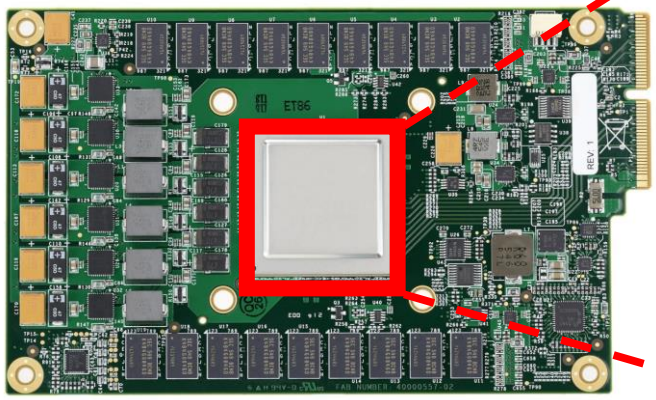
Shanghai Qi Zhi Institute


清華大學
Tsinghua University

ASP-DAC 2025

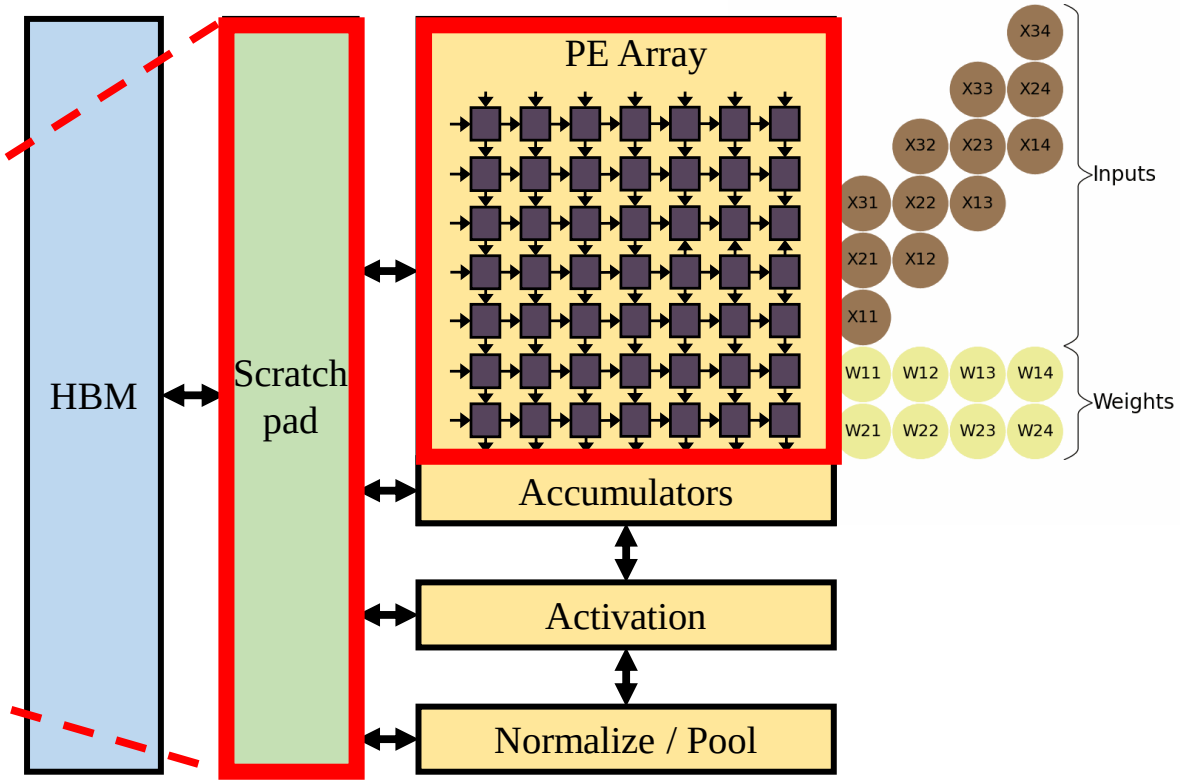
Background & Motivation

- **Domain-specific accelerators (DSAs) for neural networks (NNs)**
  - Abundant parallelism of PEs
  - Localized memory accesses
  - Reduced control overheads



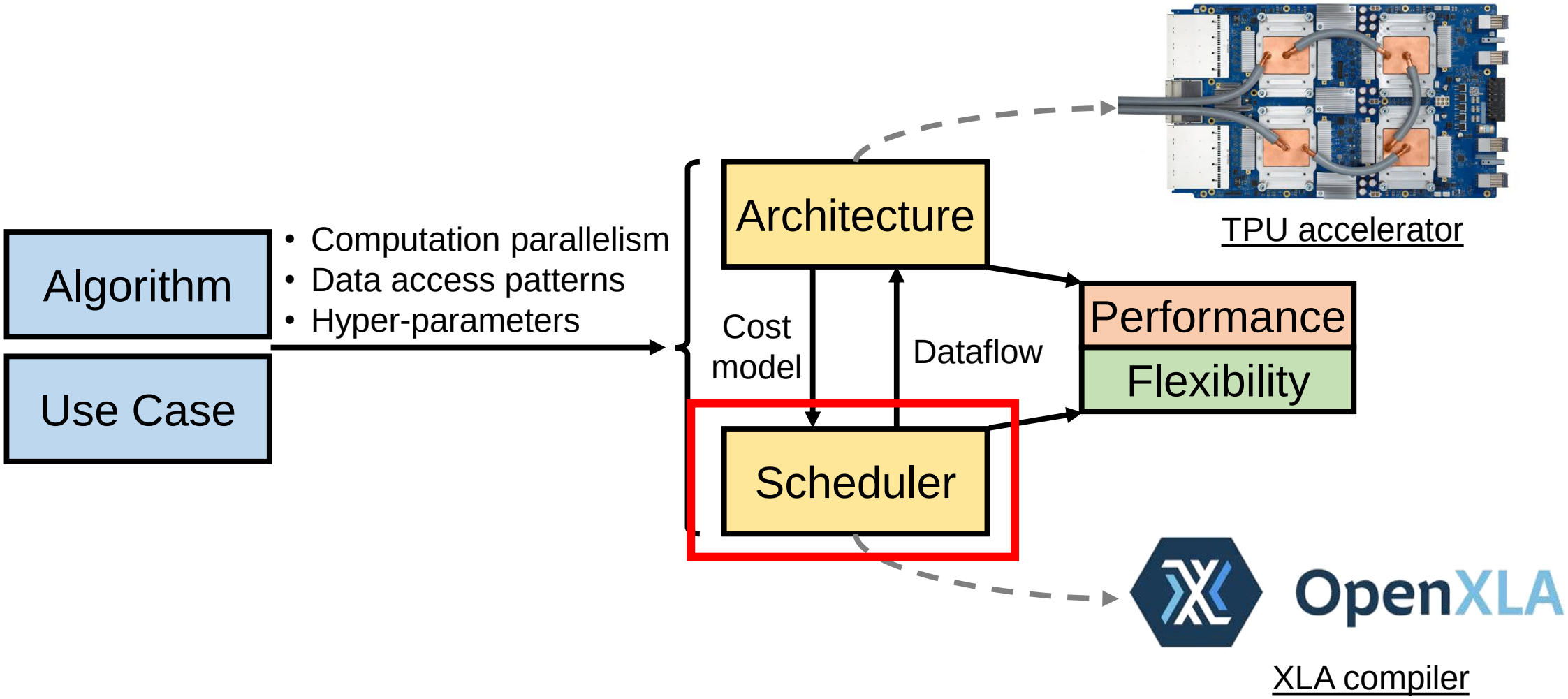
Google TPU v1 [ISCA'17]

70x efficiency



Background & Motivation

- **DSAs' hardware architectures and dataflow schedulers**



A Large Design Space of Dataflow Scheduling

- **Complex hardware structures**
  - Diverse PE array structures: 1D, 2D, adder tree, …
  - Deep hierarchical buffer storage: register file, global buffer, main memory, …
  - Numerous constraints: PE dataflow, buffer capacity, …
- **Various algorithm designs**
  - Complex model topologies
  - 3D/4D tensor dimensions
  - Abundant layer types
- **Flexible use scenarios**
  - Offline: traditional compilers
  - Online: MLaaS, NAS, …

Takeaway message:

Time-consuming and frequently-occurring scheduling can no longer be ignored!

Hierarchical NN Dataflow Taxonomy

Segment slicing

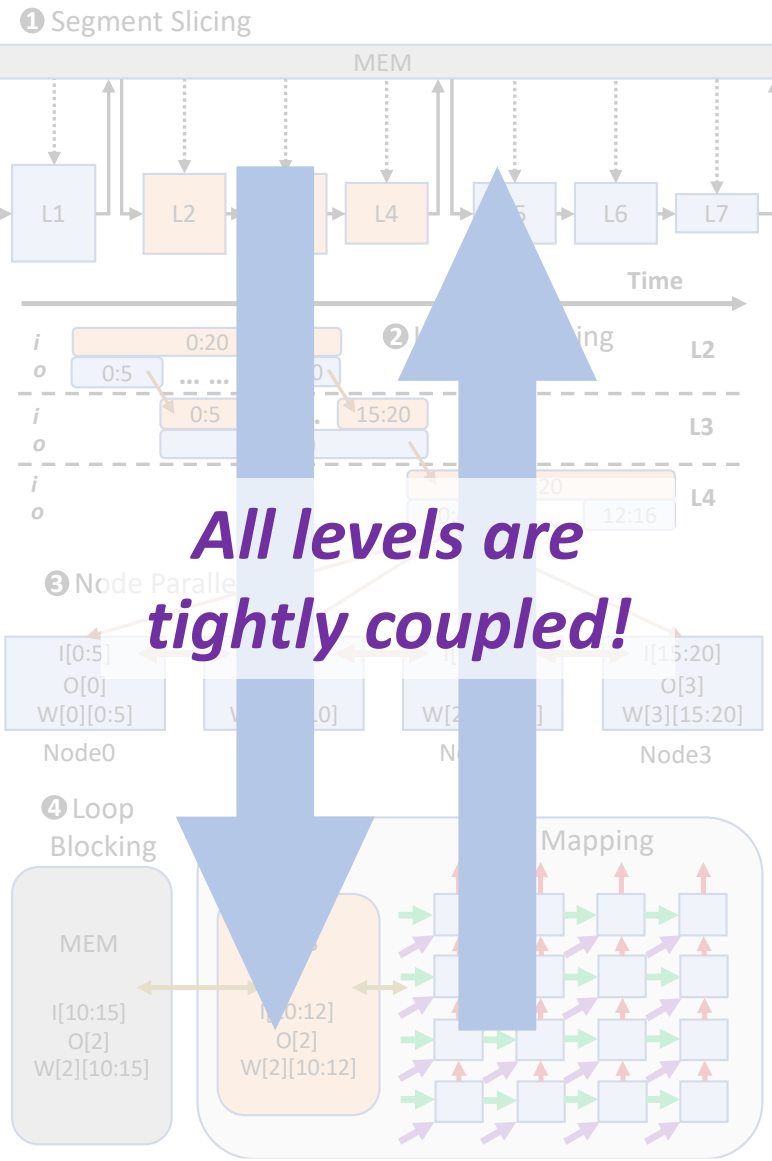
- Model DAG is sliced into multiple segments where each contains one or more layers
- Each segment is scheduled on hardware one after one

Node parallelism

- Each layer is parallelized across multiple hardware processing nodes
- Data parallelism, tensor parallelism, …

Loop blocking

- For each (sub)layer in one hardware node, optimize off-chip data accesses using nested loop transformation techniques, including loop blocking, reordering, unrolling, etc.



Layer pipelining

- Layers in a segment are processed in a pipelined style
- Intermediate data are forwarded on-chip without spilled to off-chip memory

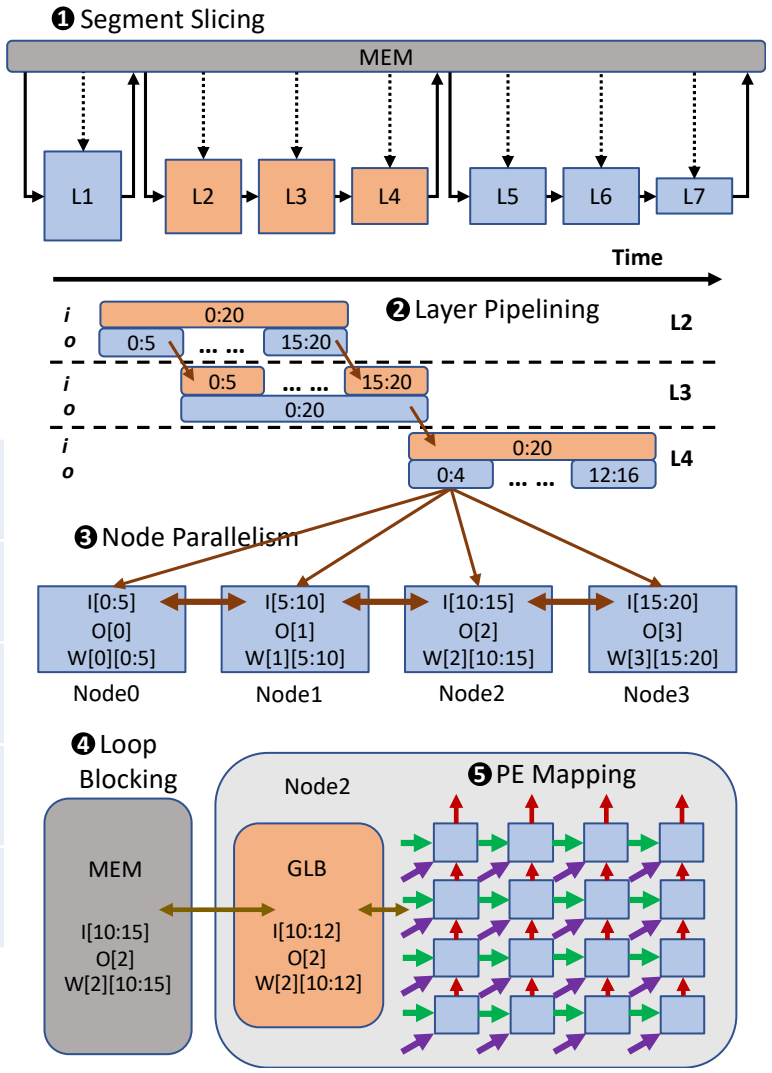
PE mapping

- For each (sub)layer in one hardware node, optimize on-chip dataflow on the PE array, such as systolic, row-stationary, etc.

Prior Dataflow Techniques

- **Prior studies mainly focusing on one level while ignoring other levels**
  - Inaccurate and suboptimal
  - Incompatible or even conflicting

Segment slicing	TANGRAM [ASPLOS'19]
Layer pipelining	NeoCPU [ATC'19], PipeLayer [HPCA'17], ScaleDeep [ISCA'17]
Node parallelism	TETRIS [ASPLOS'17], NeuroCube [ISCA'16]
Loop blocking	Timeloop [ISPASS'19], Interstellar [ASPLOS'20]
PE mapping	TPU [ISCA'17], Eyeriss [ISCA'16], ShiDianNao [ISCA'15]



Prior Scheduling Methods

- **Prior schedulers are either too slow or inaccurate**

Method	Representative	Problems
Exhaustive search	TANGRAM [ASPLOS'19]	Long search time
Random sampling	Timeloop [ISPASS'19]	Low scheduling quality
ML-guided search	AutoTVM [NeurIPS'18]	Long search time; no quality guarantees
Polyhedral models	AKG [PLDI'21]	Restricted search space

Design goal:

A generic, optimized, and fast scheduler is needed!

Our Key Ideas

Key techniques:

- ***Decouple inter-layer from intra-layer***
- ***Effective validity check and efficiency estimation***

Inter-layer pruning and prioritization

Validity check

- Conservative pruning

Efficiency estimation

- Fast cost estimation
- Dynamic-programming-based prioritization

Intra-layer stacking and caching

Validity check

- Tensor sizes from tensor-centric directives

Efficiency estimation

- Maximize temporal reuse through caching
- Maximize spatial parallelism through stacking

Tensor-Centric Dataflow Directives

`tensor(dim=size, ...[, shr])`

⟶ (Sub)tensors in each buffer

`stack(dim+=shift, ..., repl)`

⟶ Spatial parallelism, in multiple buffers at one level

`update(dim+=step, ...)`

⟶ Temporal accesses, across multiple buffer levels

- **Advantages**
  - Tensor as first-class citizen
    - **Fast** validity checking of tensor sizes in buffers
  - Bottom-up dataflow construction
    - **Fast** memory hierarchy traffic calculatation
  - Inter-operator pipeline native
    - **Easy to express** layer pipelining optimizations

Directive examples for CONV and DWCONV layers, with **row-stationary** PE mapping, output + batch **hybrid node parallelization**, and **layer pipelining**.

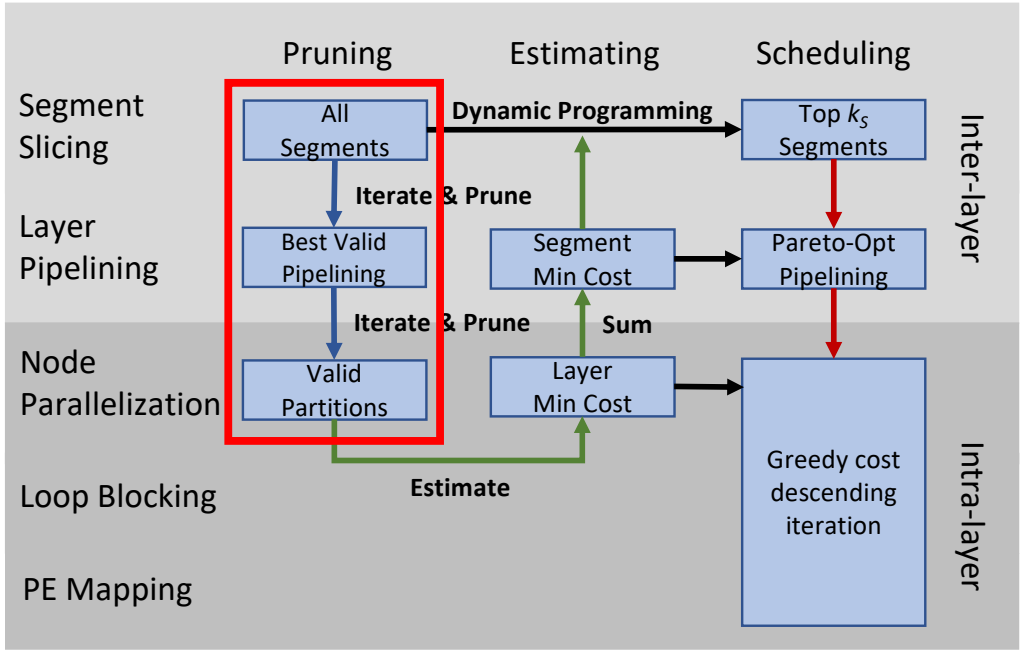
```
CONV:
 REGF:
  tensor{0}(N=1, C=2, Xi=5, Yi=1)
  tensor{w1}(C=2, K=3, R=5, S=1)
  tensor{1}(N=1, K=3, Xo=1, Yo=1)
  stack(Yi+=1, Yo+=1, 8)  % PE columns
  stack(S+=1, Yi+=1, 5)   % PE rows
  update(Xi+=1, Xo+=1)    % 1D conv
  update(Yi+=8, Yo+=8)    % folding
  update(N+=1)
  update(C+=2)
  update(K+=3)
 GBUF:
  tensor{0}(N=4, C=4, Xi=19, Yi=19, shr=4)
  tensor{w1}(C=4, K=6, R=5, S=5)
  tensor{1}(N=4, K=6, Xo=15, Yo=15)
  stack(K+=6, 4)    % output node parallel
  stack(N+=4, 16)   % batch node parallel
  update(C+=4)
  update(K+=24)
  update(N+=64)
DWCONV:
 REGF:
  % ...
 GBUF:
  % DWCONV input is the same as CONV output
  tensor{1}(N=4, C=4, Xi=9, Yi=15)
  tensor{w2}(C=4, R=3, S=3)
  tensor{2}(N=4, C=4, Xo=4, Yo=7)
  stack(C+=4, 6)    % channel node parallel
  stack(N+=4, 16)   % batch node parallel
  stack(Xo+=4, 2)   % output width node parallel
  update(Yo+=7)
  update(C+=24)
  update(N+=64)
```

KAPLA Dataflow Solver

	Validity check	Efficiency estimation
Inter-layer	Conservative pruning	
Intra-layer		

Estimate the minimum required buffer capacity of a layer, with optimistic intra-layer schemes

	Total Schemes	After Pruning	% Pruned
AlexNet	700	2	99.7%
MobileNet	331	42	87.3%
VGGNet	7	1	85.7%
GoogLeNet	560	1	99.8%
ResNet	63	4	93.6%
MLP	784	18	97.7%
LSTM	70	3	95.7%



The overall workflow of KAPLA

KAPLA Dataflow Solver

	Validity check	Efficiency estimation
Inter-layer	Conservative pruning	**Prioritization based on estimated cost**
Intra-layer		

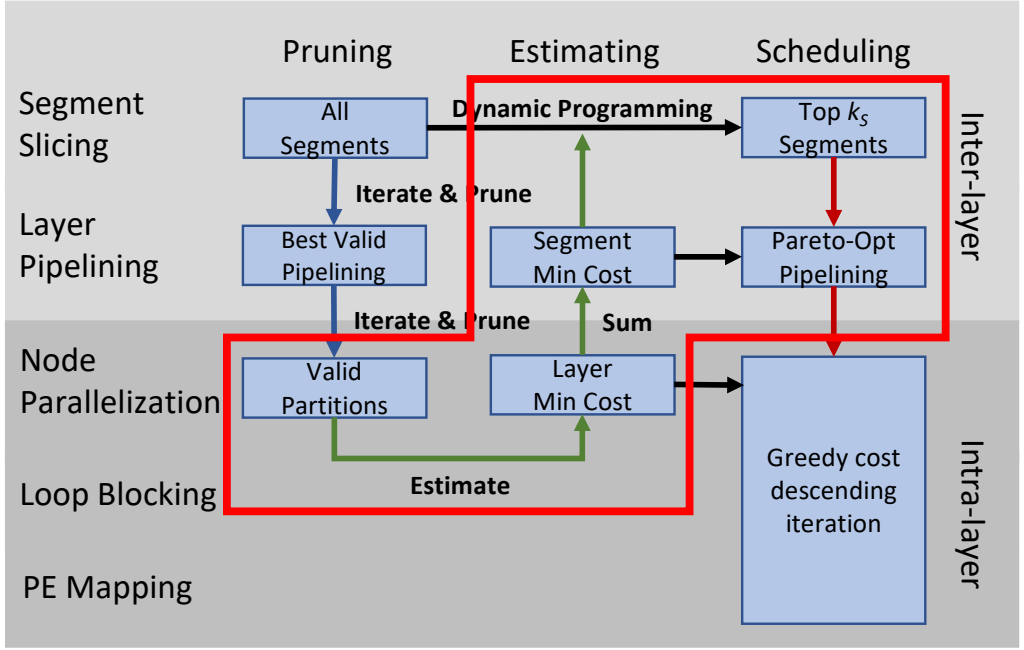
Prioritize candidates based on optimistic intra-layer cost estimation

Fast

No further intra-layer searching

Accurate

More efficient candidates are considered and validated first

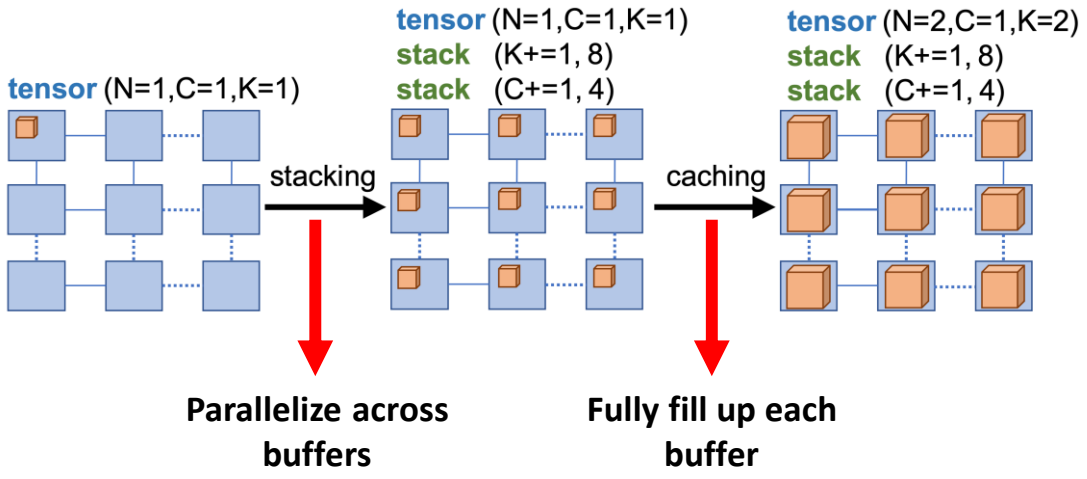


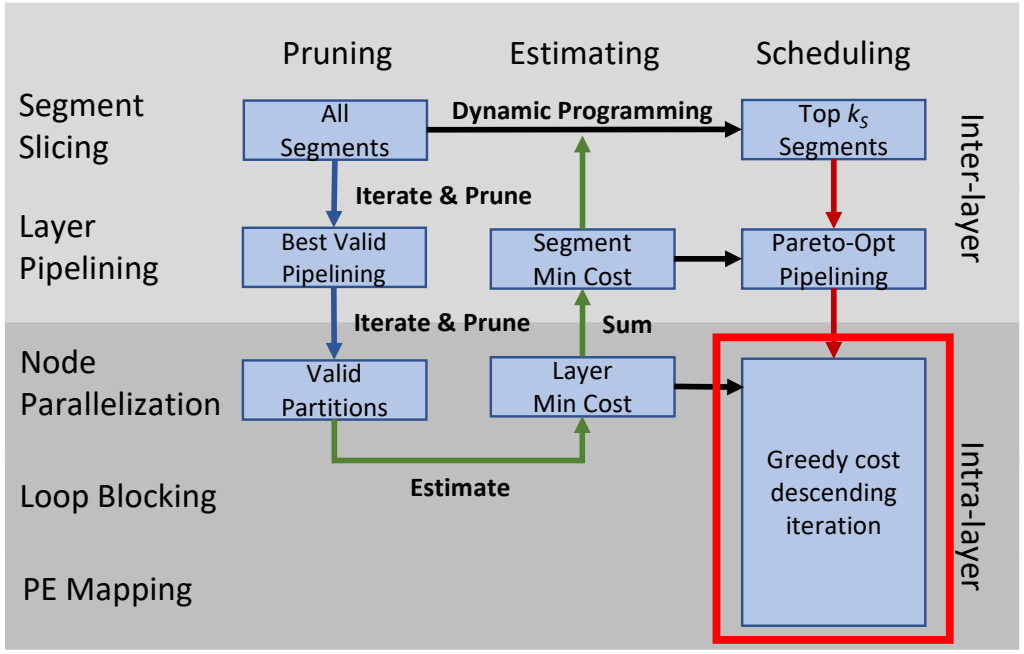
The overall workflow of KAPLA

KAPLA Dataflow Solver

	Validity check	Efficiency estimation
Inter-layer	Conservative pruning	Prioritization based on estimated cost
Intra-layer	**Bottom-up tensor construction**	

Bottom-up tensor construction ensures schemes are always valid





The overall workflow of KAPLA

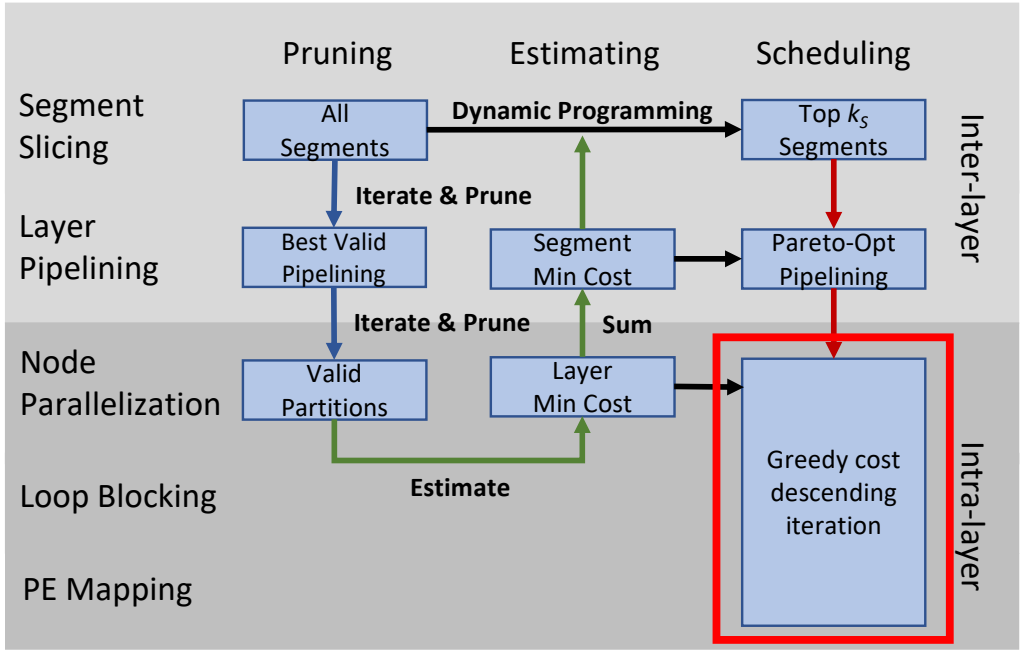
KAPLA Dataflow Solver

	Validity check	Efficiency estimation
Inter-layer	Conservative pruning	Prioritization based on estimated cost
Intra-layer	Bottom-up tensor construction	**Cost descending search**

Iteratively apply stacking and caching with the best loop orders

Algorithm 1 Greedy cost descending algorithm

```
Input: NN layer spec LayerSpec, hardware spec HWSpec, and constraints Constrs
Output: Optimized dataflow scheme OptDf
MemoryHier = [RegFile, GBUF, DRAM]
Df, LoopCnt = initUnitMapping(LayerSpec, HWSpec)
for each m in MemoryHier do
    for order in genLoopOrder(m) do
        Df, LoopCnt = stack(Df, m, LoopCnt, order)
        Df, LoopCnt = cache(Df, m, LoopCnt, order)
        if checkValid(Constrs, LoopCnt) is False then
            continue
        if evalCost(Df) > evalCost(OptDf) then
            OptDf = Df
```



The overall workflow of KAPLA

Evaluation Methodology

- **Evaluated workloads**
  - Inference & training on various NNs
  - Batch sizes 64 & 1
- **Evaluated architectures**
  - 16 x 16 multi-array cloud accelerator
  - single-array edge accelerator
- **Baselines**
  - (**B**) nn-dataflow [ASPLOS'19]: exhaustive search
  - (**S**) KAPLA directives: exhaustive search
  - (**R**) Timeloop [ISPASS'19]: random search
  - (**M**) AutoTVM [NeurIPS'18]: XGBoost

Evaluation Results: Scheduling Speed

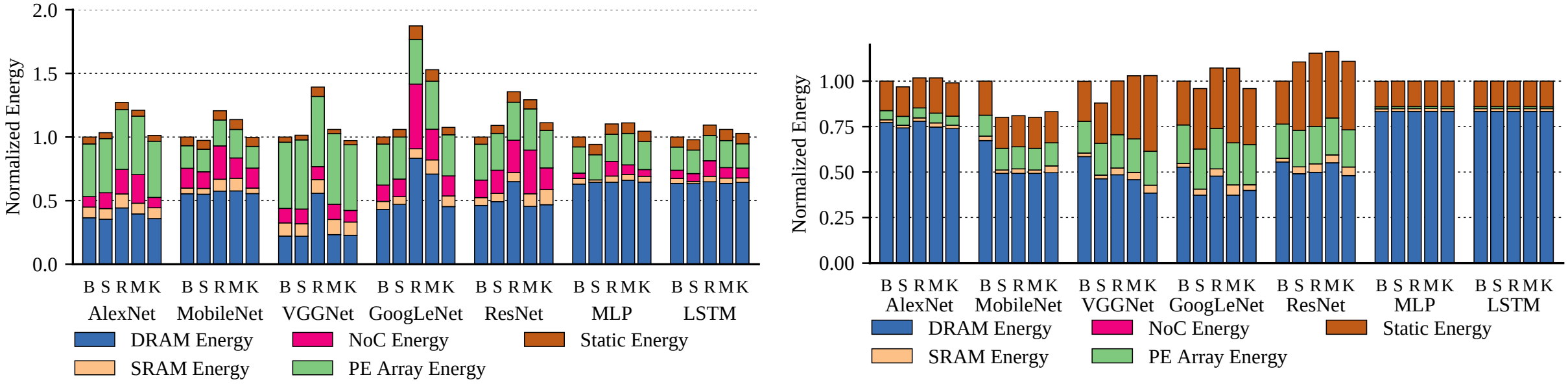
- **518x search speedup**

	B	S	R	M	K
AlexNet	8.7 h	12.7 h	4.6 min	3.2 h	32 s
MobileNet	17.6 h	33.8 h	16.6 min	10.1 h	53 s
VGGNet	6.8 h	12.2 h	5.5 min	3.6 h	35 s
GoogLeNet	64.2 h	111.0 h	31.4 min	37.1 h	262 s
ResNet	78.6 h	114.8 h	42.4 min	28.6 h	118 s
MLP	0.2 h	0.7 h	1.0 min	0.2 h	13 s
LSTM	0.2 h	0.9 h	7.4 min	0.1 h	6 s

Comparison of scheduling time for NN **training** on **multi-array** accelerators.
Measured on an Intel Xeon Gold 5120 processor with 8 parallel processes.

Evaluation Results: Scheduling Quality

- **Only 2.2% quality (energy) loss on average**



Comparison of energy for **training** (left) and **inference** (right) on **multi-array** Eyeriss-like accelerators with batch 64.

Summary

- **A hierarchical taxonomy** that highlights the tight coupling across different levels of the dataflow space as the major difficulty for fast design exploration
- **A set of tensor-centric directives** that can accurately express various inter-layer and intra-layer schemes, and allow for quickly determining their validity and efficiency
- **A generic, optimized, and fast dataflow solver, KAPLA**, which makes use of the proposed directives to explore the design space, and achieves **similar scheduling quality** with **much faster scheduling speed**

Thank you!