

清華大學深圳国际研究生院
Tsinghua Shenzhen International Graduate School

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

MCCSys@ISCA 2026

Simulation, Compilation, and Deployment Optimization for Near-DRAM Processing Architectures

Zhenhua Zhu

Assistant Professor

Shenzhen International Graduate School, Tsinghua University; Dept. of ECE, HKUST

zhuzhenhua@sz.tsinghua.edu.cn, eezhuzh@ust.hk



2026/6/29

NICS EFC

目录 Contents

1 Background and Motivation

2 UniNDP [HPCA' 25]
Unified Compiler for NDP Architecture

3 FlexQ-NDP [ISCA' 26]
NDP Compiler towards Low-bit FP Operators

4 Summary

Big Data Drives the Development of AI

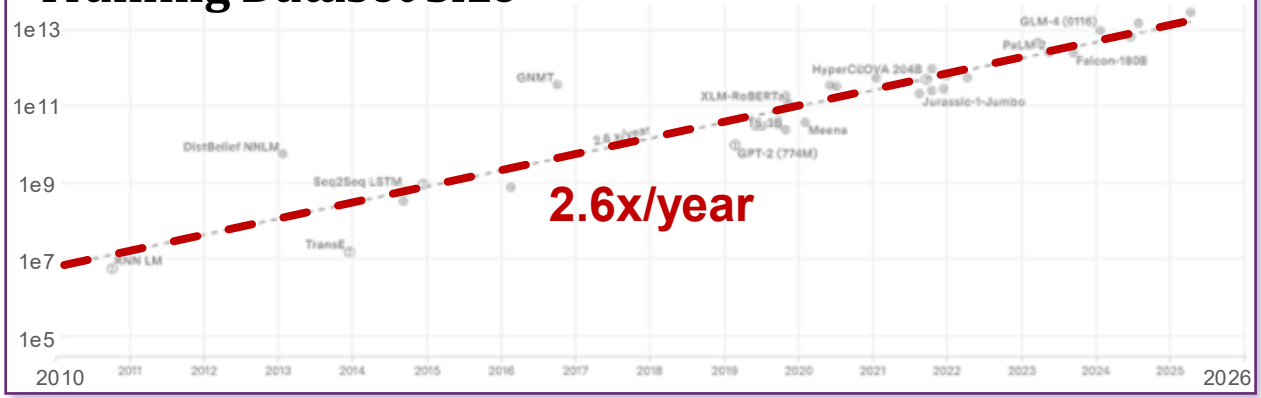


Geoffrey Hinton
Nobel Prize
Turing Award

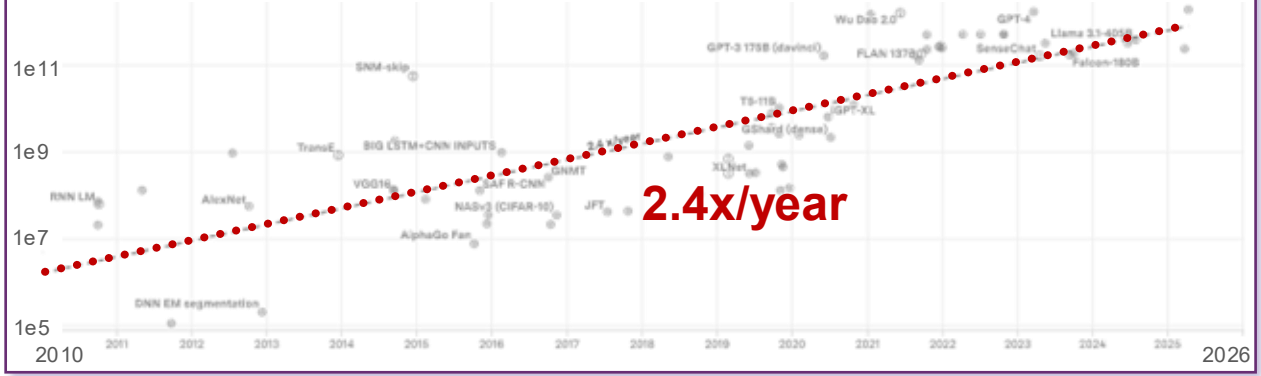
"All you need is **lots and lots of data** and lots of information about what the right answer is, and you'll be able to train a big neural net to do what you want."

—— Reuters Newsmaker event, 2017

Training Dataset Size



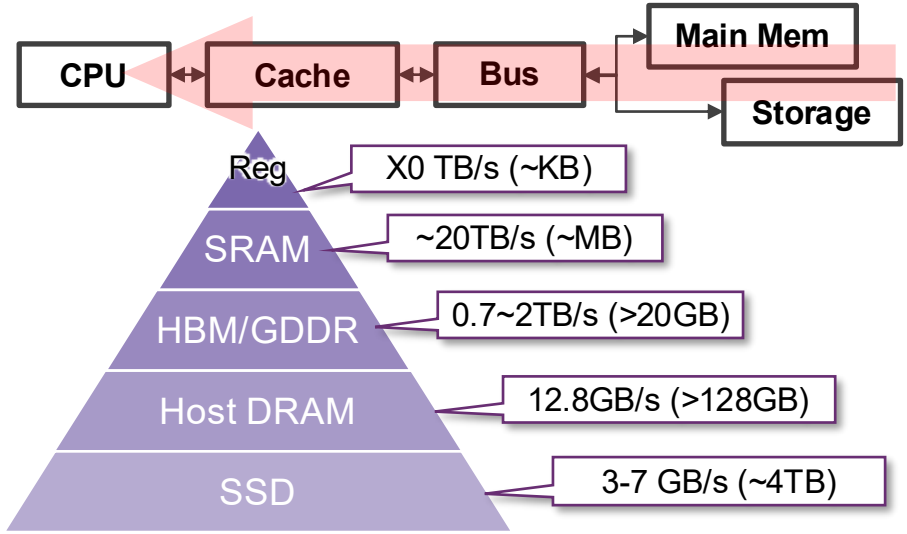
Model Parameter Number



Storing Data in Computer: Memory of von Neumann Arch.

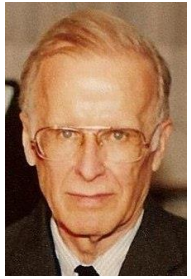
- In von Neumann architectures, data are stored in memory, physically **separated** from the compute units.

Separation between Processor and Memory



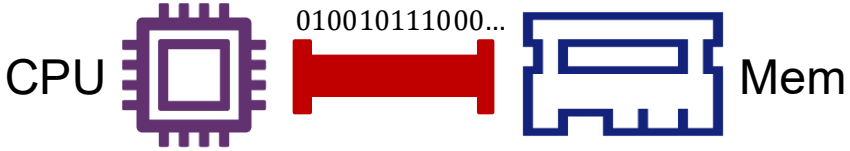
Data resides in
a **hierarchical memory** system

von Neumann Bottleneck



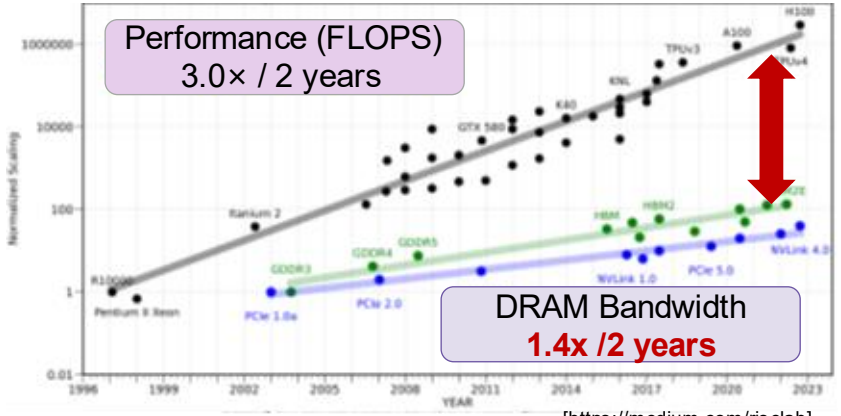
John Backus
Turing Award

In its simplest form a von Neumann computer has three parts: a CPU, a store, ***and a connecting tube*** that can transmit a single word between the CPU and the store.
I propose to call this tube the von Neumann bottleneck.



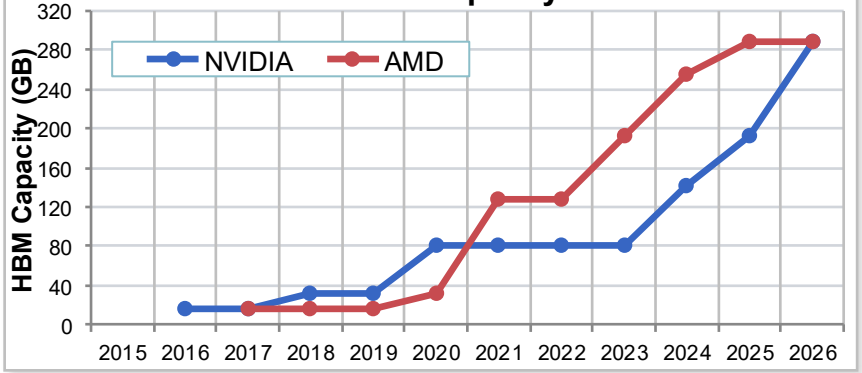
Background: Memory Bottleneck

The Memory Wall Problem

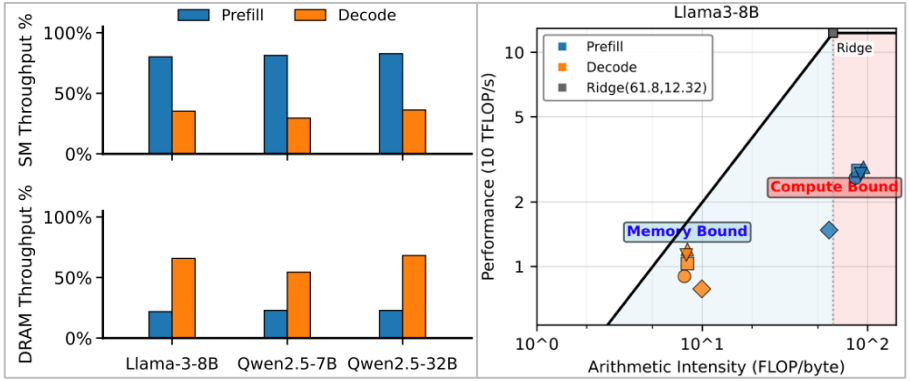


[https://medium.com/riselab]

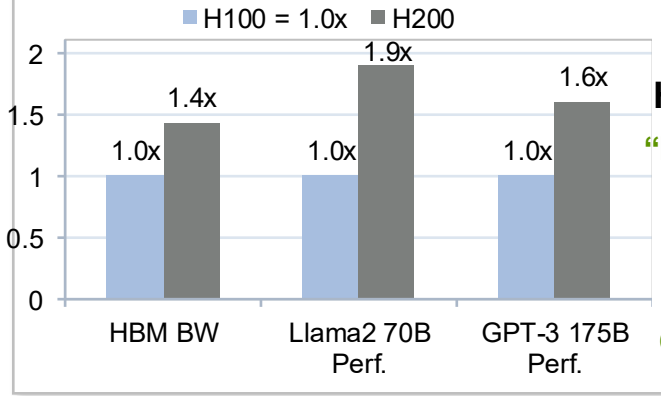
GPU's Moore's Law: HBM Capacity Doubles / 2 Years



LLM: Decode Phase – Memory Bound



[H. Wang et al., A Systematic Characterization of LLM Inference on GPUs, arXiv 2025]



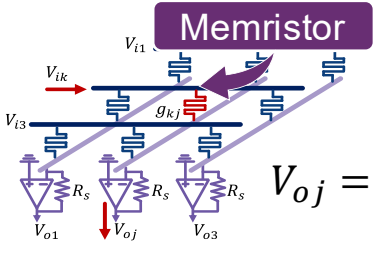
H200 (4.8TB/s)
H100 (3.35TB/s)

"Higher bandwidth alleviates LLM bandwidth bottlenecks and improves Tensor Core utilization."

Breaking the Memory Wall: Processing-In-Memory (PIM)

- Philosophy: perform computations inside memory → ***three*** approaches
  - **Low** data movements, **High** internal bandwidth

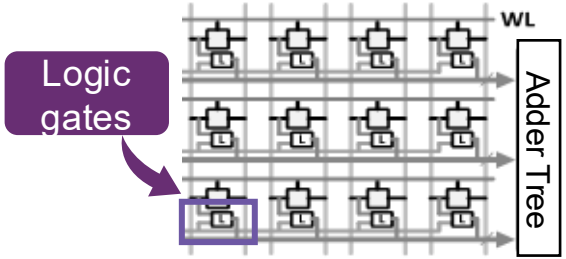
Analog-domain In-Mem-Comp.



$$c_{kj} = -\frac{g_{kj}}{g_s}$$

$$V_{oj} = \sum_k V_{ik} \cdot c_{kj}$$

Digital-domain IMC



Digital-domain Near Memory Processing Architecture

Trend of NDP

Overcome bandwidth limitation of ext. bus

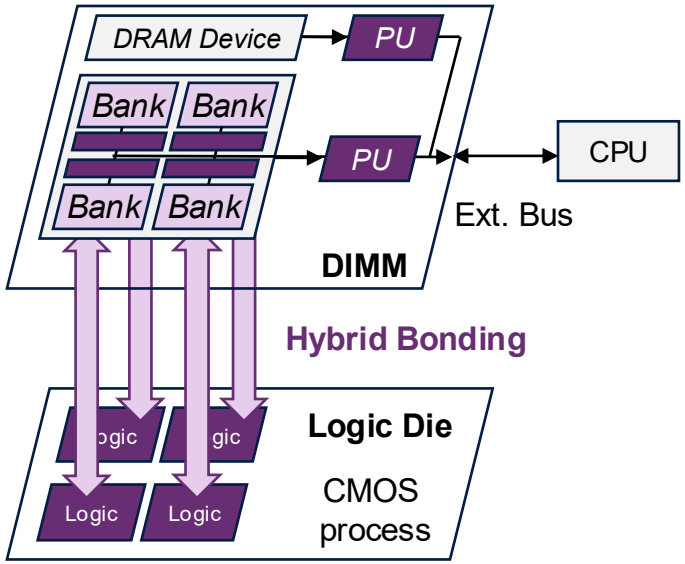
Processing unit (PU) on DIMM board

Fully use internal bandwidth

PU within DRAM device w/ DRAM process

Break limitation of DRAM process

Use 3D stack tech (e.g., Hybrid bonding), build PU in logic die



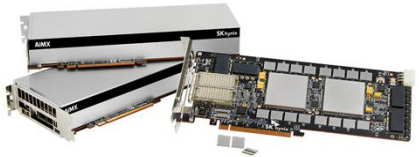
Commercial NDP Products/Prototypes

- Near DRAM Processing (NDP/NMP/PIM/…)
  - Place processing unit (PU) near / inside DRAM to fully utilize internal bandwidth



UPMEM 1st GEN.

General-purpose Computation



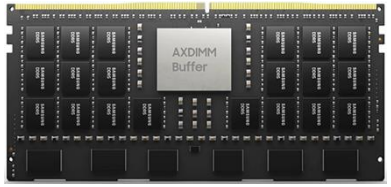
Hynix GDDR6-AiM

Neural Networks & LLM

2018

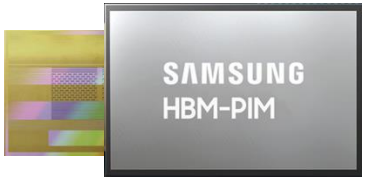
2021

Samsung AxDIMM



2021

Samsung HBM-PIM



2022

2024

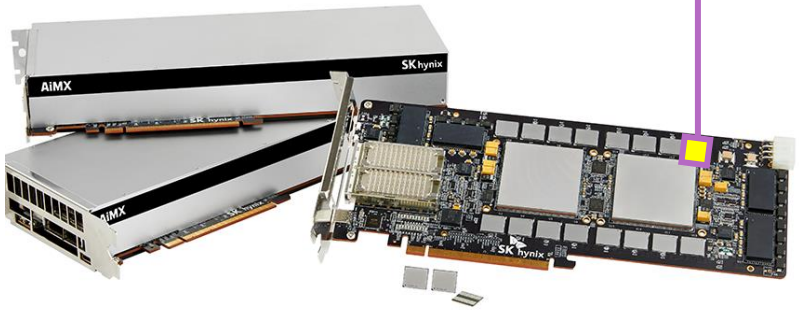
UPMEM 2nd GEN.

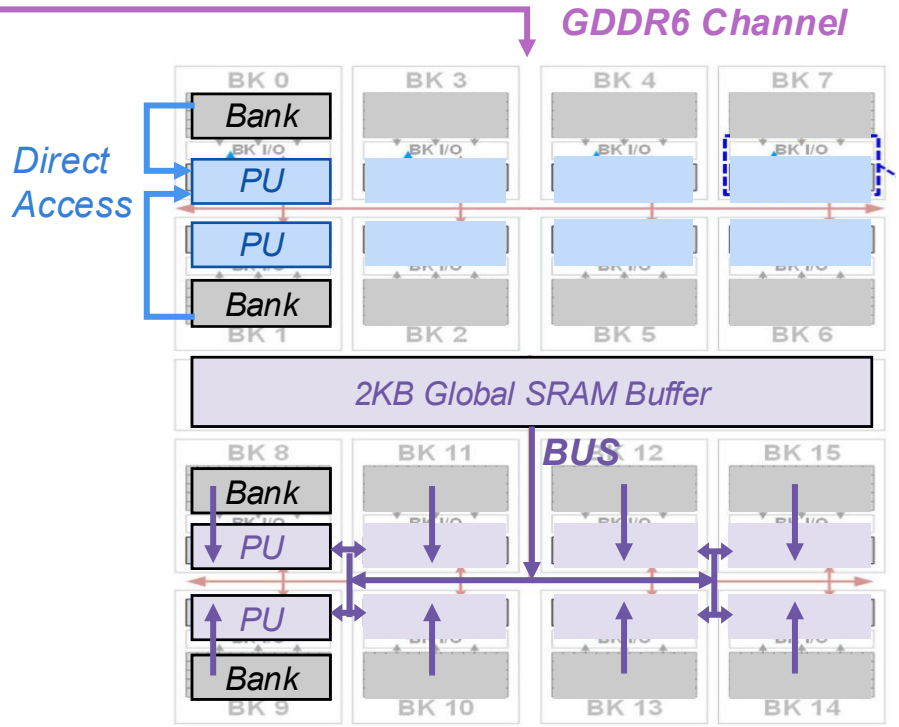


Recommendation System

Typical NDP Architecture: Hynix AiM

- Take *GDDR6-AiM (Accelerator-in-Memory)* as an Example of NDP Architecture

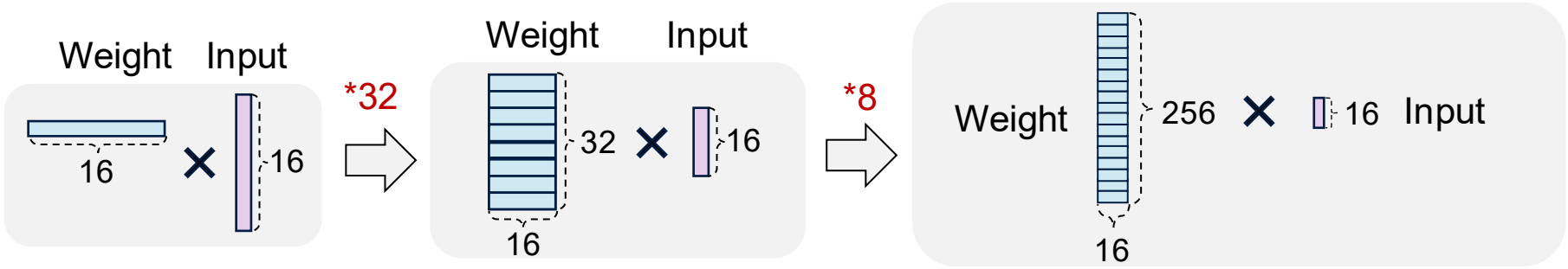


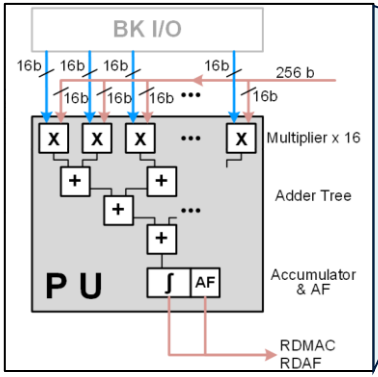


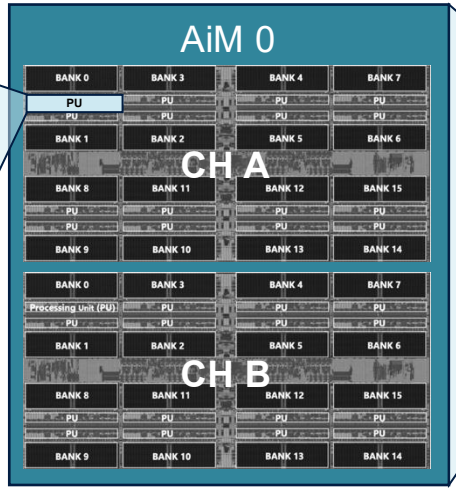
Hynix GDDR6-AiM

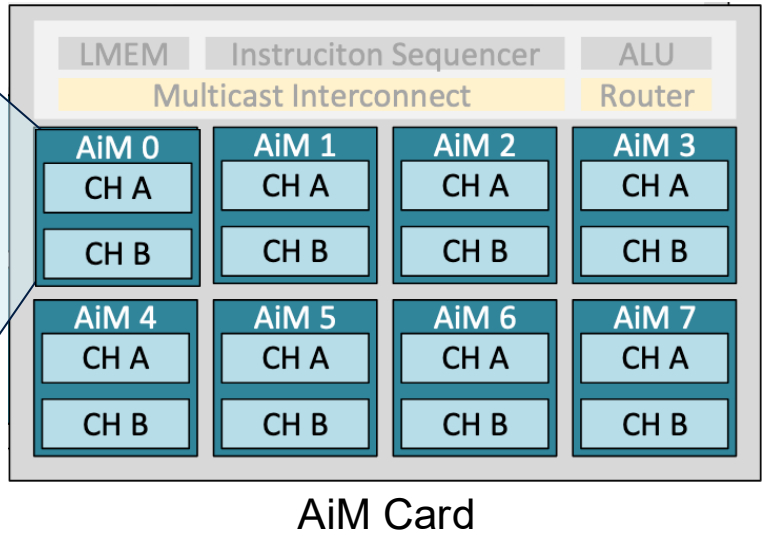
- DRAM type: GDDR6
- 16 PU / Channel, 32 GFLOPS each
- 2KB Global SRAM Buffer / Channel

Typical NDP Architecture: Hynix AiM





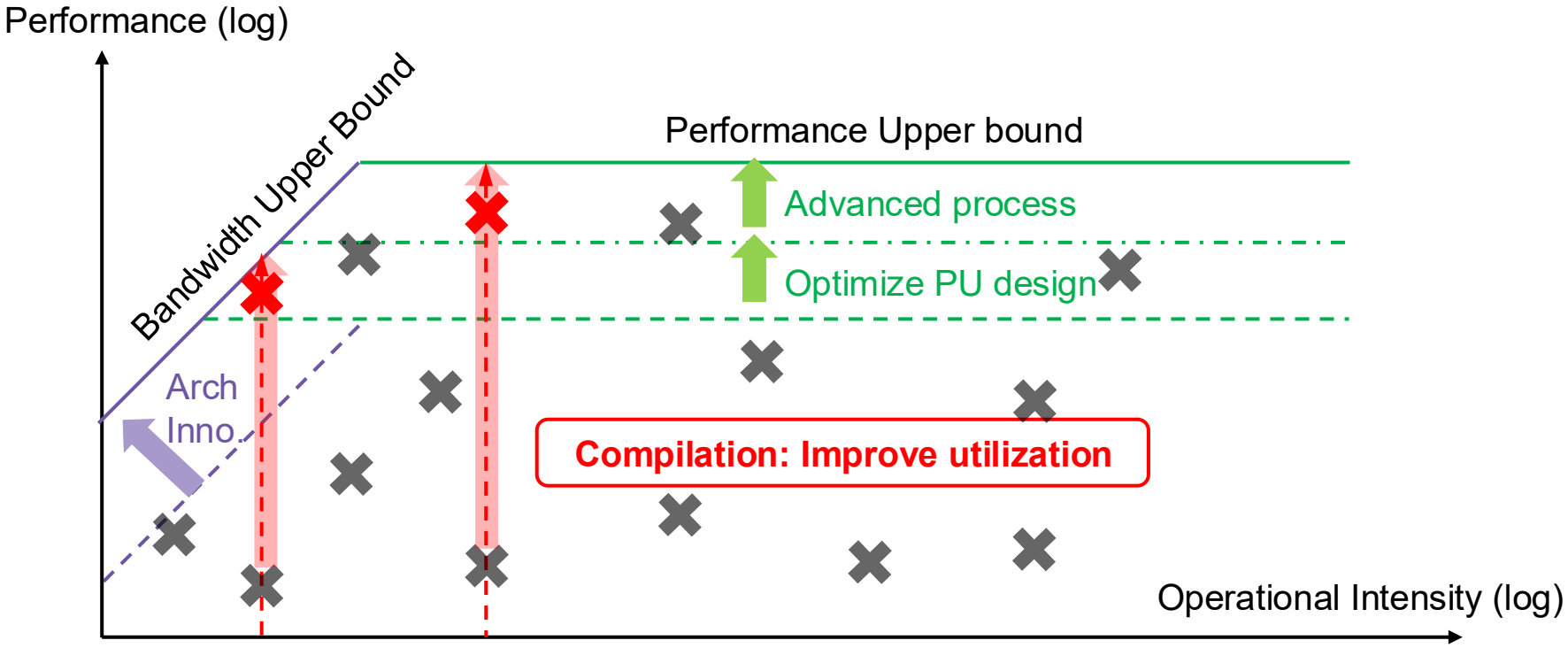




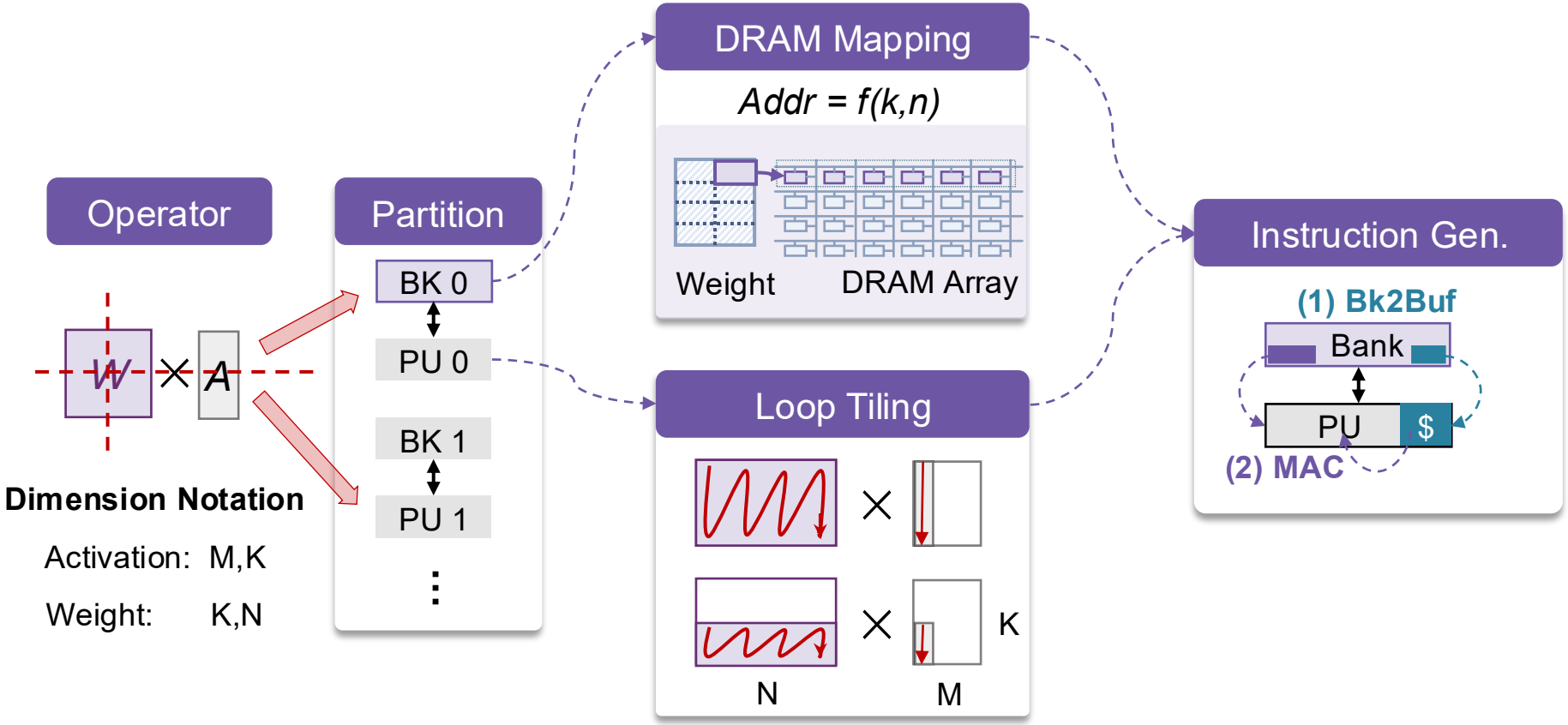
AiM Card

How to Push NDP Performance to the Limit

- Simplified roofline model for NDP architecture

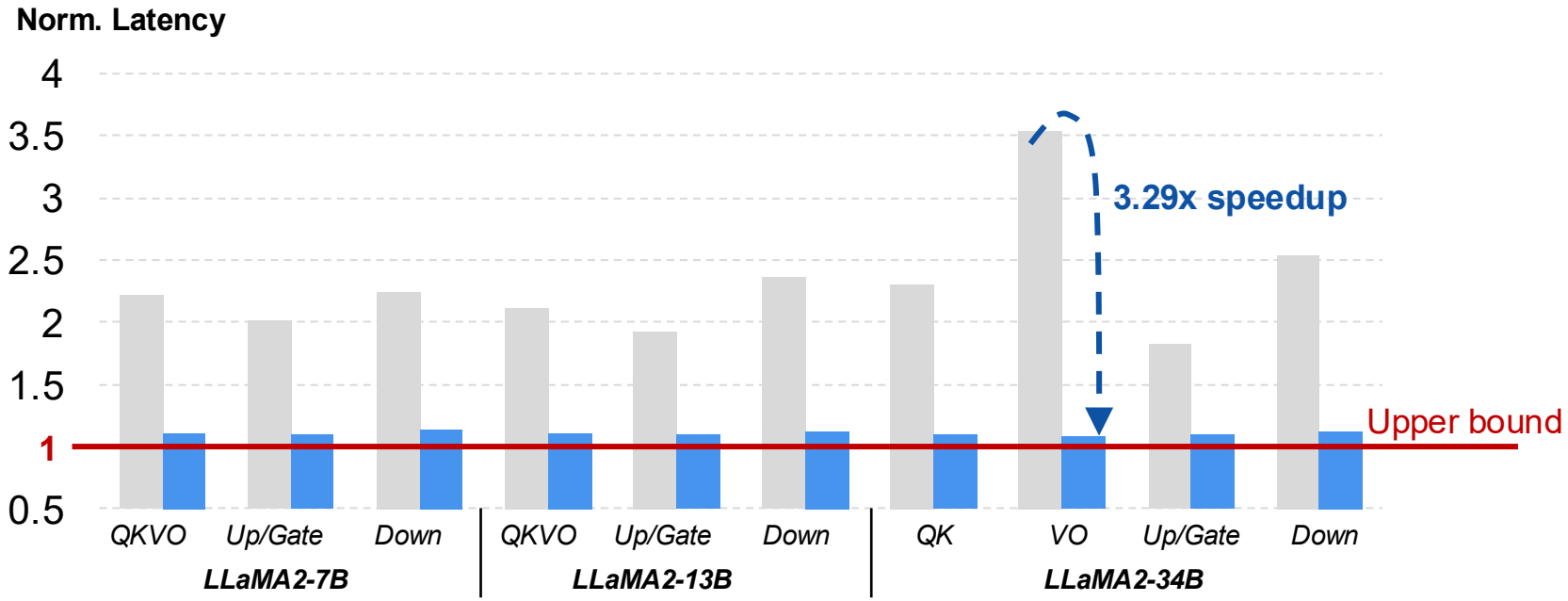


Typical NDP Compilation Flow



Our Point: How to Exploit NDP Performance

- Running the same GEMM operator on the same NDP architecture:
  - Compiler strategy optimization can lead to up to **3x** performance difference
  - Compiler optimization can unlock the performance potential of NDP, approaching to the theoretical upper bound





清華大學深圳国际研究生院
Tsinghua Shenzhen International Graduate School

Github



[HPCA'25, https://github.com/thu-nics/UniNDP]

UniNDP: A Unified Compilation and Simulation Tool for Near DRAM Processing Architectures

Tongxin Xie*, **Zhenhua Zhu***, Bing Li, Yukai He, Cong Li, Guangyu Sun, Huazhong Yang, Yuan Xie, and Yu Wang

Tsinghua University, Hong Kong University of Science and Technology, Chinese Academy of Sciences, Capital Normal University, Peking University

E-mail: xtx23@mails.tsinghua.edu.cn, zhuzhenhua@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn



NICS EFC

Arch & Dataflow Impact Compilation Strategy

- **DRAM devices**: Different read / write latency and bandwidth
- **Processing units**: Different memory levels & computational power

UPMEM: 1 PU / **Bank**
HBM-PIM: 8 PU / **Device**
AxDIMM: 1 PU / **Rank**

Effective Bandwidth

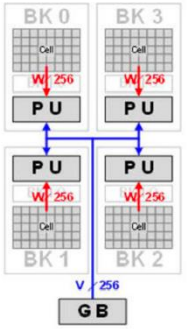


Local memory space (each PU)



- **Data Flow:** Input + Output, Buffer Design & Size

Example	OpSrc1	OpSrc2	Input Source	Output Destination
HBM-PIM	DRAM	PU Input LB	Host	PU Output LB - DRAM bank
		PU Input LB	DRAM	PU Output LB - DRAM bank
GDDR6-PIM		Device GB	Host	Host maintenance
		Device GB	DRAM	Host maintenance
		DRAM	Host	Host maintenance
UPMEM		**PU Input LB**	DRAM	PU Output LB - DRAM bank
AxDIMM		PU Output LB	PU	PU Output LB



Input Sharing

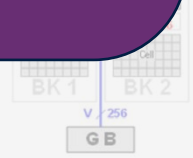
Arch & Dataflow Impact Compilation Strategy

- **DRAM devices**: Different read / write latency and bandwidth
- **Processing units**: Different memory levels & computational power

UniNDP is a unified compilation tool from NN to NDP that:

- **Supports** compilation on different NDP architectures.
- **Find Optimal Compilation Strategies** for different NDP architectures

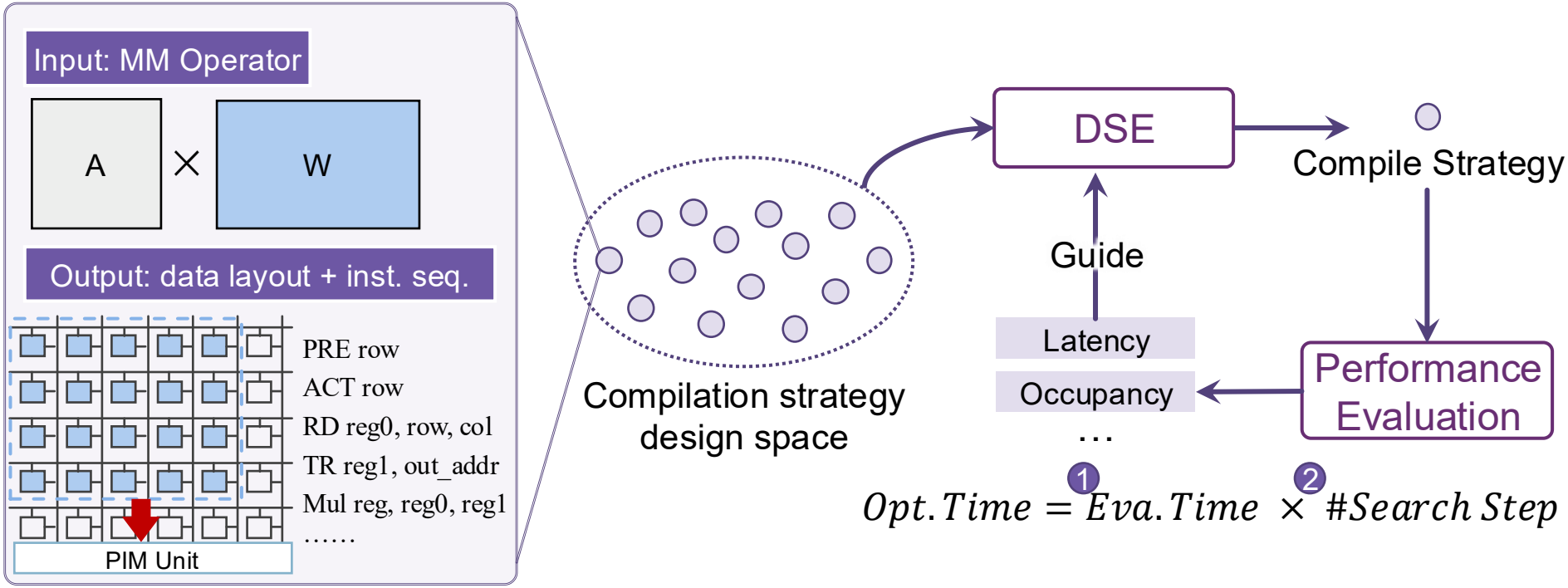
		DRAM	Host	Host maintenance
UPMEM		**PU Input LB**	DRAM	PU Output LB - DRAM bank
AxDIMM		PU Output LB	PU	PU Output LB



Input Sharing

Key Idea: Optimize Data Mapping and Dataflow

- ***PIM feature***: storage and computation are **tightly coupled**, performance is heavily depends on the location of data storage (i.e., data layout)
  - We model the compilation process as a **Design Space Exploration (DSE)** problem



UniNDP Framework Overview

1. How To support ?

Different Archs

→ *Unified Architecture Abstraction*

Different Dataflows

→ *Unified NDP ISA*

2. How to Finding Optimal Compilation Strategies?

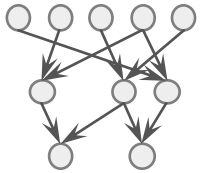
Which strategy is better?

→ *Simulator*

Generate search space & find best?

→ *Compiler*

UniNDP Design



Operator Graph

UniNDP Compiler



Instruction File @Top-K Strategies

UniNDP Simulator

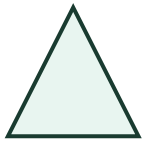


Instruction File & Compilation Strategy @ Best Performance

How to Support Different NDP Archs

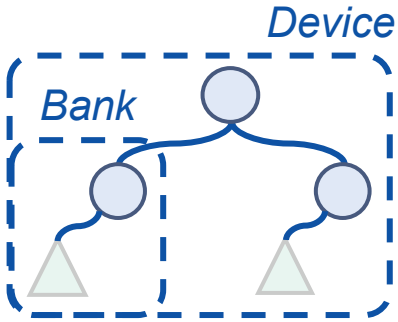
- Unified Tree-based NDP **Architecture Abstraction**

1. Memory Node | **2. Central Node** | **3. PU Node** | **4. Buffer Node**



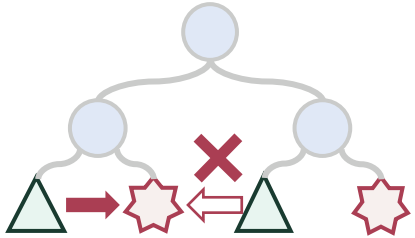
Memory Configs

- Capacity
  - #Rows / #Columns
- Timing Parameters
  - RL / WL / tCCD
- Device Width
- Burst Length, …

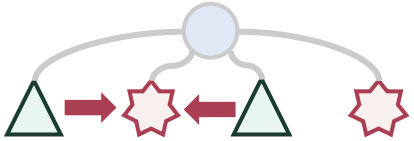


DRAM Hierarchy

- Channel
- Rank
- Device
- Bank, Sub-array, …

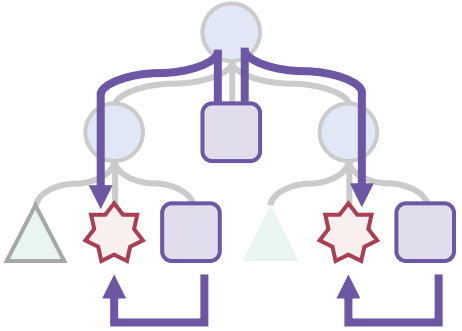


Direct Access Memory Under Same Central Node



Access Op A & B from 2 Banks

Global Buffer (GB), Access Through Bus



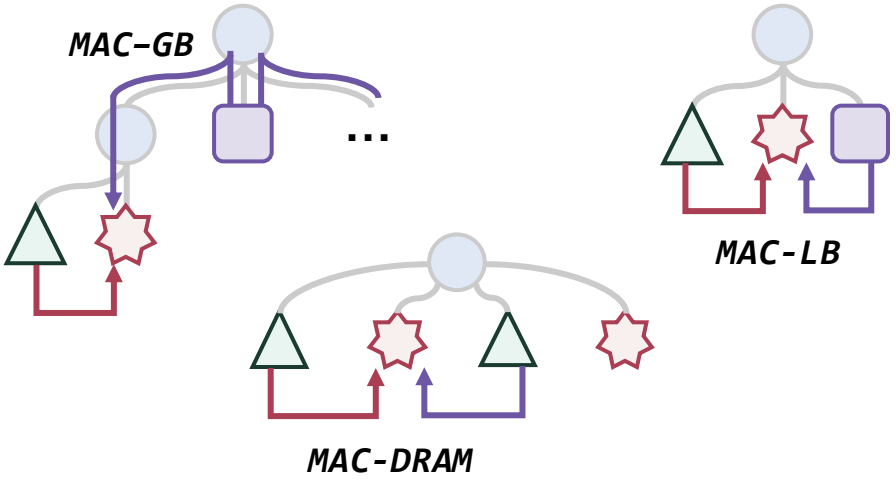
Local Buffer (LB)

How to Support Different NDP Archs

- Unified NDP **ISA**

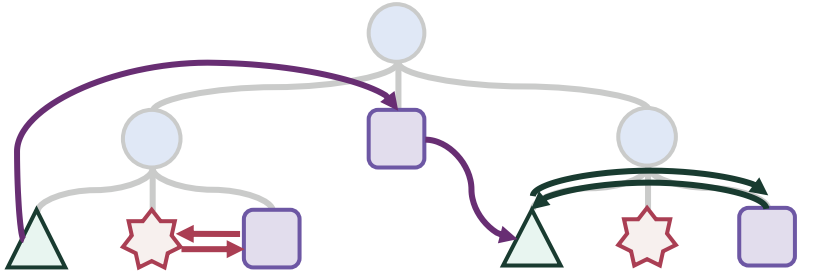
1. Compute Instructions ***(Different Input Source)***

- Op A → from DRAM
- Op B → from Local Buffer / Global Buffer / Another DRAM Bank



2. Data Copy Between MEMs

DRAM to/from GB
Data Exchange between DRAM



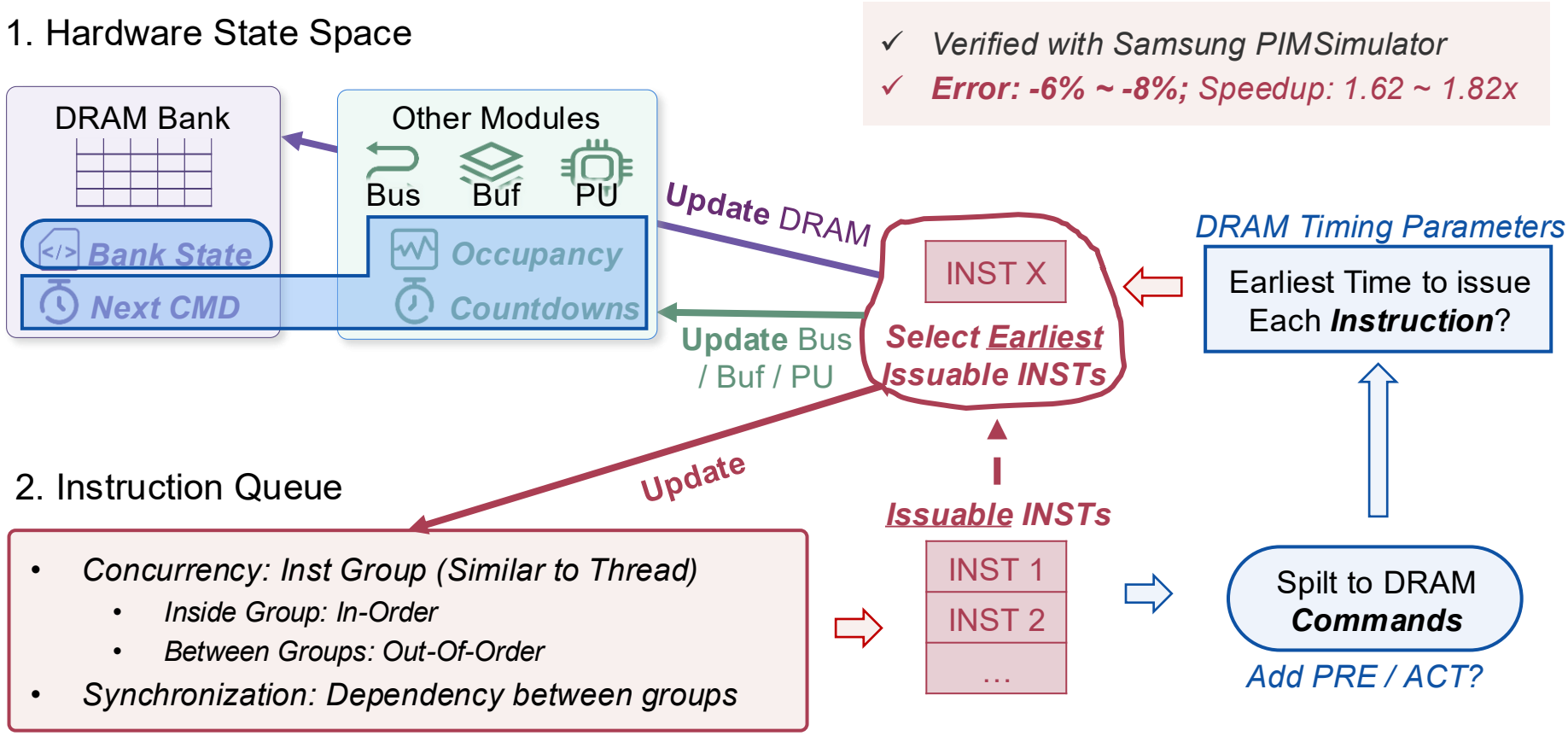
Reg to/from LB
Save & Accumulate to Partial Sum

LB to/from DRAM
DRAM serves as a bigger buffer

3. Host Commands for Input / Weight Distribution & Output Collection



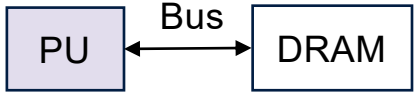
Eva.Time: Instruction-driven Cycle-accurate PIM Sim



#Search Step: Space Pruning-based Compilation

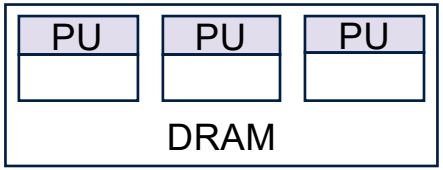
- What's New about PIM Compilation? ***Data Layout !***

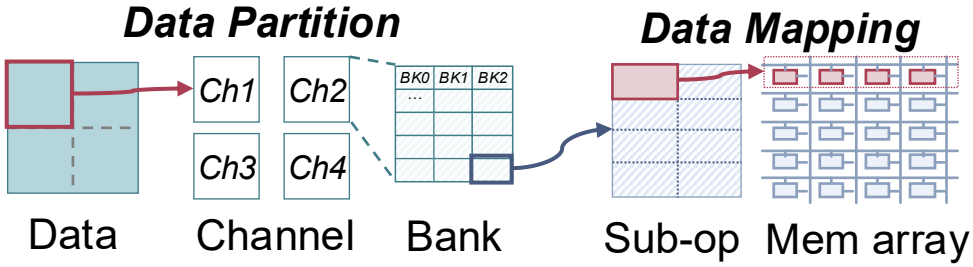
Von-Neumann



Simple Layout (Row-first / Col-first)

NDP





Data partition search space

For operator $O^{(M,N)} = W^{(M,K)} \times A^{(K,N)}$, construct search space based on the split numbers of each dimension across hardware levels:

$$\{M_{\{Ch,Rk,Dv,Bk\}}, N_{\{Ch,Rk,Dv,Bk\}}, K_{\{Ch,Rk,Dv,Bk\}}\}$$

Hardware constraint:

$$e.g., M_{Ch} \times K_{Ch} \times N_{Ch} \leq \#\,Ch$$

Data mapping search space

Allocate DRAM space for data $W^{(M.K)}$:

$$(M^B, K^B), (M^R, K^R)$$

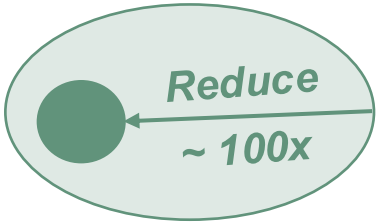
Hardware constraint:

$$M^B \times K^B \times BitWidth \leq BurstLen \times DeviceWidth$$

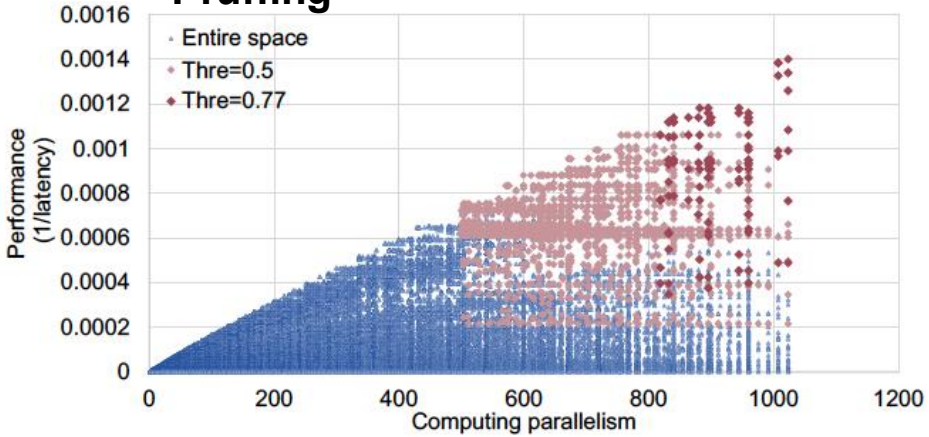
$$BurstLen \times M^R \times K^R \leq \#Col\ per\ Row$$

#Search Step: Space Pruning-based Compilation

- Efficient Search: search space pruning + performance predictor



Search Space Pruning



Pruning indicator

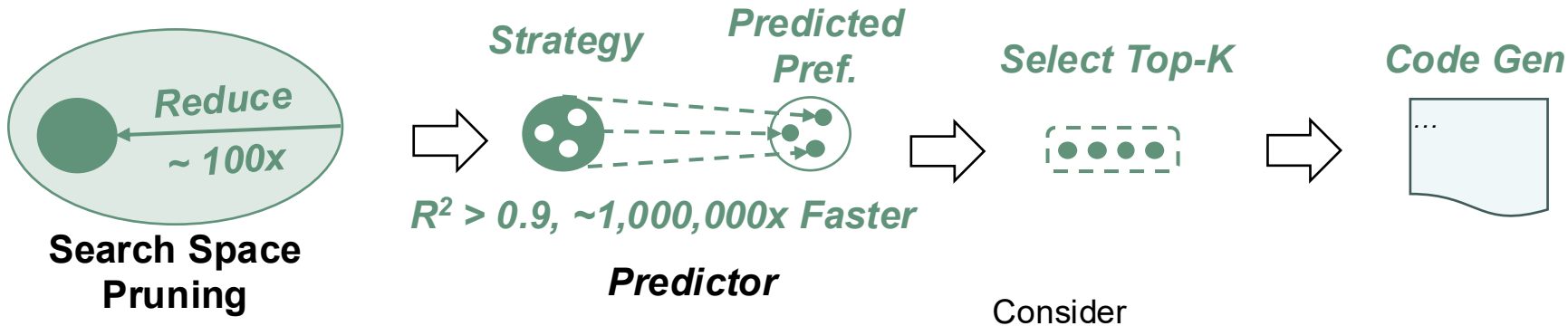
- Hardware utilization (PU/Buf)
- Perf. upper bound: ignore host I/O latency

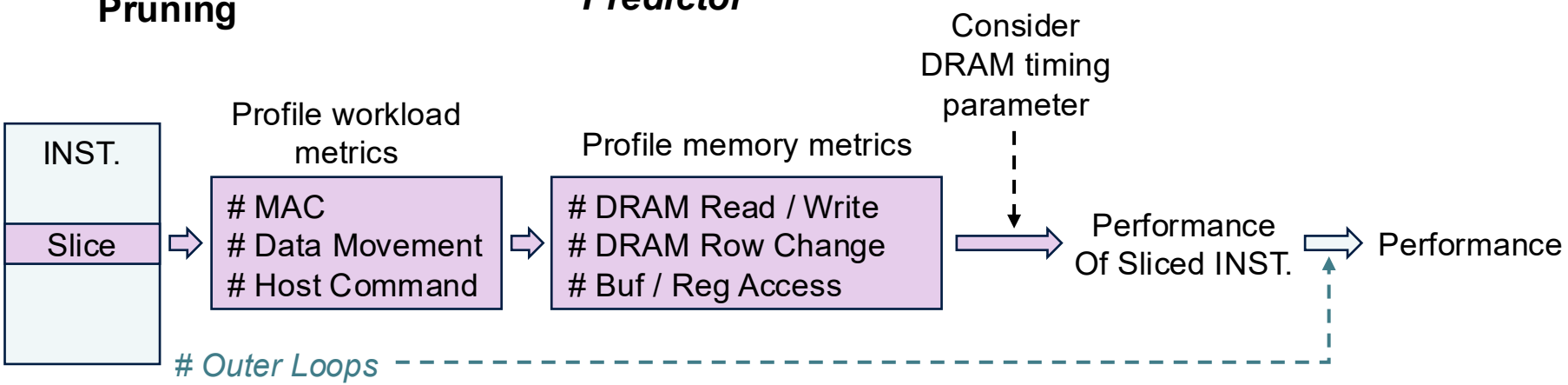
Pruning results

- Search time: 4200s → 35s, **120x ↑**

#Search Step: Space Pruning-based Compilation

- Efficient Search: search space pruning + performance predictor





Results - Experiment Setup

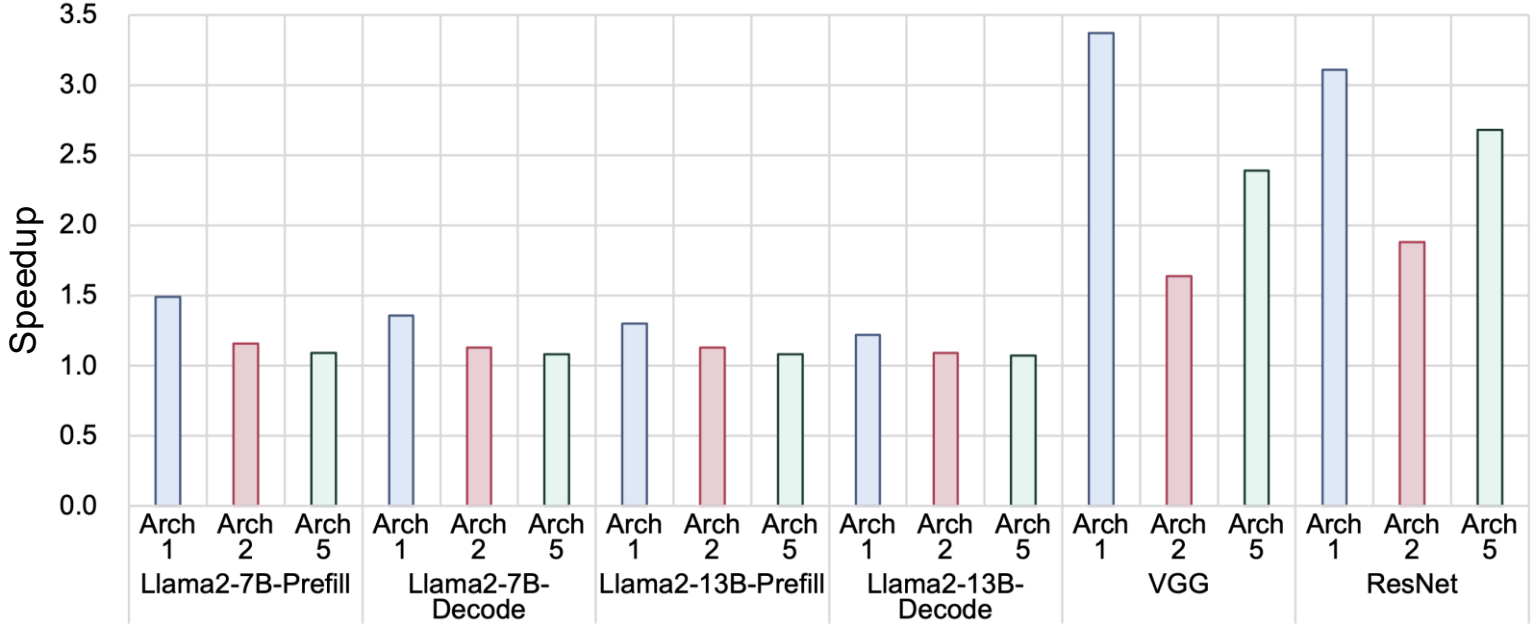
- Hardware configuration

Arch Number	1. UPMEM	2. AiM (16 PU)	3. AiM (8 PU)	4. HBM-PIM	5. DIMMining
NDP level	**Bank**	**Device**	**Device**	**Device**	**Rank**
# CH / RA / DE / BA	20 / 2 / 8 / 8	64 / 1 / 1 / 16	64 / 1 / 1 / 16	64 / 1 / 1 / 16	20 / 2 / 8 / 8
# PU	1 per Bank	16 per Device	8 per Device	8 per Device	4 per Rank
Buffer Design	64KB local	2KB shared	2KB shared	0.5KB local	64KB local
Memory Spec	DDR4 (x8)	GDDR6 (x16)	GDDR6 (x16)	HBM2 (x16)	DDR4 (x8)
Timing parameters	Follows DRAMSim3				
PU capability	**Matches** the internal bandwidth of the DRAMs by default				

- Workload (CNN+LLM)
  - Single operator: MM & MVM (AlexNet, ResNet, VGG, Llama2-7/13/34B)
  - End-to-end: LLama2-7/13B, ResNet, VGG
- Baseline: Layout method in Hynix AiM / Samsung HBM-PIM

Experimental Results

- Compared to the SOTA commercialized PIM compilation Strategy (e.g., Samsung HBM-PIM, Hynix AIM)
  - CNN (ResNet, VGG, etc.): 1.64-3.37x performance improvement;
  - LLM (Llama2 7B/13B, prefill/decode): 1.08-1.49x performance improvement





清華大學深圳国际研究生院
Tsinghua Shenzhen International Graduate School

Github



[ISCA'26, https://github.com/ISCA26-FlexQ-NDP-ae/flexq_ndp]

Bringing Near Data Processing into the Low-Bit Floating-Point Era

Tongxin Xie, Mingyu Gao, Zehao Wang, Zhihao Jia, Yuechen Xi, Bing Li,
Mo Guang, Jiale Yan, Kaiwen Long, Xingcheng Zhang,
Huazhong Yang, Yuan Xie, **Zhenhua Zhu**, Yu Wang

Tsinghua University, Hong Kong University of Science and Technology,
Chinese Academy of Sciences, Li Auto
E-mail: xtx23@mails.tsinghua.edu.cn, zhuzhenhua@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn



NICS EFC

Algorithmic Trend: Low-Bit Floating-Point

FP4 Training

Quartet: Native FP4 Training Can Be Optimal for Large Language Models

Roberto L. Castro* ISTA
Andrei Panferov* ISTA
Soroush Tabesh ISTA
Oliver Sieberling ETH Zürich
Jiale Chen ISTA
Mahdi Nikdan ISTA
Saleh Ashkboos ETH Zürich
Dan Alistarh ISTA & Red Hat AI

Speedup Results. Next, we evaluate the efficiency of our implementation on the NVIDIA RTX 5090 GPU by measuring its performance across single layers of standard shapes, and aggregating across an entire Transformer block. Speedup results are shown in Figure 3, using a batch size 64 and sequence length of 512. The FP8 baseline is provided by CUTLASS MXFP8 kernels, while the BF16 baseline uses PyTorch, both using Blackwell-optimized kernels. Inference speedups are more pronounced due to the lower cost of the forward pass compared to the backward pass, and the latter's higher computational complexity. The speedup scales with the arithmetic intensity (i.e., model size), reaching up to 2.4× over FP8 and 4× over BF16 on the forward pass, where it stabilizes. In the backward pass, our implementation achieves up to 1.6× over FP8 and 2.3× over BF16, resulting in an overall training speedup of up to around 1.8×, and 2.6×, respectively.

4× Forward, 2.6× Backward Speedup vs. BF16

FP4 Inference (Attention)

SageAttention3: Microscaling FP4 Attention for Inference and An Exploration of 8-bit Training

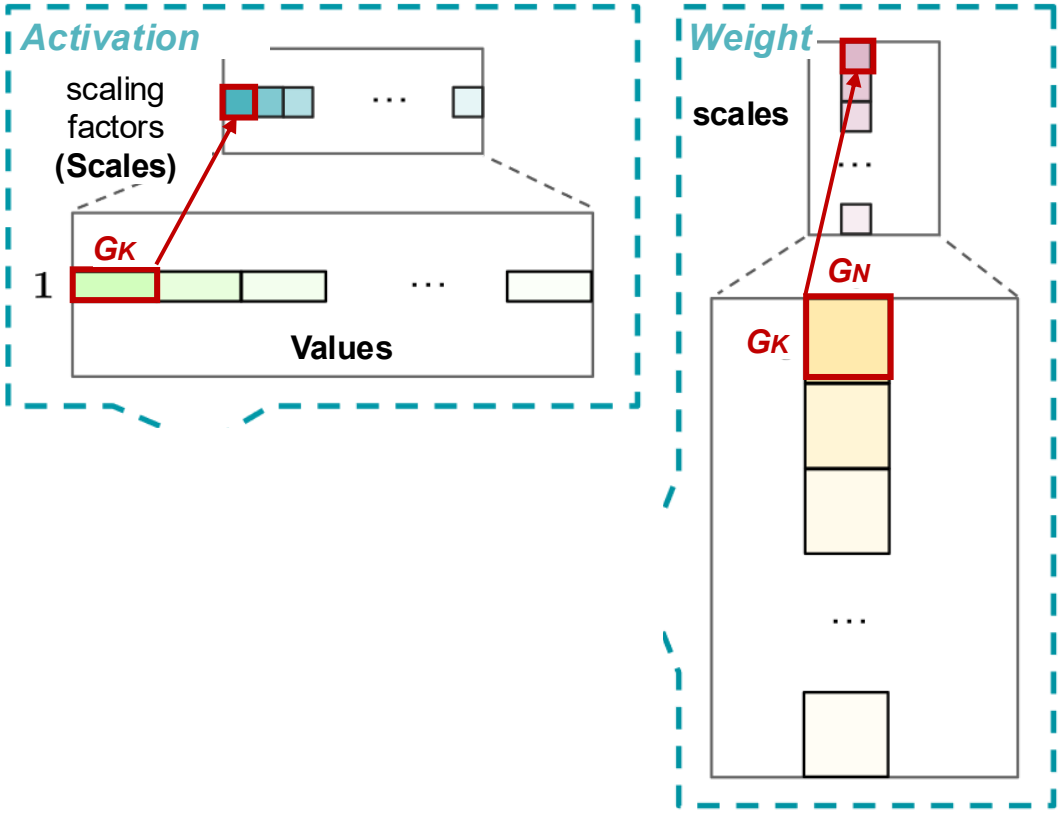
Jintao Zhang*, **Jia Wei***, **Pengle Zhang**, **Xiaoming Xu**, **Haofeng Huang**, **Haoxu Wang**, **Kai Jiang**, **Jun Zhu**, **Jianfei Chen**
Tsinghua University
{zhang-jt24@mails., jianfeic@, dcszj@}tsinghua.edu.cn

BRIDGING THE GAP BETWEEN PROMISE AND PERFORMANCE FOR MICROSCALING FP4 QUANTIZATION

Roberto L. Castro* ISTA & Red Hat AI
Vage Egiazarian* ISTA
Denis Kuznedelev* Yandex Research
Andrei Panferov* ISTA

3~4× Speedup vs. BF16

Low-Bit FP Requires Fine-Grained Quantization



Low-Precision FP Features

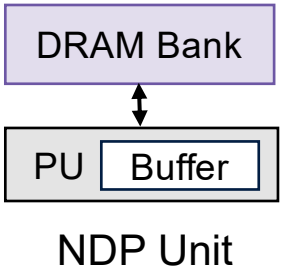
- Grouped Data with Per-Group Scales
- Extra Dequantization Overhead: ✕**scales**

- Lower Bit-Width -> Finer Grouping

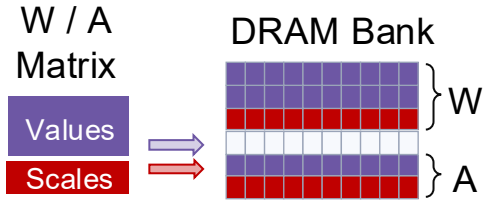
Data Format	GK	GN
DS (FP8)	128	128
MX (FP4)	32	1
NV (FP4)	16	1

- More Scale Metadata (~12.5%)
- Higher Dequantization Overhead

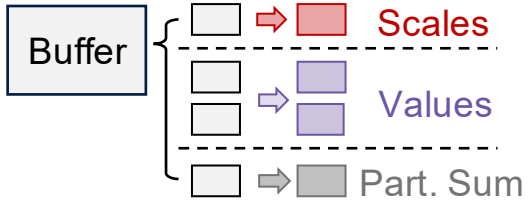
Baseline: FP-Compatible Compilation Flow



DRAM Mapping

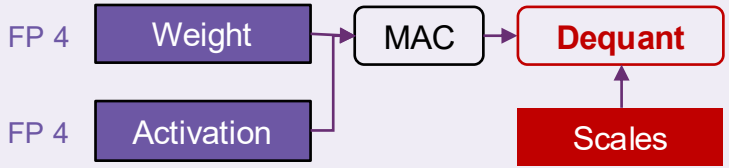


Buffer Allocation（New）

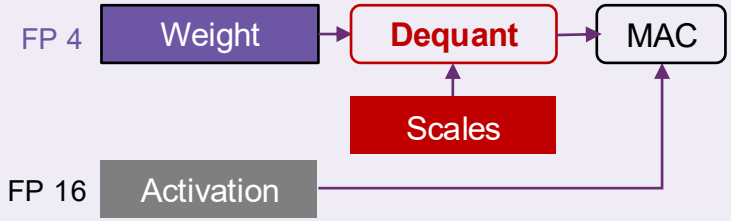


Dataflow（Code Generation）

W-A Quantization **(Result Dequant.)**



W-Only Quantization **(Weight Dequant.)**



Challenge 1： Excessive Compilation Strategy Space

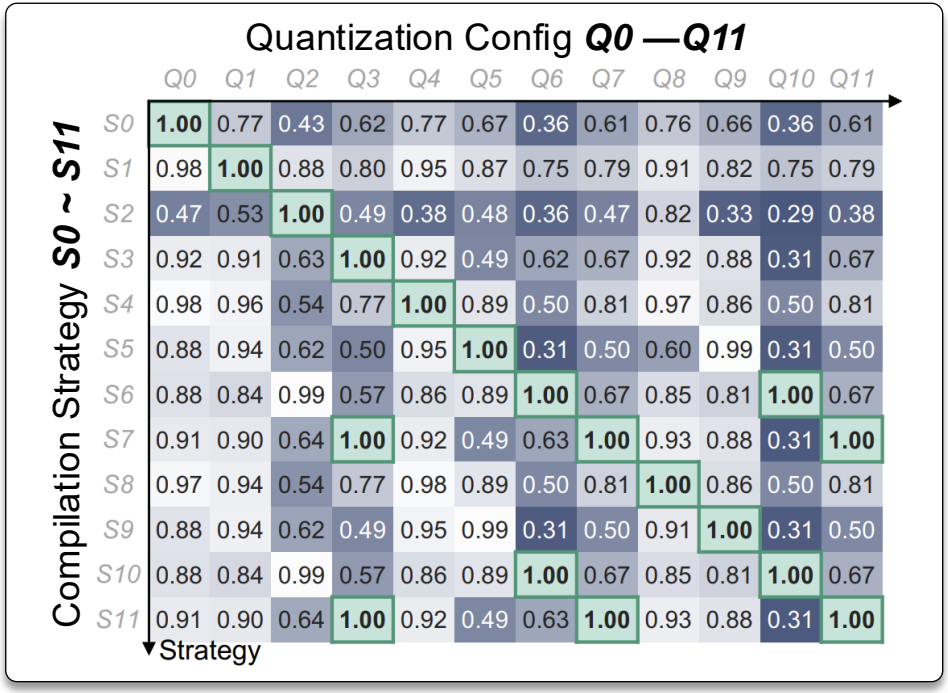
- Quantization-aware strategies: no **one-size-fits-all** compilation strategy
- Huge design space: >10^{10} candidates per quantization config, requiring efficient exploration

Quantization Configurations

Quant Method	W-A
Precision (Value, Scale)	(4, 8) / (8, 16), (8, 32)
Quantization Group (N,K)	(1, 16), (1, 32), (64, 64), (128, 128)

1. Search optimal strategy for each config
2. Swap and evaluate performance (/ optimality)

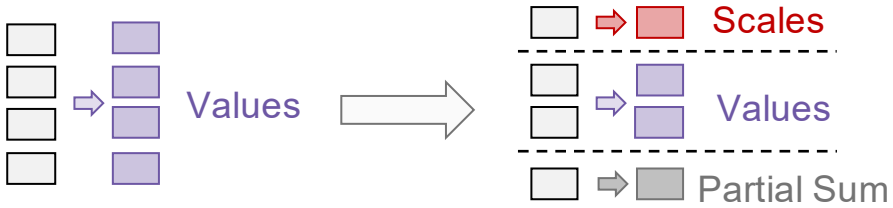
Large Performance Gap (30–100%) Found,
No Single Strategy Fits All Groupings



Challenge 2: Extra DRAM Row-Switch Overhead

- Limited on-chip SRAM (~Kb) makes buffer hit rate critical
- Fine-grained quantization groups introduce two issues:
  - Non-negligible scale metadata **occupies buffer space**, reducing hit rate

For FP4 Value & FP8 Scale,
QGorup= (1,16), scale data size
is **12.5%** of value data size



- Scale accesses cause DRAM switches between value/scale, increasing #**RowChange** by **2-10x**

Buffer Size (bit)	1280	2560	3840	**5120**
# Row Change	11.56x	4.17x	2.42x	**2.47x**
+ Latency	392.98%	85.50%	37.96%	**34.04%**

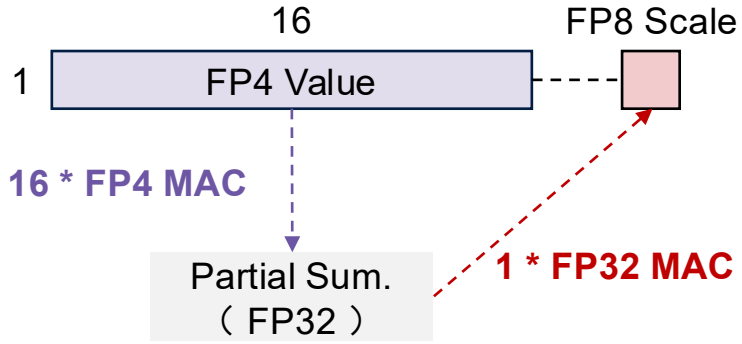
Increase Latency by 34%

Challenge 3: Extra Dequantization Latency

- Low-bit FP introduces **fine-grained, high-precision dequant** with significant overhead

Dequantization (NVFP4)

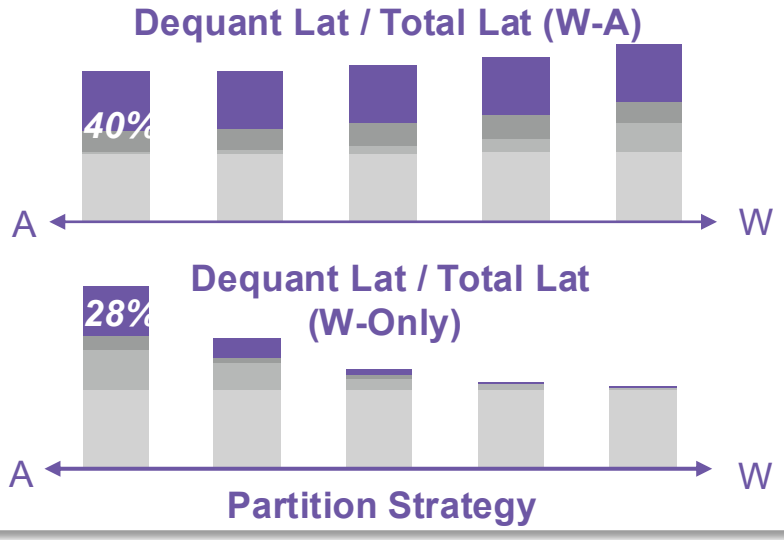
- High Frequency: fine-grained QGroups
- High Precision partial sums and scales



Increase Latency by ~35%

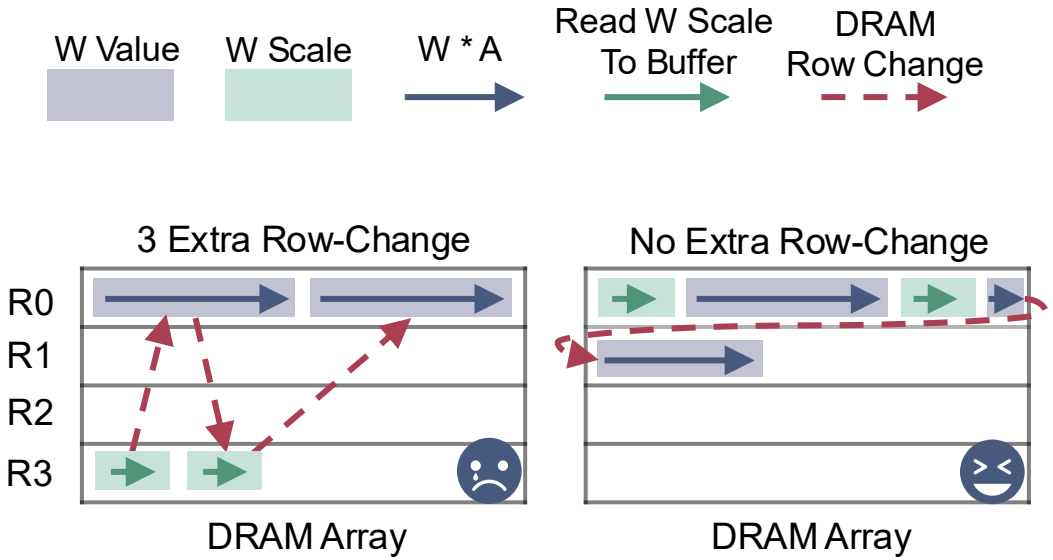
Dequant differs between quant. methods

- W-A: cannot be optimize by better partition



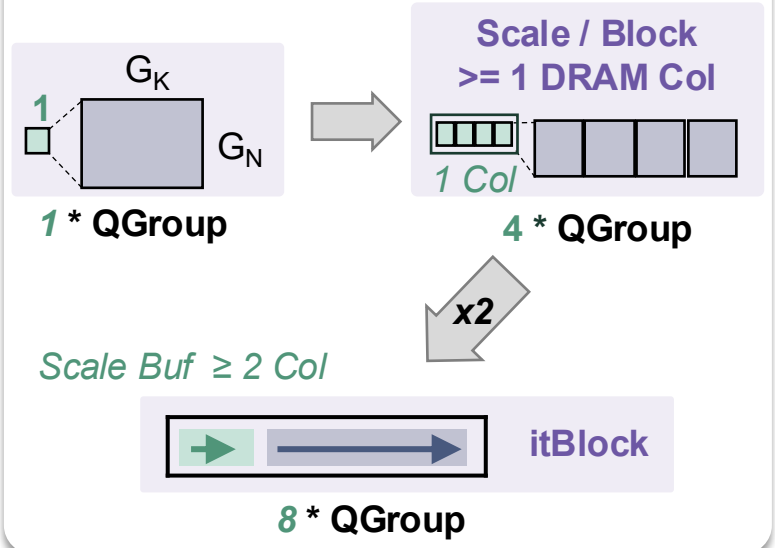
Technique 1: Scale-Value Interleaving Layout

- **Core Idea:** store scales close to the corresponding values (Interleaving)
- Reduce #RowChange by **2.67x**



Construct interleaving block (itBlock) accord to DRAM column Size

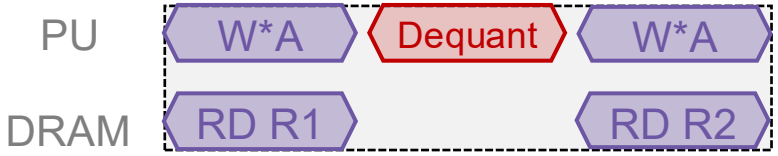
- **Determine the size of itBlock**



Technique 2: Instruction Reorder

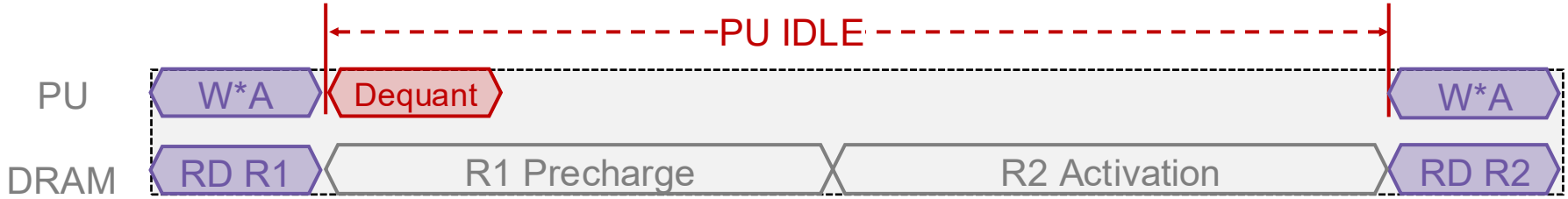


- **Case1**：Dequant introduce extra latency（***~90%*** **of all Dequant**）

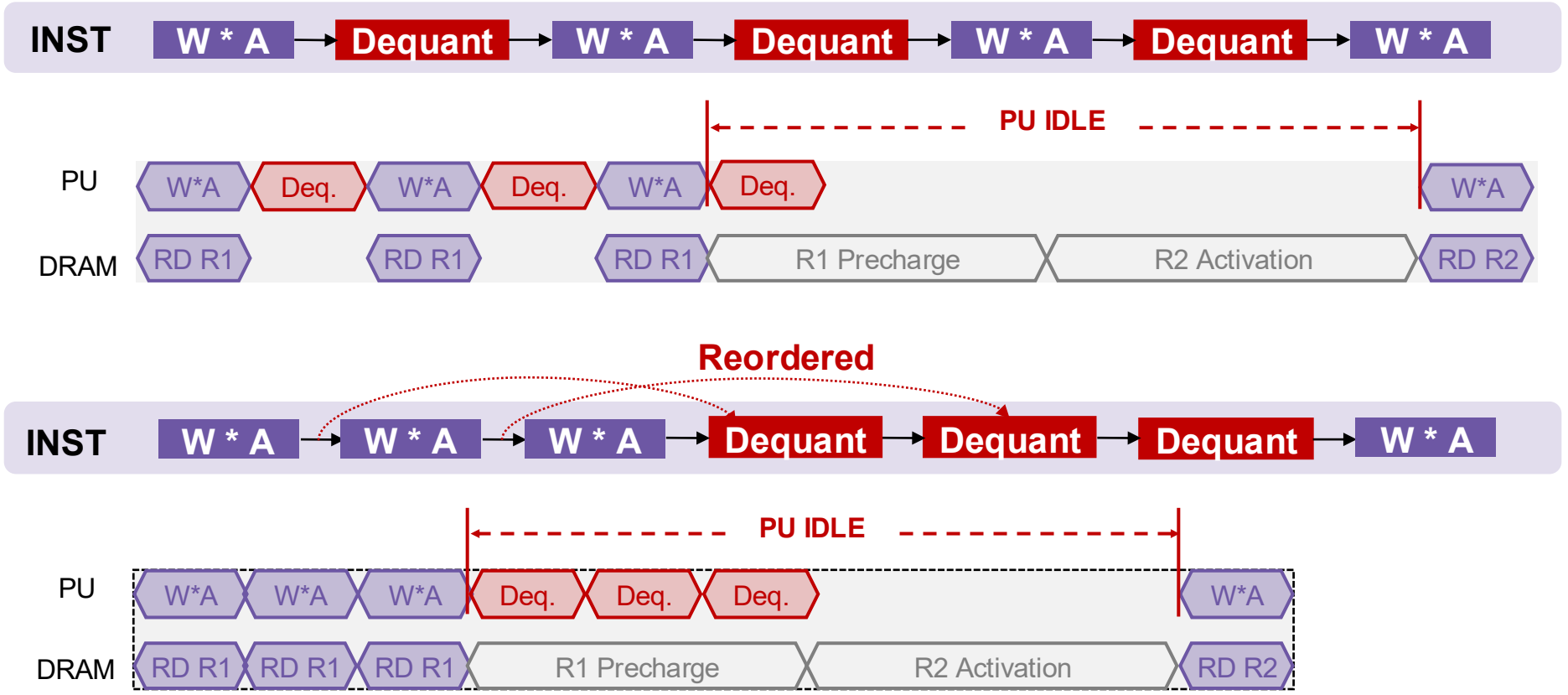


Most Dequant cannot be naturally hided

- **Case2**：Dequant hided in DRAM row change（***~10%*** **of all Dequant**）

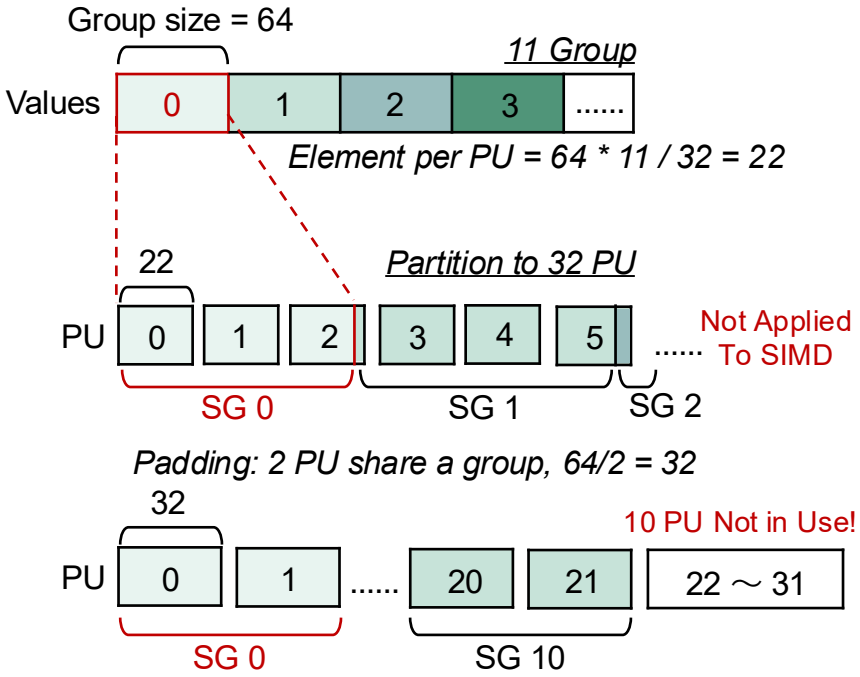


Technique 2: Instruction Reorder

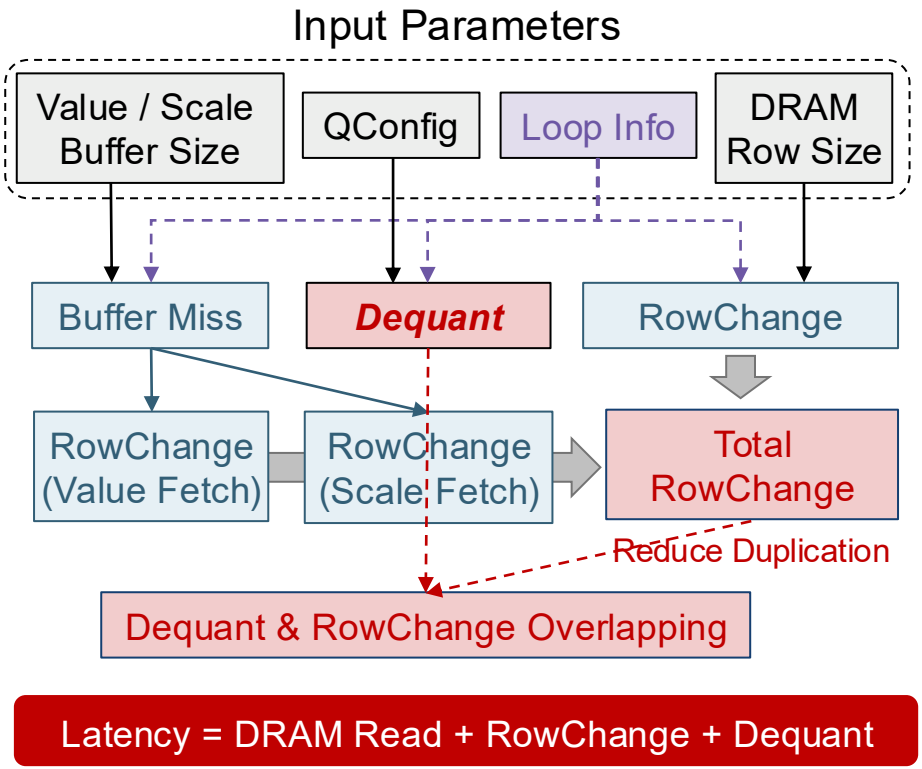


Technique 3: Compilation Strategy Space Exploration

- **Pruning Hints:** e.g. Conflict between partition & QGroup size



- **Cost Model**



Results - Experiment Setup

Experiment Configs

- Workload: LLaMA2 - 7/13/34B
- Quantization Configs

Quantization Method	W-A, W-Only
Precision(Data,Scale)	(4, 8), (8, 16)
Q1-Q6 (G_N, G_K)	(1, 16), (1, 32), (16, 16), (32, 32), (64, 64), (128, 128)

- Hardware Parameters

DRAM Parameter	Value	PU Parameter	Value
# GDDR6 Chip	32	FP4 Throughput	51.2 GFLOPs
# Channel / Chip	2	FP8 Throughput	25.6 GFLOPs
# Bank / Channel	16	FP16 Throughput	12.8 GFLOPs
# PU / Channel	16	FP32 Throughput	6.4 GFLOPs
Data Rate / pin	12 Gbps	Frequency	0.4 GHz
Total External BW.	1.5 TB/s	SRAM Buffer Size	20×32 Bytes
Total Capacity	32 GB	Total PU Num.	1024

Timing Parameters (from DRAMSim3 [38]): $t_{CK} = 0.66ns$, $t_{RCD} = 24$, $t_{RP} = 24$, $t_{CCDL} = 4$, $BL = 16$, $t_{CL} = 24$

Baseline Strategy

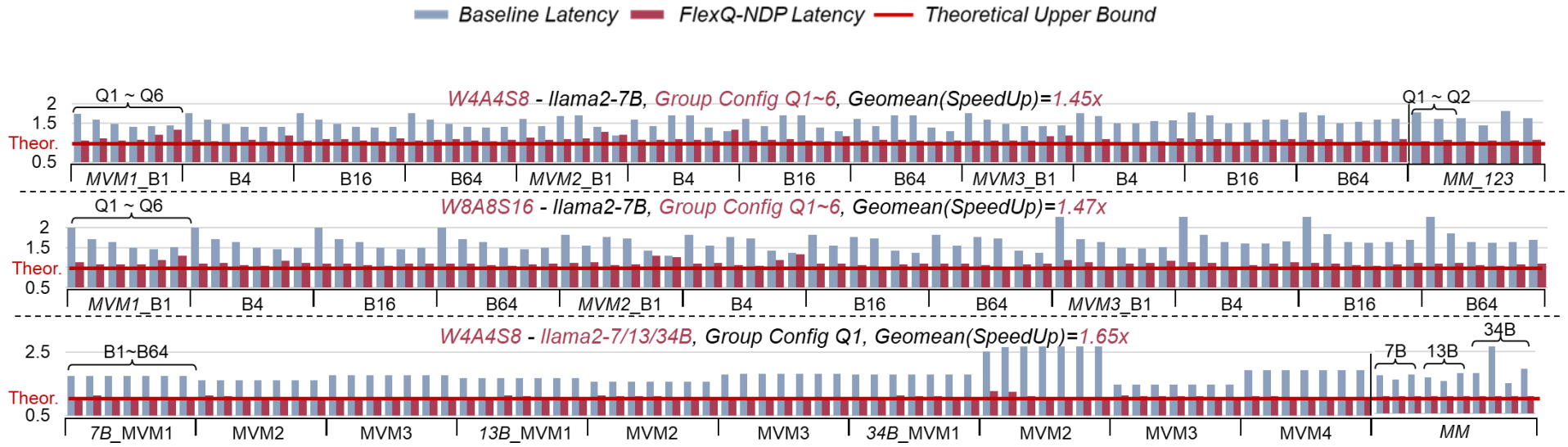
- Buffer allocated by the Value-to-Scale ratio
- Partition along the N dimension

Extra Latency in End-to-End Experiment

- Activation requires Online Quantization (W-A)
- Weight rewrite when exceed DRAM capacity

Experimental Results

- Operator: MVM + MM + batched MVM
- Latency normalized to theoretical lower bound (assuming PU keeps working）
- Achieve up to **3.29x** speedup

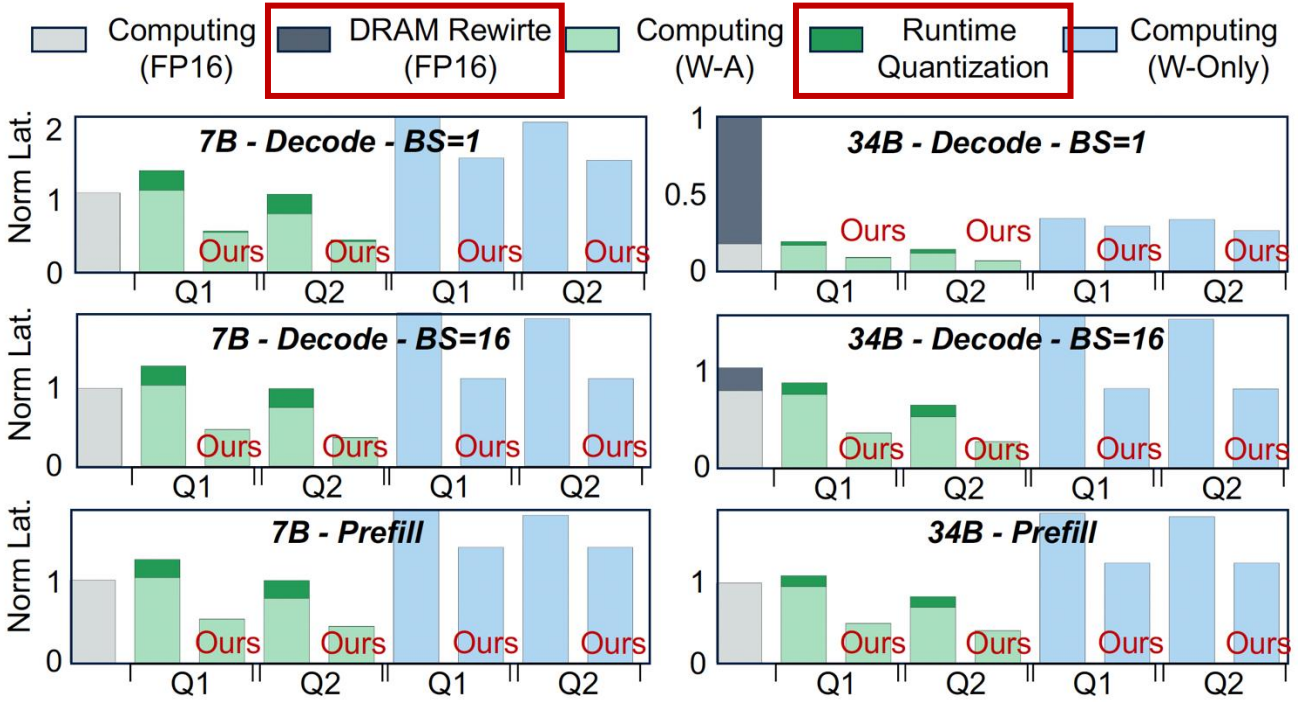


Experimental Results

- Latency normalized to FP16 model latency

- v.s. *FP4 Baseline*
- *Prefill - 2.36x*
- *Decode - 2.73x*

- v.s. *FP16*
- *Prefill - 2.47x*
- *Decode - 15x*



Future Directions

1. NDP Compilation Toolchain Validation

- Access to commercialized Prototype / Emulator



UPMEM 1st GEN.



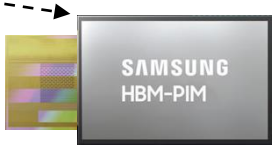
UPMEM 2nd GEN.

SK hynix LPDDR-AiM

Hynix LPDDR-AiM

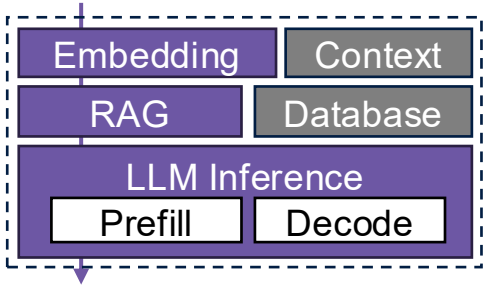


Hynix GDDR6-AiM



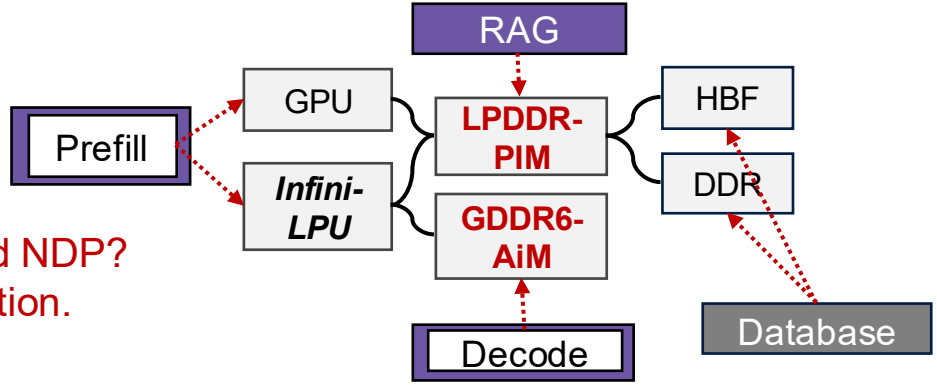
Samsung HBM-PIM

2. Application of NDP in Agentic Workflow



AI Agent

- Where do we need NDP?
- Mapping Optimization.



清華大學深圳国际研究生院
Tsinghua Shenzhen International Graduate School

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

UniNDP *FlexQ-NDP*

Thank You!

Zhenhua Zhu

Assistant Professor

Shenzhen International Graduate School, Tsinghua University; Dept. of ECE, HKUST

zhuzhenhua@sz.tsinghua.edu.cn, eezhuzh@ust.hk



2026/6/29

NICS EFC