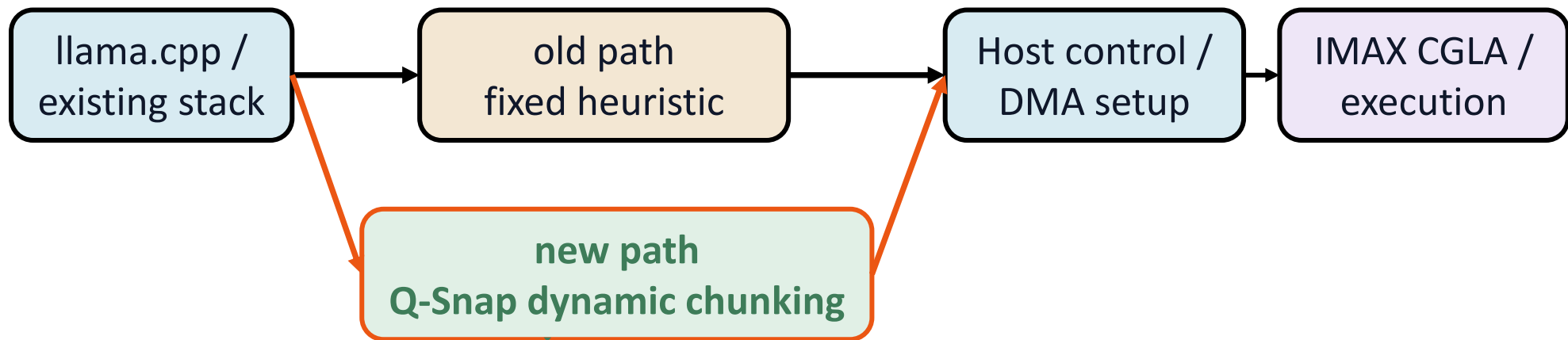


Q-Snap: Quantization-Aware Dynamic Chunking for LLM Execution on a CGLA

Takuto Ando, Ayumu Takeuchi, Yu Eto, Yoshifumi Munakata, Yasuhiko Nakashima
Nara Institute of Science and Technology (NAIST)



The key change is replacing chunk control with a CGLA-aware policy

AI's Energy Problem & Our Solution

Problem: AI relies on power-hungry GPGPUs.

GPGPU: High Flexibility 😄 , Low Efficiency 😞

Specialized ASIC: High Efficiency 😄 , Low Flexibility 😞



Solution: CGRA - Bridging the Gap

We focus on CGRAs (Coarse-Grained Reconfigurable Arrays) as an architecture to solve the trade-off.

- ▲ **Efficiency:** They achieve ASIC-like efficiency by directly mapping dataflows.
- ▲ **Flexibility:** They retain flexibility by dynamically reconfiguring the datapath.

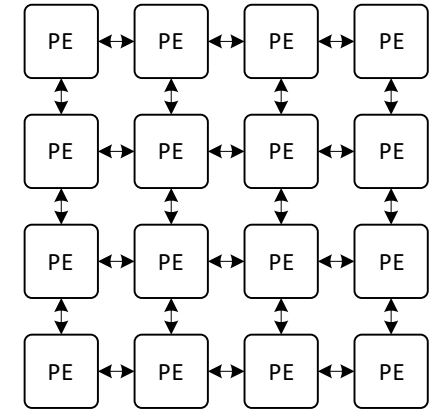
Our CGLA “IMAX” ...What’s this? 🤔

2

Conventional CGRA

Concept: Reconfigurable at the PEs level on a 2D array

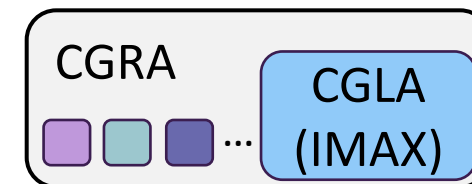
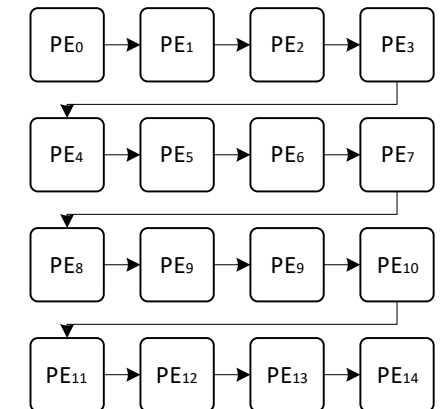
Challenge: Complex routing algorithms lead to long compilation times



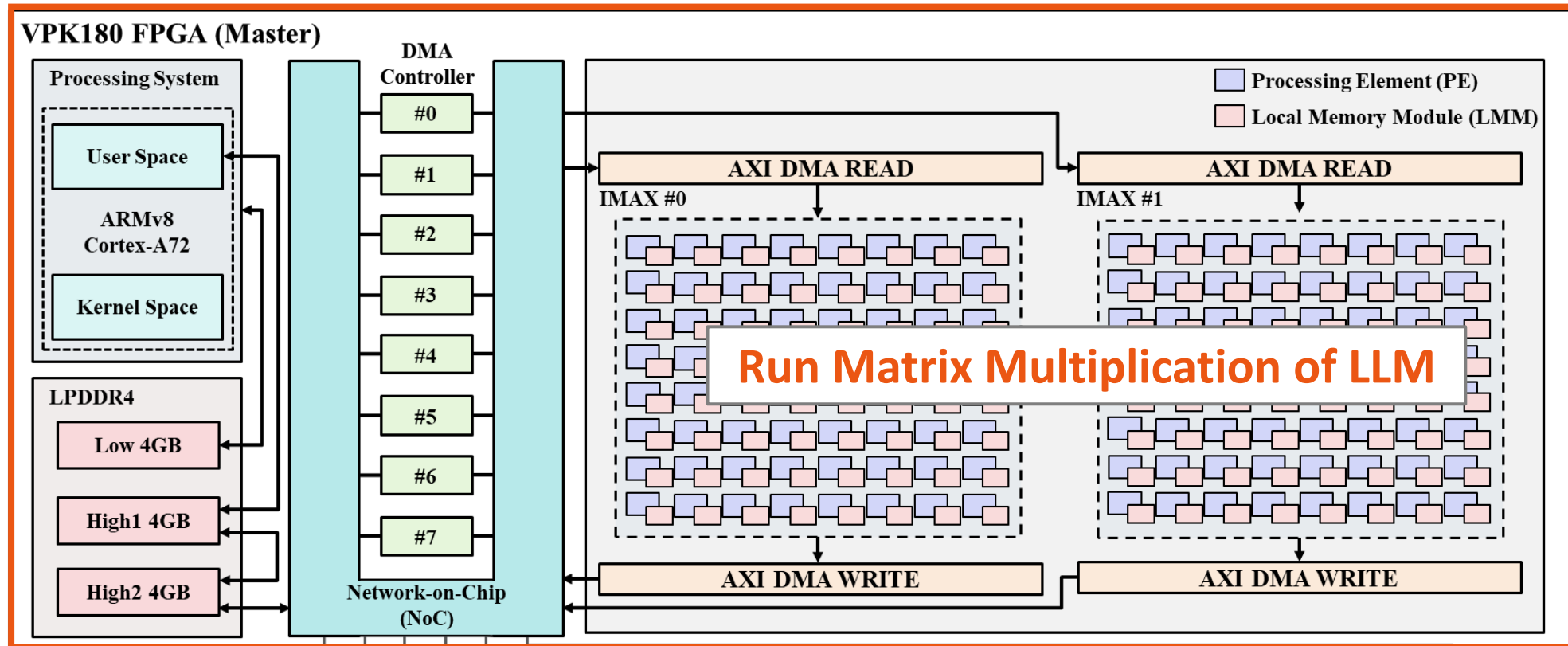
IMAX Architecture (CGLA: Coarse-Grained Linear Array)

Concept: PEs are arranged in a linear structure, fast compilation

Design: Interleaves 64 PEs with 512KB of local cache memory to minimize slow main memory access



IMAX Configuration and LLM



Most processing time is spent on Matrix Multiplication

This study keeps the existing LLM software stack and changes only the chunking policy

Software-Stack Mismatch on a CGLA

Replace fixed chunking with a CGLA-aware runtime policy

Existing stack: fixed chunks



Many fine-grained launches DMA latency stays visible

Effects

- More kernel launches
- DMA latency stays visible
- Especially harmful in Prefill phase

Proposed: Q-Snap



Choose the safe max for better burst efficiency

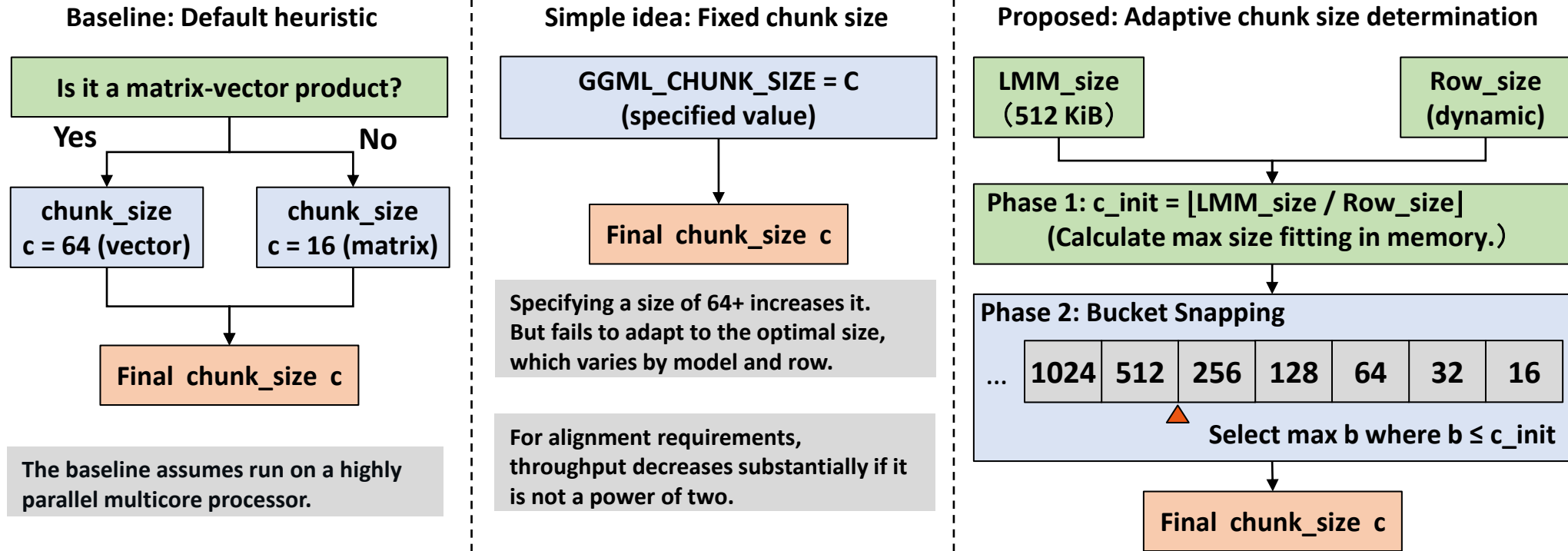
Effects

- Longer burst execution
- Transfer latency is easier to hide
- More stable efficiency than a fixed constant

Our Objective

Implement Q-Snap as a software-stack algorithm that dynamically changes chunk size according to quantization blocks and local memory.

Proposed: Adaptive chunk size determination



Q-Snap is a runtime algorithm for adaptive chunk sizing.

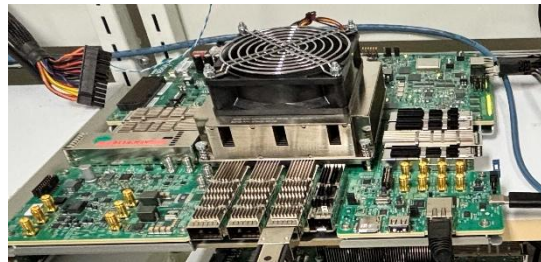
This turns chunking into a hardware-aware runtime decision.

Experimental Setup

Workload: Qwen3 0.6B / 1.7B / 8B in Q3_K. llama.cpp-based stack with IMAX offload.

Objective: Compare the original fixed-chunk runtime against Q-Snap.

Input tokens: 32



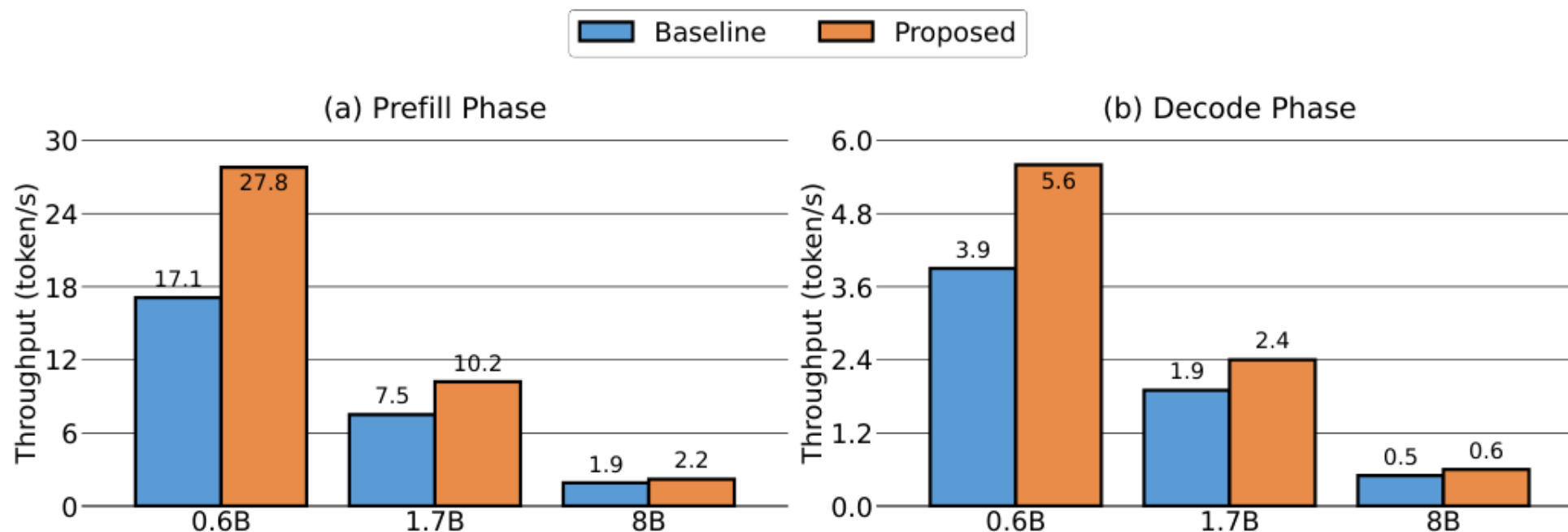
Output tokens: 32

VPK180 FPGA (145 MHz)

Two key metrics for **the end-to-end task**:

- ▲ **Performance:** Prefill / Decode throughput.
- ▲ **Energy Efficiency:** The **Power-Delay Product** (PDP), calculated as $Latency (s) \times Power (W)$.

Throughput Improvement



Throughput comparison between baseline and proposed methods

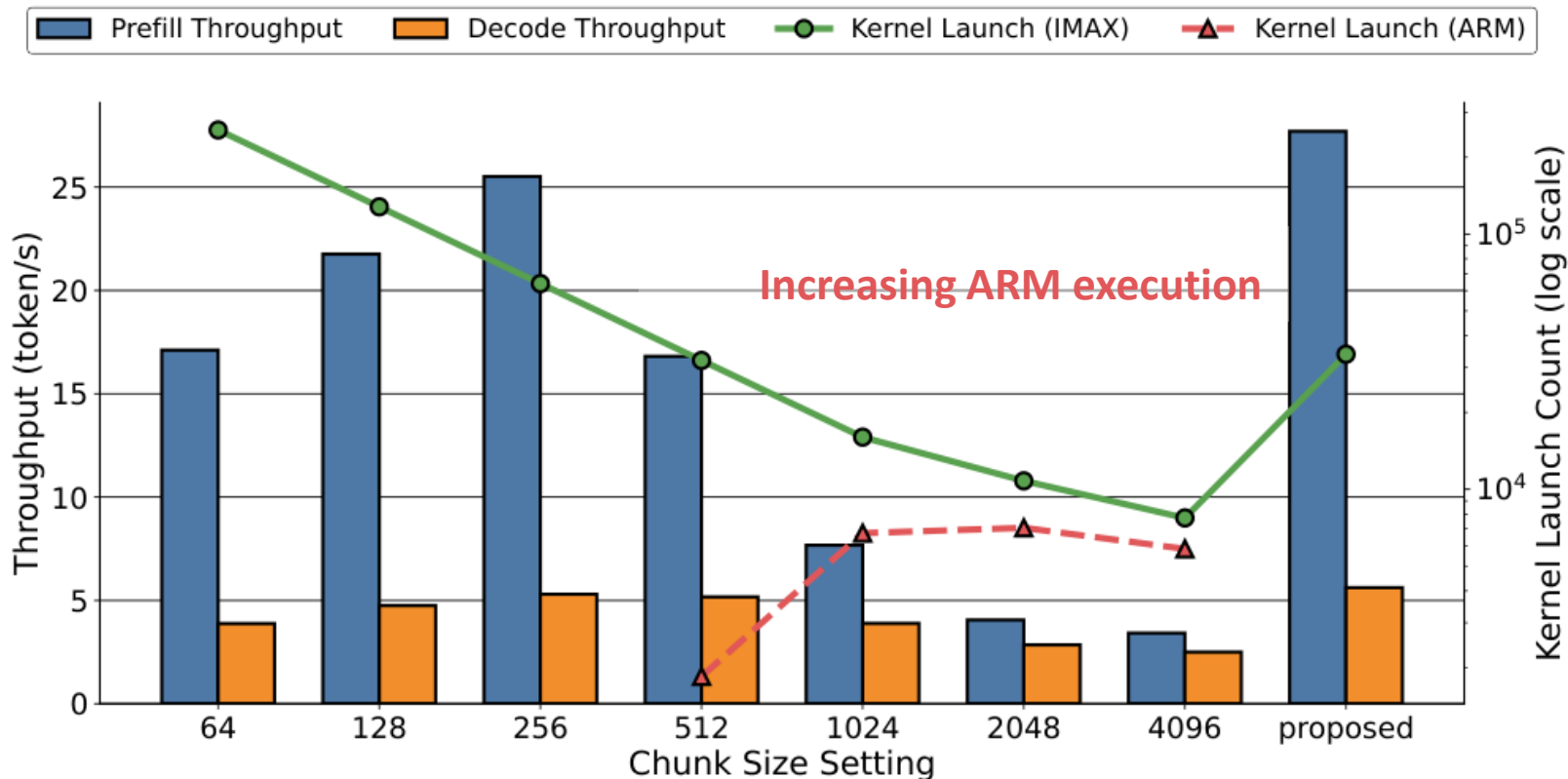
Prefill gains the most:

- Fixed chunk baseline: 17.11 tokens/s
- Q-Snap: 27.70 tokens/s
- Speedup: 1.62x

Decode also improves:

- Fixed chunk baseline: 3.88 tokens/s
- Q-Snap: 5.62 tokens/s
- Speedup: 1.45x

Adaptive Chunking Matters



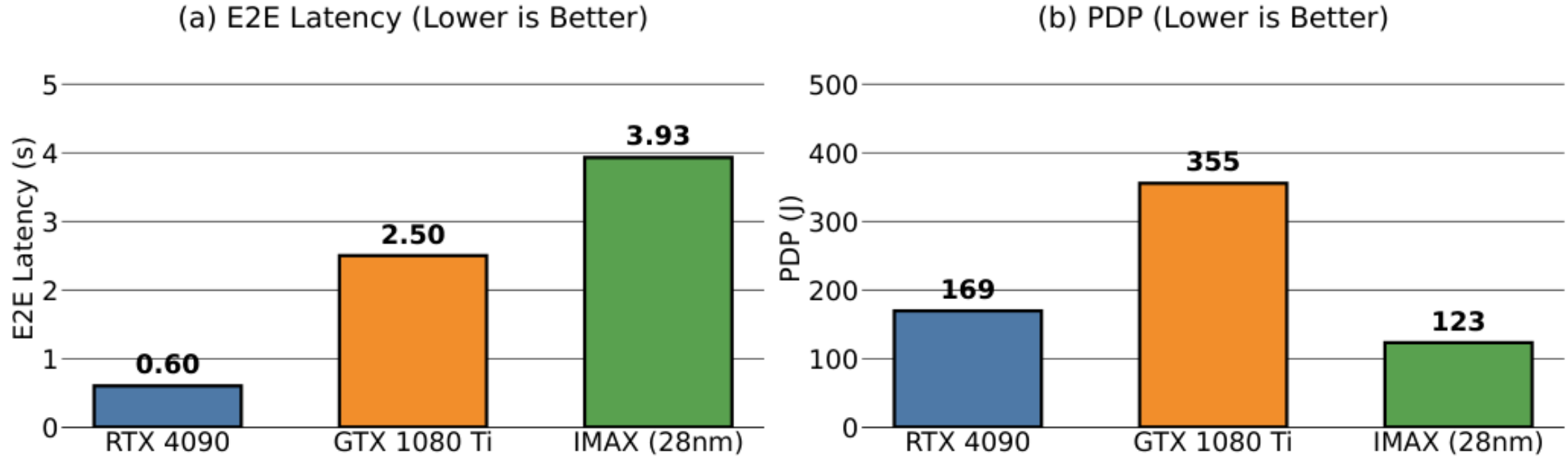
Sensitivity analysis of the impact of chunk size setting on throughput and kernel launch count

Why not just increase the fixed chunk?

A larger constant can reduce launches, but it still ignores layer-dependent row size.

Some settings overflow LMM and lose performance.

Energy Efficiency (PDP)



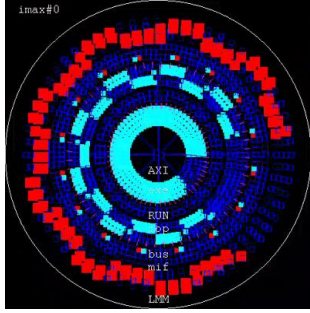
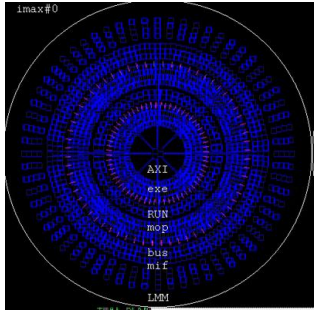
Latency is worst with IMAX

BUT IMAX reaches 123 J PDP. 27% better than RTX 4090, 65% better than GTX 1080 Ti.

This shows that runtime scheduling affects practical energy efficiency.

Summary

<u>Problem</u>	The existing LLM stack reused fixed chunking designed for CPUs and GPUs, which is inefficient on IMAX.
<u>Our Approach</u>	We implemented Q-Snap, which dynamically changes chunk size using quantization-block and local-memory information.
<u>Key Results</u>	Q-Snap improves throughput over the fixed baseline and reaches 123 J PDP in the 28 nm ASIC projection.
<u>Conclusion</u>	The contribution is a CGLA-aware software stack for LLM inference, not just a kernel-level optimization.
<u>Future Work</u>	Extend the policy to more formats, larger models, and broader system configurations.



IMAX execution visualization DEMO

Thanks!



LENZO Page



This paper

Do you have any **questions**?



Takuto ANDO



Yu ETO



Ayumu TAKEUCHI



Yoshifumi MUNAKATA



Yasuhiko NAKASHIMA

appendix

Dataflow within a Single PE Unit

1. Input Registers:

Hold data for the current cycle (n).

2. Pipelined Execution:

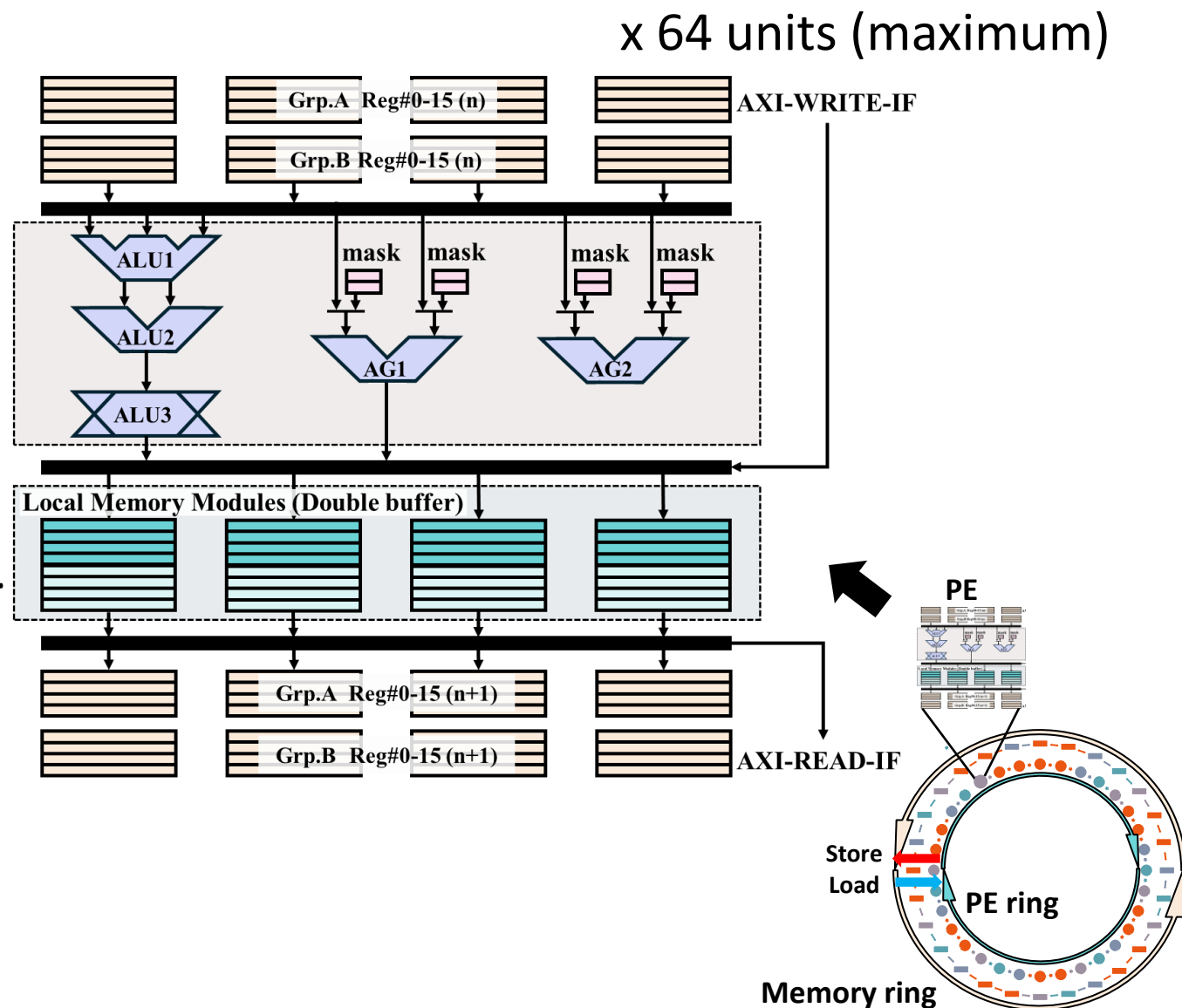
Data flows through a 3-stage ALU and AGs(address generation).

3. LMM (Load and Store)

Load or store data in 512KB of local memory, overlapping computation with data transfers.

4. Output to Next Stage:

Results are forwarded to the next PE's registers (n+1).



Highlights

World's First

Successfully demonstrated **Whisper ASR*** kernels on a **CGRA**.

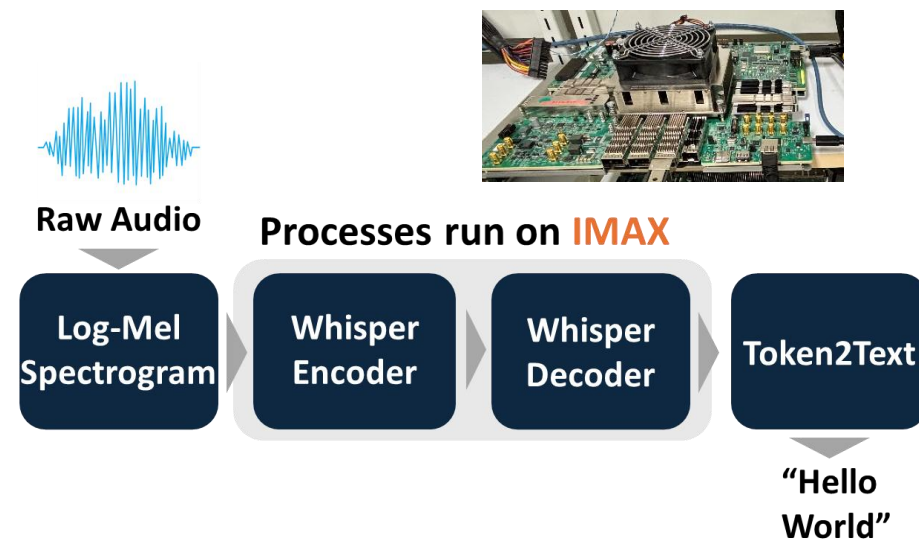
Superior Energy Efficiency (PDP)

Outperforms existing GPUs (ASIC Prediction)

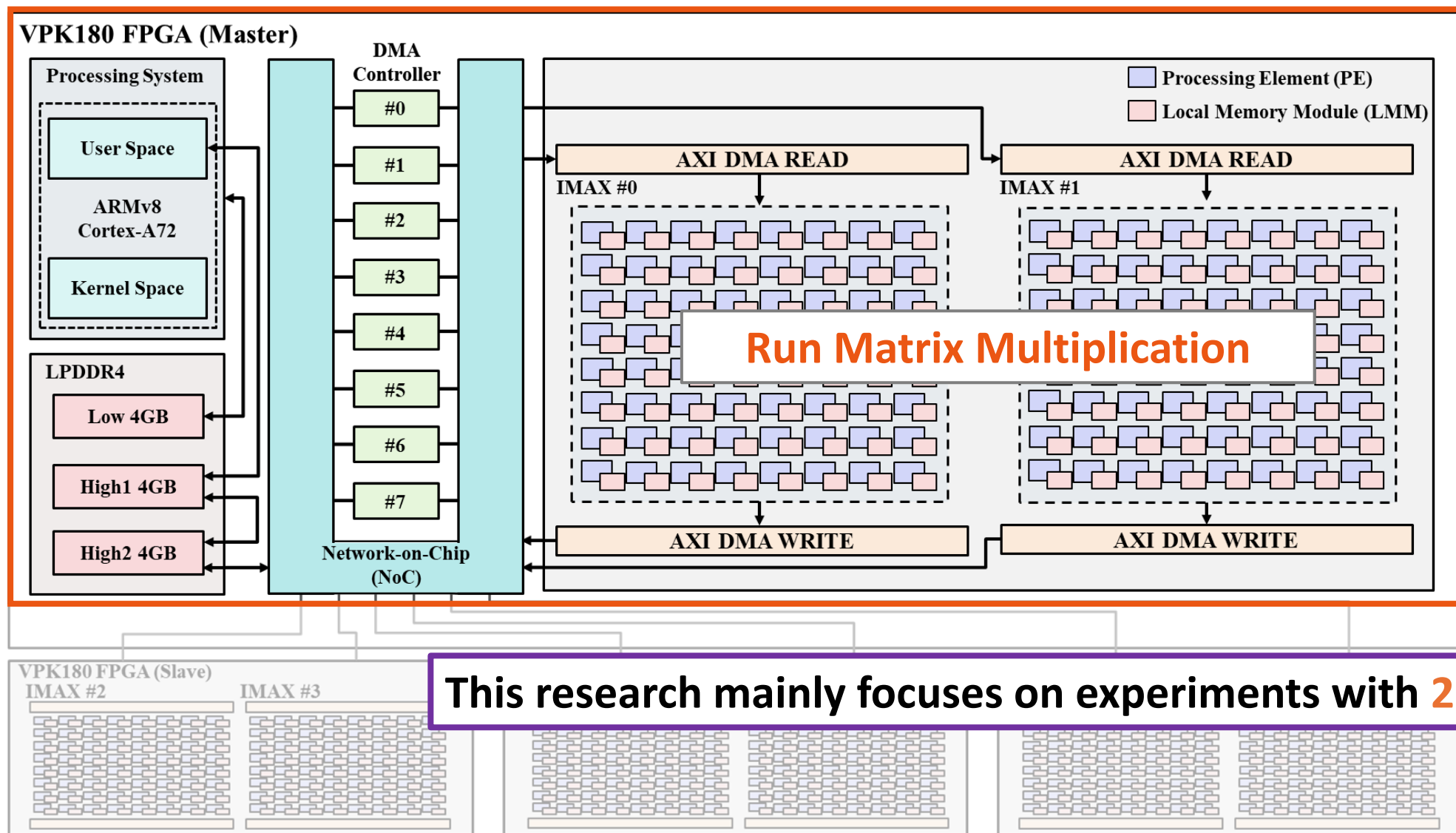
- **1.90x** vs. Edge GPU (Jetson AGX Orin)
- **9.83x** vs. High-end GPU (RTX 4090)

Conclusion

CGLA shows potential as a **sustainable ASR platform for the edge**.



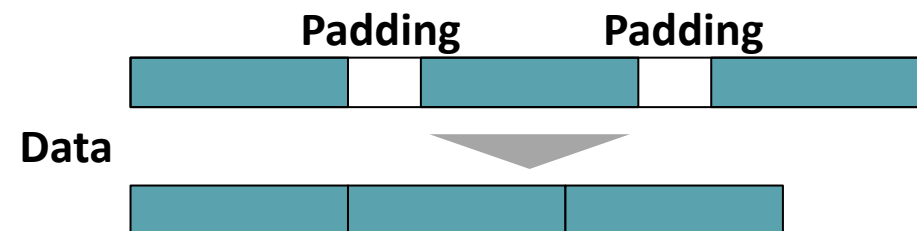
IMAX Configuration



Data Transfer & Memory

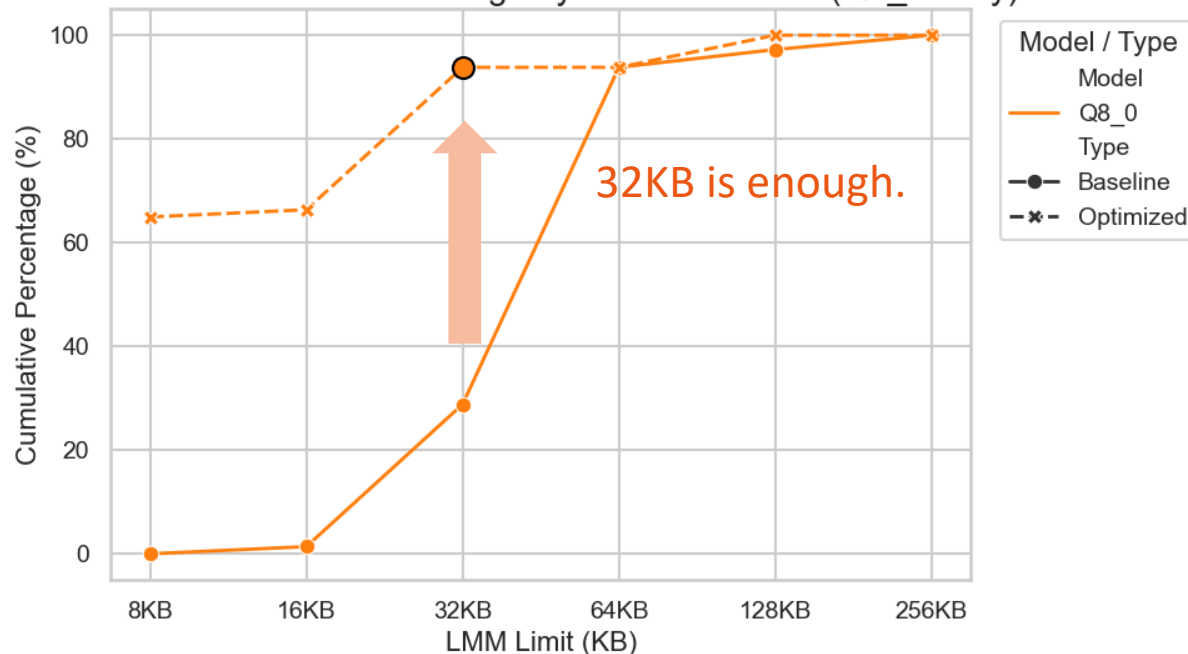
Challenge: Data tensors contain **significant padding** for alignment

Solution: Pre-Transfer **Padding Removal**

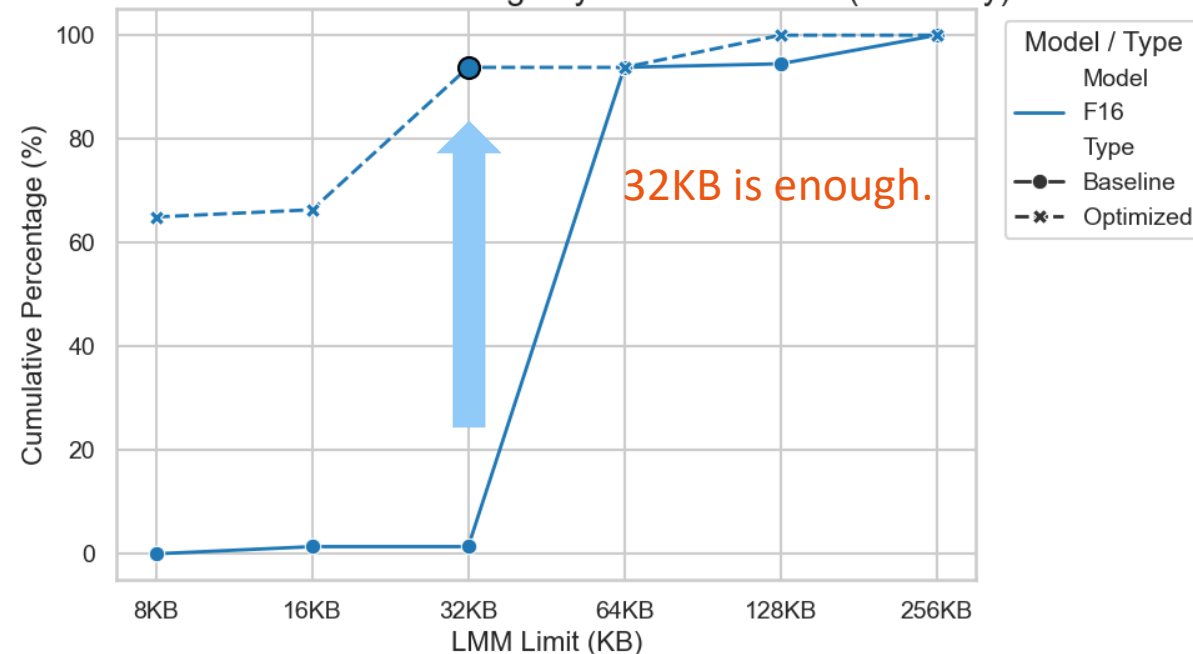


The Host CPU **strips all padding** before DMA transfer

Cumulative Model Percentage by LMM Size Limit (Q8_0 Only)



Cumulative Model Percentage by LMM Size Limit (F16 Only)



Experimental Setup



VPK180

TABLE III: Physical specifications and performance comparisons of various devices

Device	CPU	Cores	Chip area (mm ²)	Process node	Operating frequency	Memory	Power ^c (W)
ARM Cortex-A72 (on Versal)	-	2	-	7 nm	1400 MHz	8 GB DDR4	0.6485
IMAX3 (Xilinx VPK180)	ARM Cortex-A72	64 ^a	-	7 nm	145 MHz	8 + 4 GB DDR4 ^b	180
IMAX3 (28nm)	-	64 ^a	14.6	28 nm	840 MHz	-	0.647 or 1.32
Jetson AGX Orin 32GB	Arm® Cortex-A78AE	1792	200	8 nm	930 MHz	32 GB DDR5	15 ^d
NVIDIA RTX 4090	Xeon W5-2455X	16384	608	5 nm	2520 MHz	24 GB DDR6	450

^a The number of cores for IMAX3 refers to the number of PEs per lane. ^b 8 GB DDR4 for OS buffer and 4 GB DDR4 for DMA buffer.

^c IMAX3 (28 nm) is an estimated value, references for other devices are from Cortex-A72 [27], Jetson AGX Orin 32GB [28], NVIDIA RTX 4090 [29].

^d This device has multiple power consumption modes (15–45 W), and we executed in the lowest power consumption mode.

IMAX Platform:

Performance and power were projected for a 28nm ASIC implementation, allowing for a realistic 6x frequency increase to 840 MHz(FPGA:145MHz).

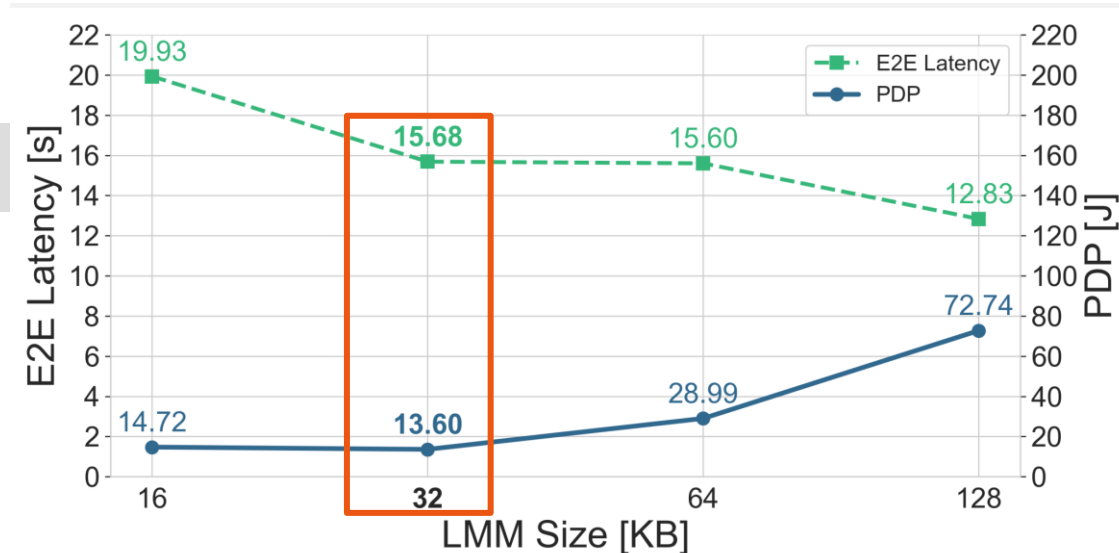
Impact of LMM Size (PDP)

Why did we choose 32KB LMM?

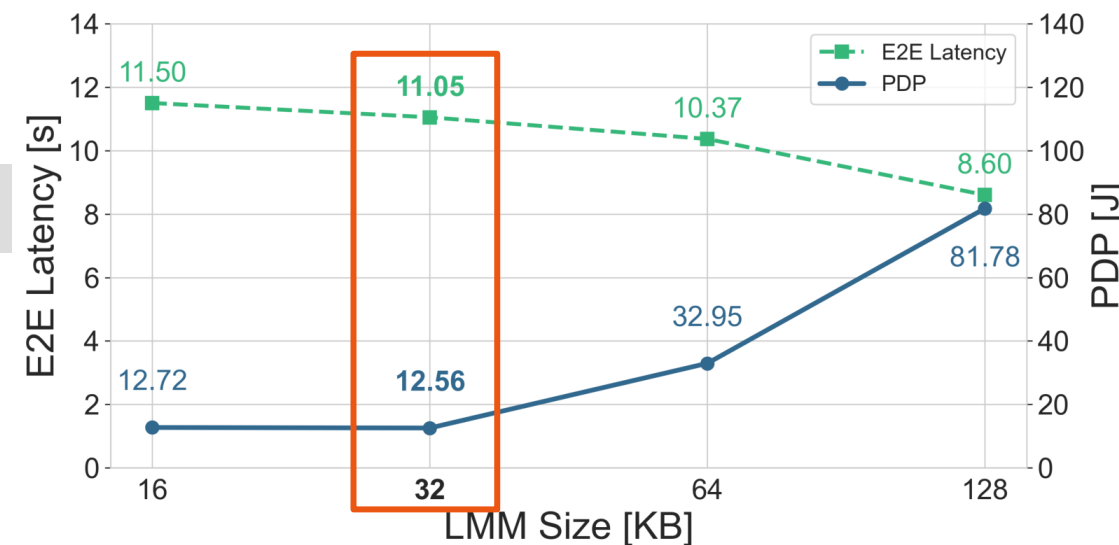
- ▲ **16 KB:** Too small. Data spills to CPU, drastically worsening PDP.
- ▲ **32 KB:** The optimal point with **minimal PDP**.
- ▲ **64~KB:** Static power consumption increases, worsening total PDP.

Point:

Simply increasing LMM(on-chip memory) is not always better.



(a) FP16 model inference



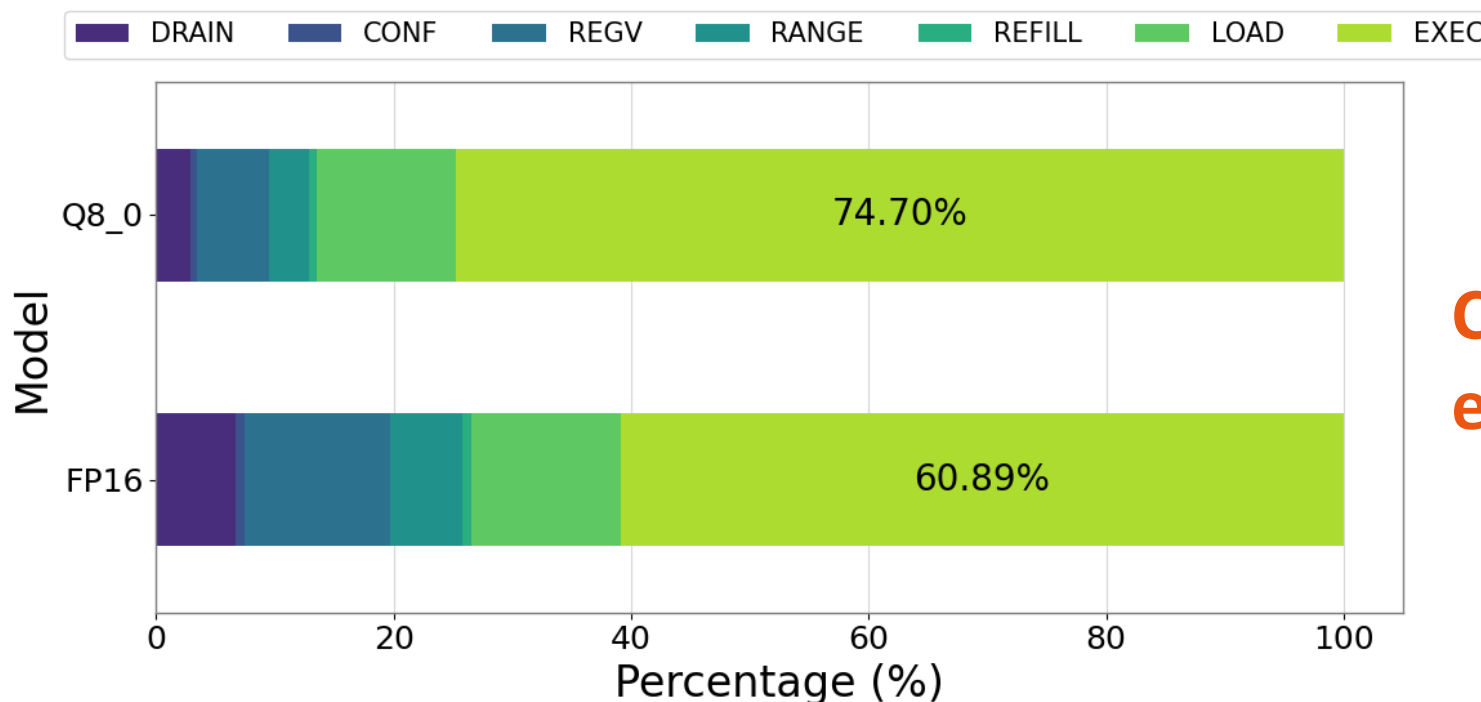
(b) Q8_0 model inference

E2E latency and PDP comparison by LMM size

Compute Efficiency & Scalability

Achieved Compute-Bound Status:

- AI workloads are often *memory-bound* (waiting for data).
- Our implementation: **74.70%** of time is "EXEC" (computation).

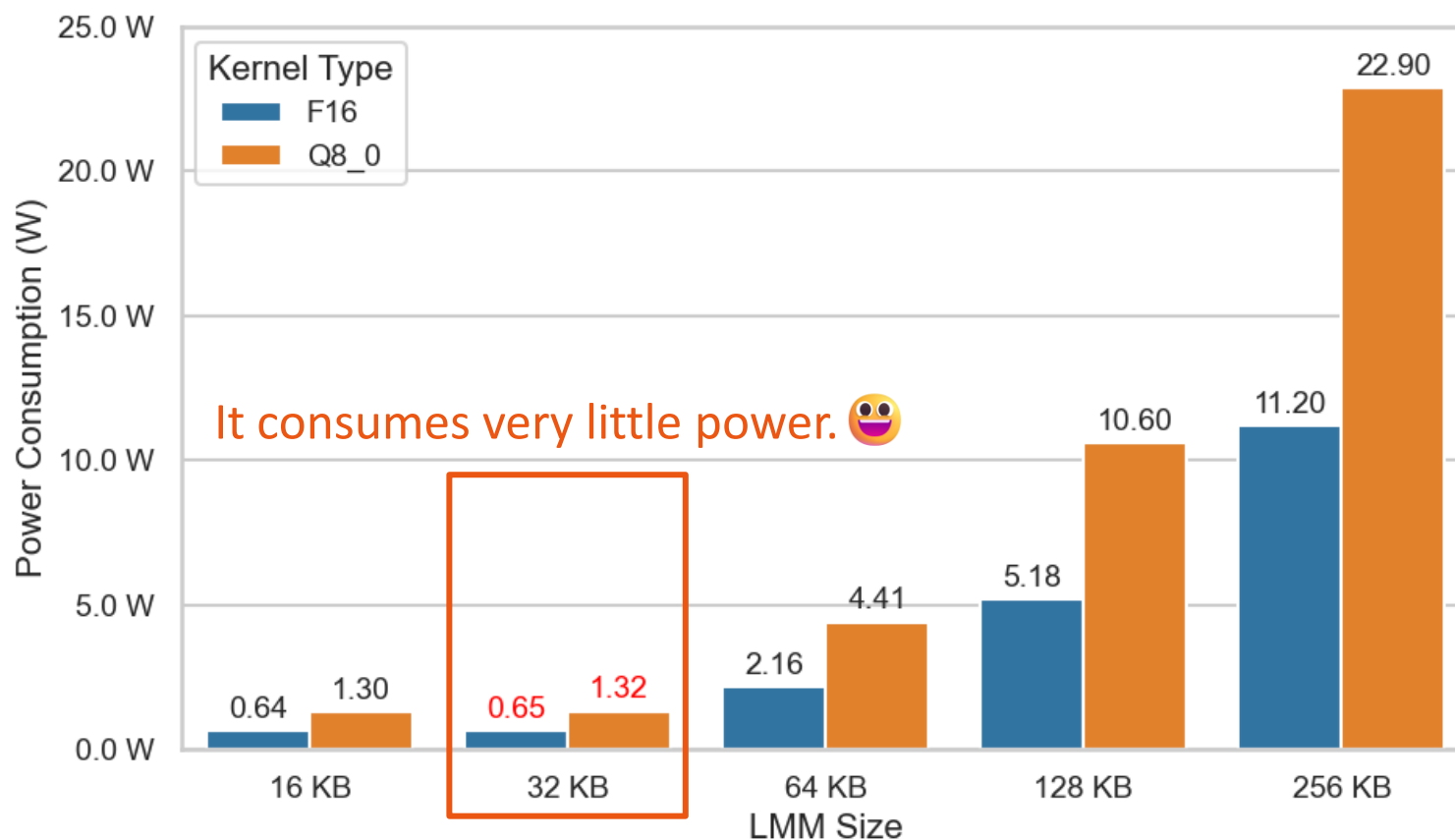


Computationally
efficient ! 🤖

What's the best Local Memory Size?

Sweet Spot

- ▲ The **power consumption** of IMAX is highly **dependent on the size of LMM**.
- ▲ **32KB** is good enough, and it uses very little power.

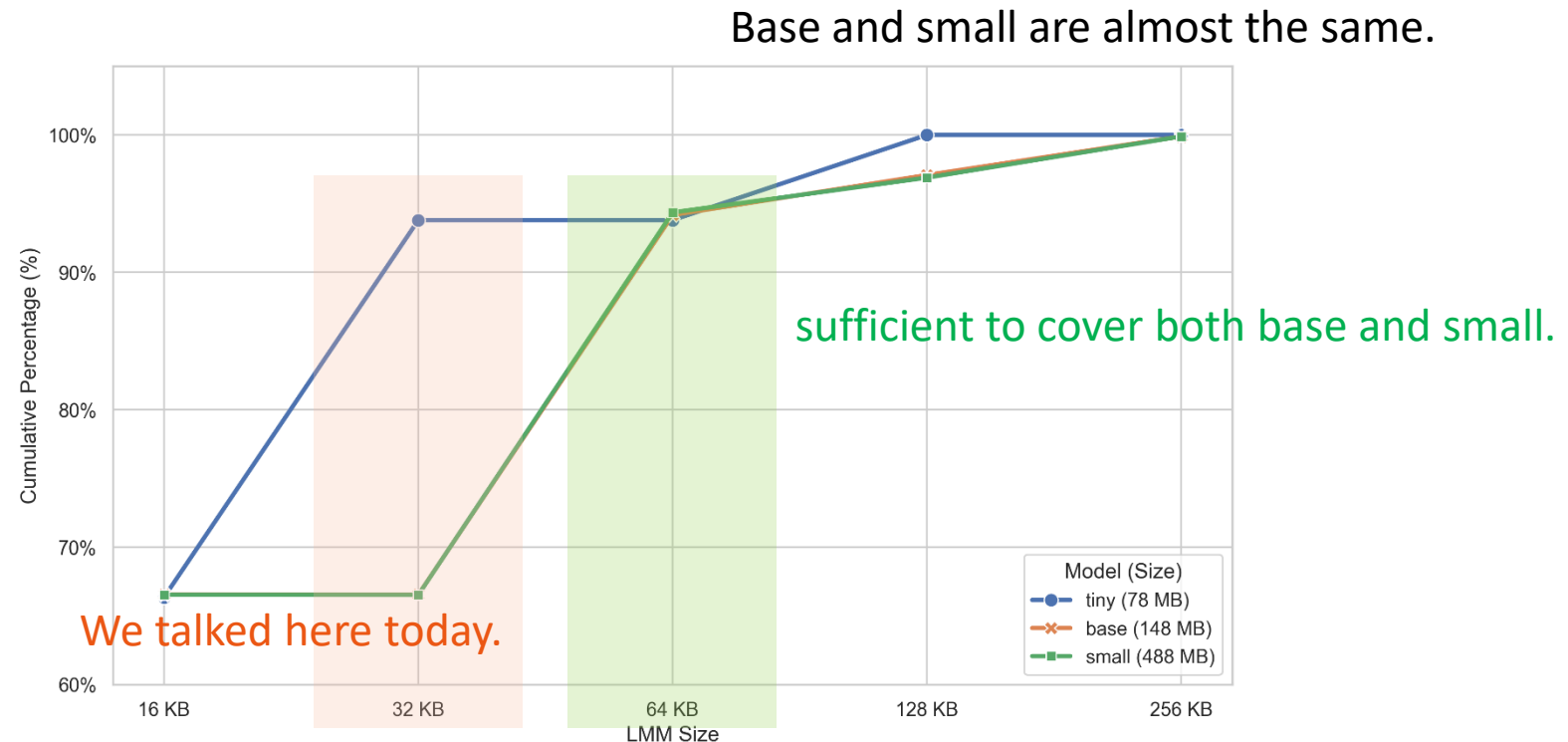


Power consumption by LMM size for FP16 and Q8_0 kernels

Future Work 1: Scalability to Larger Models

This study used the **tiny.en model**.

We confirmed that a 64KB LMM can accommodate >94% of the base and small model kernels.



Kernel coverage percentage by model and LMM size

Future Work 2: exploring kernel

Currently supported kernels:

FP32/16, INT4/8/16/32, Q80, Q40, Q2K, Q3K, Q6K

Going forward

we aim to implement kernels like **TQ1** and **TQ2**, primarily focusing on low-bit representations like Bitnet.

This will lead to a **reduction in memory footprint**.

Future Work 3: Kernel modification

- Implement the kernel with a small number of PEs
- extend burst operations.
- Currently, I can't think of any interesting ideas.

```
#define f16(r, cb, cb00, init0, init1, init2, init3, dst0, dst1, dst2, dst3) \
    mop(OP_LDR, 0, &BR[r][0][1], src0_row_base_p8[cb], (U11)iofs, MSK_W1, src0_row, SRC0_ROW_LEN, 0, Force0, (U11)NULL, SRC0_ROW_LEN); /* dismissible */\
    mop(OP_LDR, 3, &BR[r][0][0], src0_row_base_p0[cb], (U11)iofs, MSK_W1, src0_row, SRC0_ROW_LEN, 0, Force0, (U11)NULL, SRC0_ROW_LEN); /* unaligned */\
\
    exe(OP_NOP, &r10, BR[r][0][0], EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, 0x0000fffff0000ffffLL, OP_SLL, 2LL); /* x.d fp16->fp32 */\
    exe(OP_NOP, &r11, BR[r][0][0], EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, 0xffff0000ffff0000LL, OP_SRL, 14LL); /* x.d fp16->fp32 */\
    mop(OP_LDR, 0, &BR[r+1][0][1], src1_col_base_p8[cb], (U11)cofs, MSK_W1, i_m0B[LANE], SRC1_COL_LEN_TOTAL, 0, Force1, (U11)NULL, SRC1_COL_LEN_TOTAL); /* dismissible */\
    mop(OP_LDR, 3, &BR[r+1][0][0], src1_col_base_p0[cb], (U11)cofs, MSK_W1, i_m0B[LANE], SRC1_COL_LEN_TOTAL, 0, Force1, (U11)NULL, SRC1_COL_LEN_TOTAL); /* unaligned */\
\
    exe(OP_NOP, &r12, BR[r+1][0][0], EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, 0x0000fffff0000ffffLL, OP_SLL, 2LL); /* x.d fp16->fp32 */\
    exe(OP_NOP, &r13, BR[r+1][0][0], EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, 0xffff0000ffff0000LL, OP_SRL, 14LL); /* x.d fp16->fp32 */\
\
    mop(OP_LDWR, 3, &BR[r+3][0][1], (U11)i_f16[LANE], (U11)r10, MSK_W0, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][0][0], (U11)i_f16[LANE], (U11)r10, MSK_W1, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][1][1], (U11)i_f16[LANE], (U11)r11, MSK_W0, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][1][0], (U11)i_f16[LANE], (U11)r11, MSK_W1, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][2][1], (U11)i_f16[LANE], (U11)r12, MSK_W0, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][2][0], (U11)i_f16[LANE], (U11)r12, MSK_W1, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][3][1], (U11)i_f16[LANE], (U11)r13, MSK_W0, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
    mop(OP_LDWR, 3, &BR[r+3][3][0], (U11)i_f16[LANE], (U11)r13, MSK_W1, (U11)i_f16[LANE], 65536, 0, 0, (U11)NULL, 65536); /* x.d fp16->fp32 */\
\
    exe(OP_FMA, &dst0, init0, EXP_H3210, BR[r+3][0][1], EXP_H3210, BR[r+3][2][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* sdata,wdata */\
    exe(OP_FMA, &dst1, init1, EXP_H3210, BR[r+3][0][0], EXP_H3210, BR[r+3][2][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* sdata,wdata */\
    exe(OP_FMA, &dst2, init2, EXP_H3210, BR[r+3][1][1], EXP_H3210, BR[r+3][3][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* sdata,wdata */\
    exe(OP_FMA, &dst3, init3, EXP_H3210, BR[r+3][1][0], EXP_H3210, BR[r+3][3][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* sdata,wdata */\
#endif
```

Current kernel code

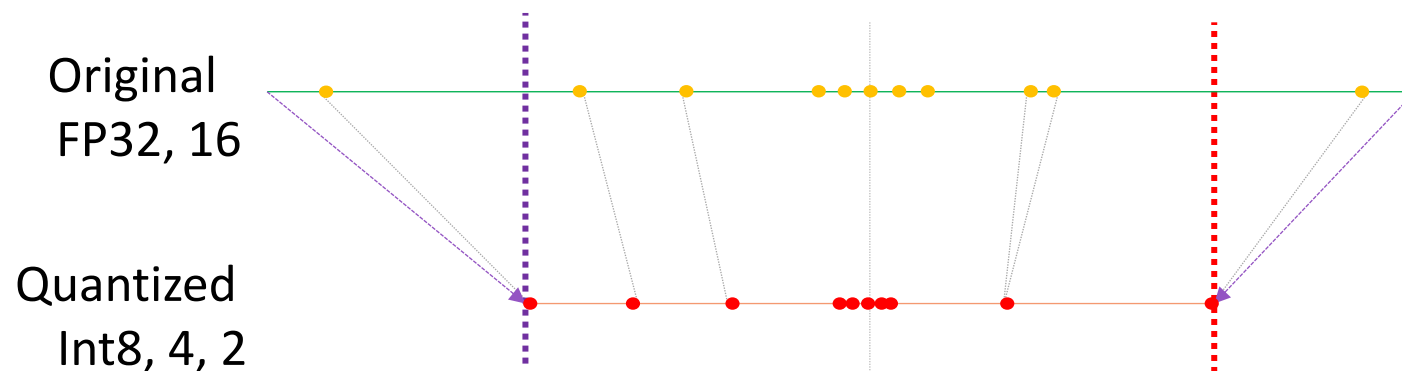
Experimental Setup & Fairness

Framework:

Identical version of `ggml.cpp` used across all platforms (GPU, Edge AI, FPGA).

Models:

Same quantized model files used for all comparisons to ensure consistency.
Since **We are using an existing GGUF model**,
the model's accuracy is out of scope.



Quantization Mechanisms in LLM

Our latency is acceptable

Target Environment:

Wearable & IoT devices (Strictly power-constrained / Battery-operated).

Redefining "Real-Time":

Processing **10s of audio in 11s** is "perceptually real-time" for the user.

The interaction feels seamless as processing finishes almost immediately after the user stops speaking.

real-time streaming

We haven't implemented live streaming yet,
but it is feasible with software updates.

Rationale for TDP (Thermal Design Power)

- To compare **peak performance energy efficiency** potential across diverse hardware architectures (GPU vs. Edge AI vs. FPGA).
- Ensures a consistent baseline for "**peak load**" conditions rather than fluctuating wall-meter readings.

TABLE III: Physical specifications and performance comparisons of various devices

Device	CPU	Cores	Chip area (mm ²)	Process node	Operating frequency	Memory	Power ^c (W)
ARM Cortex-A72 (on Versal)	-	2	-	7 nm	1400 MHz	8 GB DDR4	0.6485
IMAX3 (Xilinx VPK180)	ARM Cortex-A72	64 ^a	-	7 nm	145 MHz	8 + 4 GB DDR4 ^b	180
IMAX3 (28nm)	-	64 ^a	14.6	28 nm	840 MHz	-	0.647 or 1.32
Jetson AGX Orin 32GB	Arm® Cortex-A78AE	1792	200	8 nm	930 MHz	32 GB DDR5	15 ^d
NVIDIA RTX 4090	Xeon W5-2455X	16384	608	5 nm	2520 MHz	24 GB DDR6	450

^a The number of cores for IMAX3 refers to the number of PEs per lane.

^b 8 GB DDR4 for OS buffer and 4 GB DDR4 for DMA buffer.

^c IMAX3 (28 nm) is an estimated value, references for other devices are from Cortex-A72 [27], Jetson AGX Orin 32GB [28], NVIDIA RTX 4090 [29].

^d This device has multiple power consumption modes (15–45 W), and we executed in the lowest power consumption mode.

ASIC Projection Methodology (28 nm)

Tools:

Synopsys Design Compiler with TSMC 28 nm standard cell libraries.

Assumptions:

Average switching activity set to **10%**.

Basis:

Based on established methodology from prior work (Akebe 2025).

ASIC -> **840MHz**

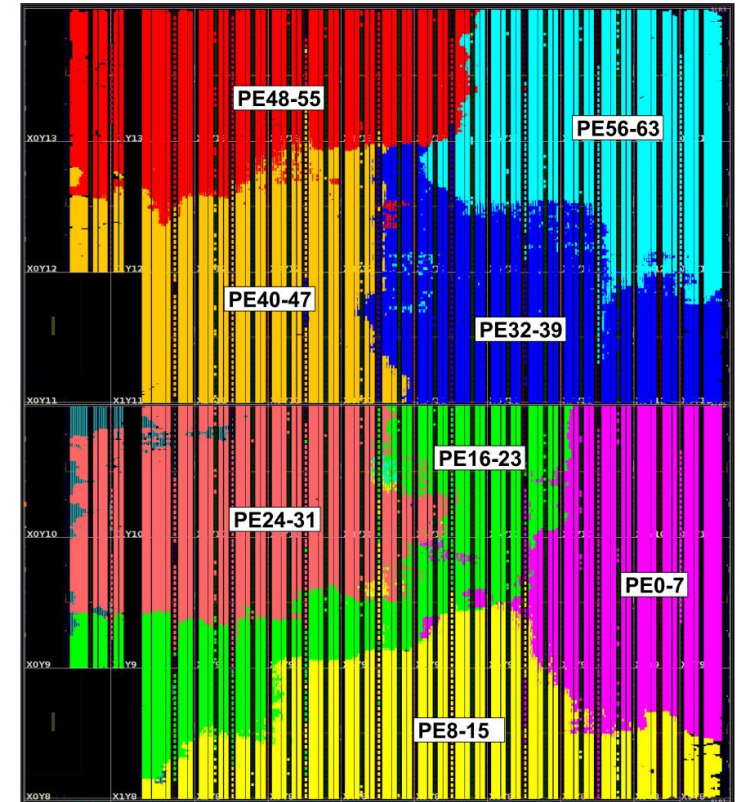


FIGURE 6. Implemented design of IMAX3 on VPK180.

ISA for IMAX

Supports general commands

Table.2.3: ALU operations

unit	usage	mode	description
CEX	CEXE ((char*)ex, c3, c2, c1, c0, imm)	4in 2bit 16bit	word-wise(w1/w0) conditional execution
EX 1,2,3	ex4.SFMA (Ull*)d s1, (s1, (Ull*)s2, (Ull*)s3, s4)	3in 8bit*32	stochastic $s1 + \sum (s2*s3) \gg s4 \rightarrow 8\text{bit}$
	ex4.FMA (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	3in 32bit*2*4	floating-point $s1+s2*s3$
	ex4.FMS (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	3in 32bit*2*4	floating-point $s1-s2*s3$
	ex4.FML (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	2in 32bit*2*4	floating-point $s1*s2$
	ex4.FAD (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	2in 32bit*2*4	floating-point $s1+s2$
	CFMA (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	3in idx 32bit	floating-point $s1+s2*s3$ if $hi(s2)==hi(s3)$
	FMA (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	3in 32bit*2	floating-point $s1+s2*s3$
	FMS (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	3in 32bit*2	floating-point $s1-s2*s3$
	FML (Ull*)d &s1, (s1, exp, s2, exp, -, -)	2in 32bit*2	floating-point $s1*s2$
	FAD (Ull*)d &s1, (s1, exp, s2, exp, -, -)	2in 32bit*2	floating-point $s1+s2$
	FML3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	3in 32bit*2	floating-point $s1*s2[idx:r3]$ for GGML
EX1	ex4.CVT86 (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	3in 32bit*2*4	llama-q6_K
	ex4.SML16 (Ull*)d &s1, ((Ull*)s1, (Ull*)s2, -)	2in 16bit*4	integer $\sum 2[16\text{bit}*8\text{bit}]\text{x}2$
	ex4.FMI (Ull*)d &s1, ((Ull*)s1, (Ull*)s2, -)	2in 32bit*2	float32*int32

-
-
-

Limitations

System-Level Limitations

Dependency on Host system performance (CPU/Interconnect).

Uncertainties inherent in ASIC projections.

Evaluation limited to specific model architectures.

Future Directions

Interconnect

Improving host-accelerator data transfer (e.g., PCIe optimization).

Hardware

Adding native support for low-bit quantization formats to further improve efficiency.

Limitations

⚠ System-Level Limitations

Host Dependency Performance bottlenecks due to CPU/Interconnect reliance.

Uncertain Projections Inherent unpredictability in ASIC performance scaling.

Limited Scope Evaluation restricted to specific model architectures.

🔮 Future Directions

Interconnect Optimization Enhancing host-accelerator data transfer rates (e.g., PCIe).

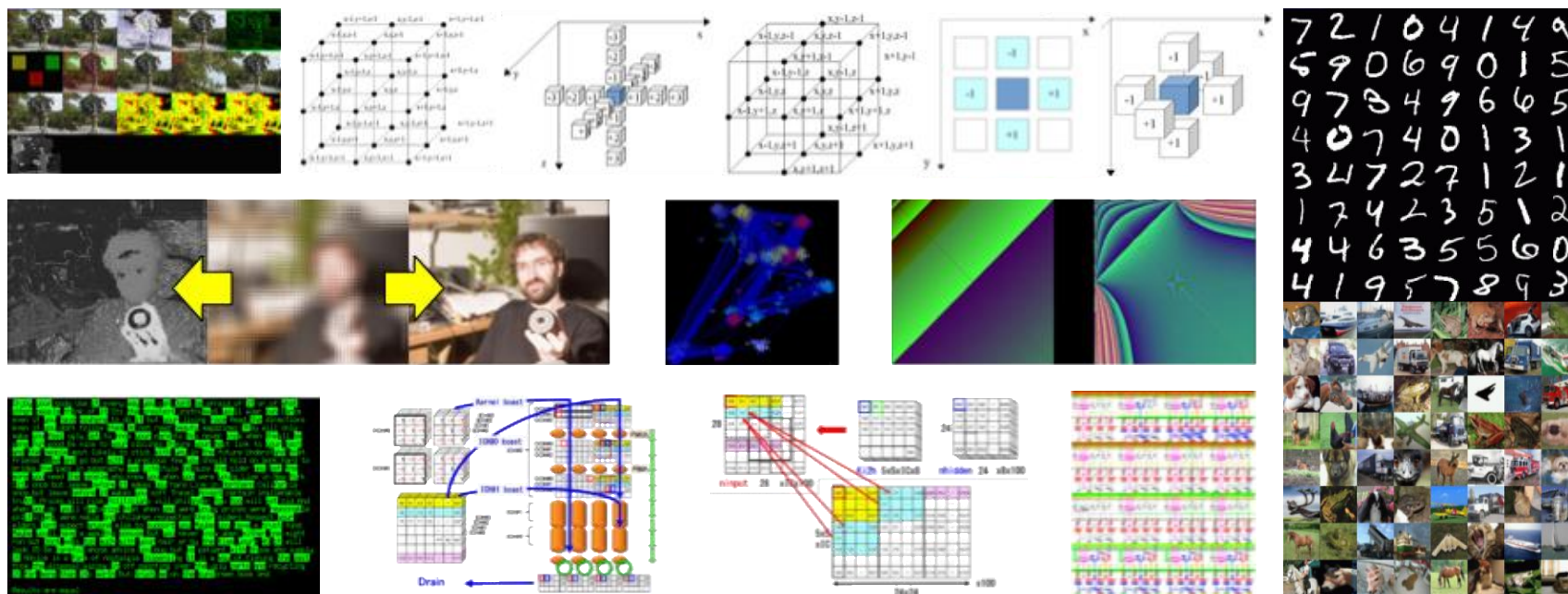
Hardware Support Native integration for low-bit quantization formats.

IMAX is All You Need

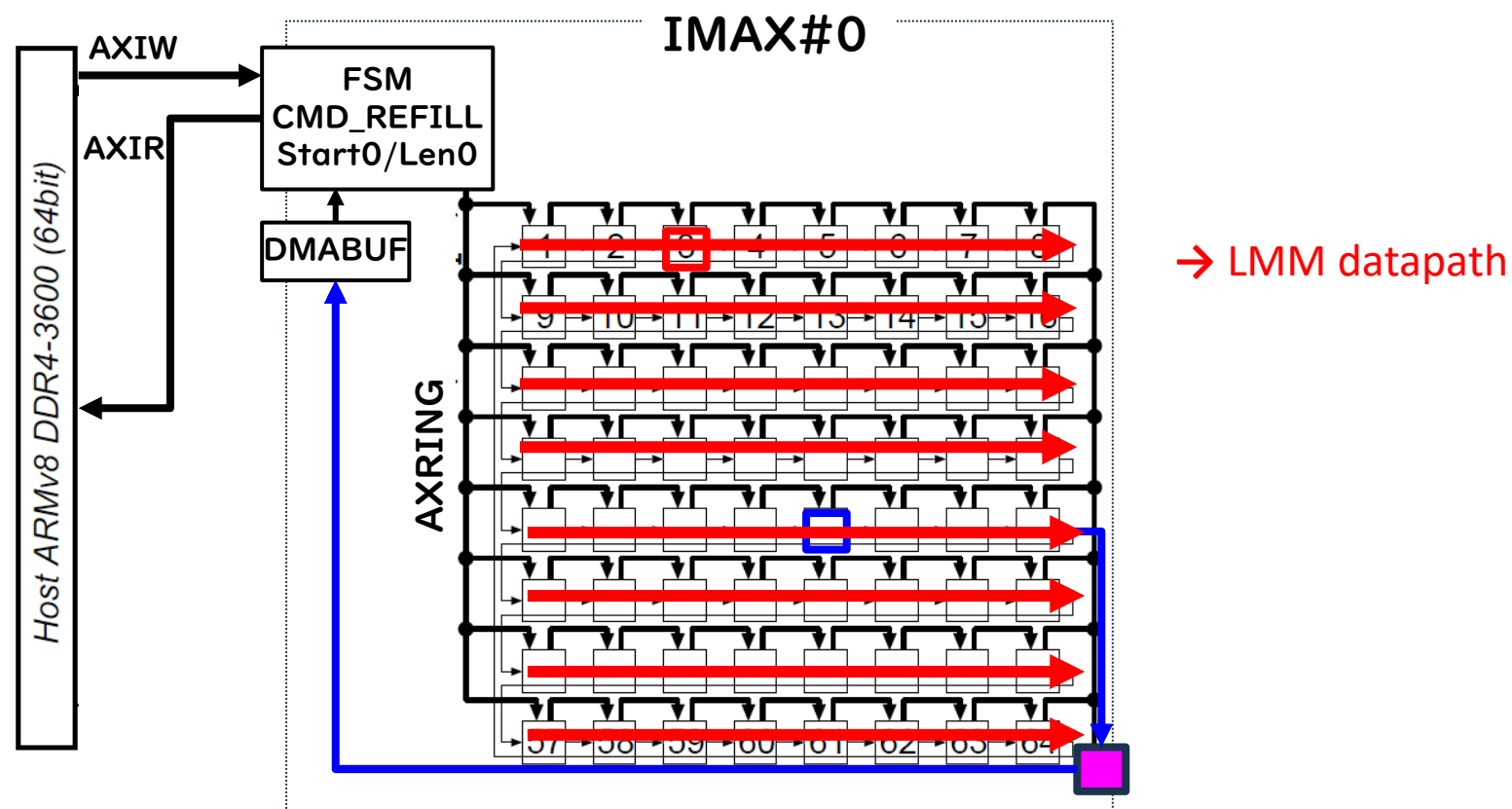
Proven Versatility: A Wide Range of Workloads

Effective on traditional kernels (SpGEMM, FFT) and image processing.

Robustly applied to AI, covering CNNs , GCNs , Attention , and LLMs.



Since the memory interface is connected every 8 units,
it's linear in the sense that **memory is not sent from one end to the other.**



Research Contributions

1. Efficacy of General-Purpose CGRA Architecture

- We have comprehensively demonstrated for the first time that general-purpose CGRA, not dedicated to AI, can achieve high energy efficiency in AI inference.
- This validates a sustainable design approach, avoiding the risk of rapid obsolescence of algorithm-specific ASICs.

2. System-Level Bottleneck Identification

Moves beyond isolated kernel benchmarks to an end-to-end evaluation using ggml.cpp.

Data transfer is the critical bottleneck, often overlooked in kernel-centric studies.

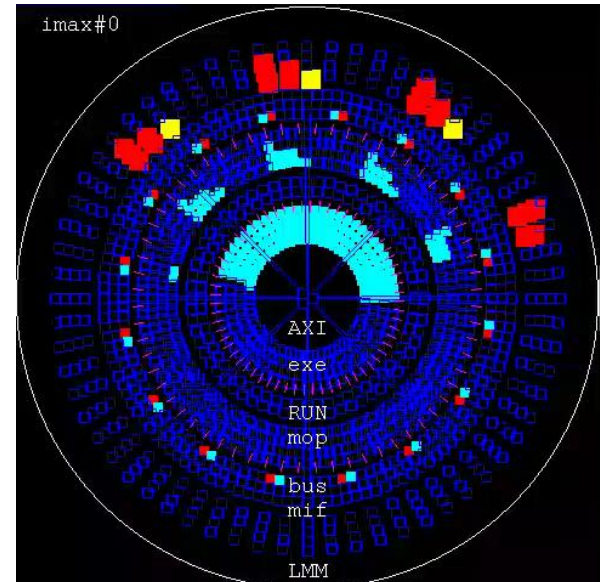
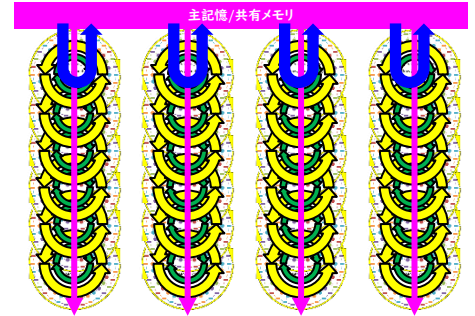
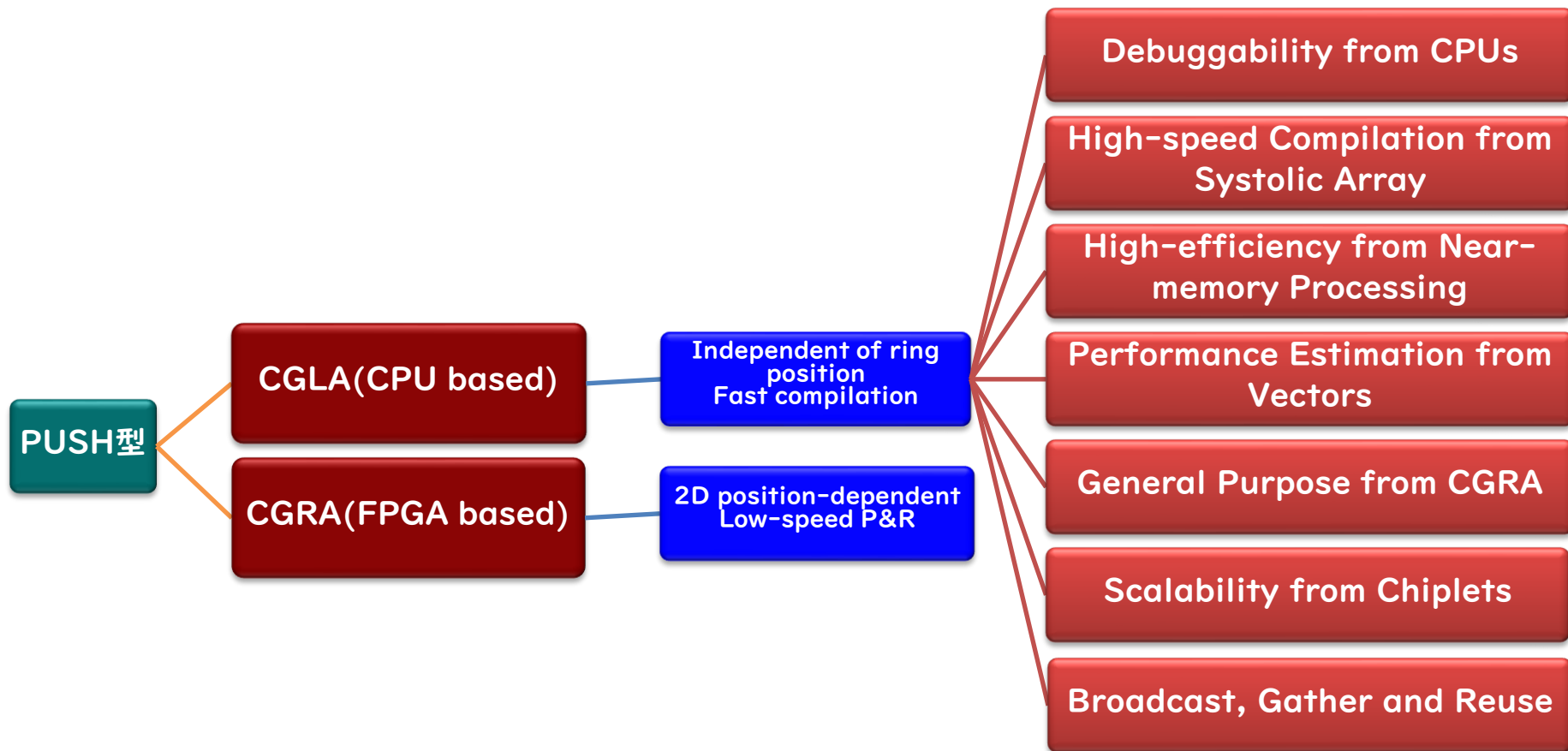
3. Viability of CGRAs for LLMs

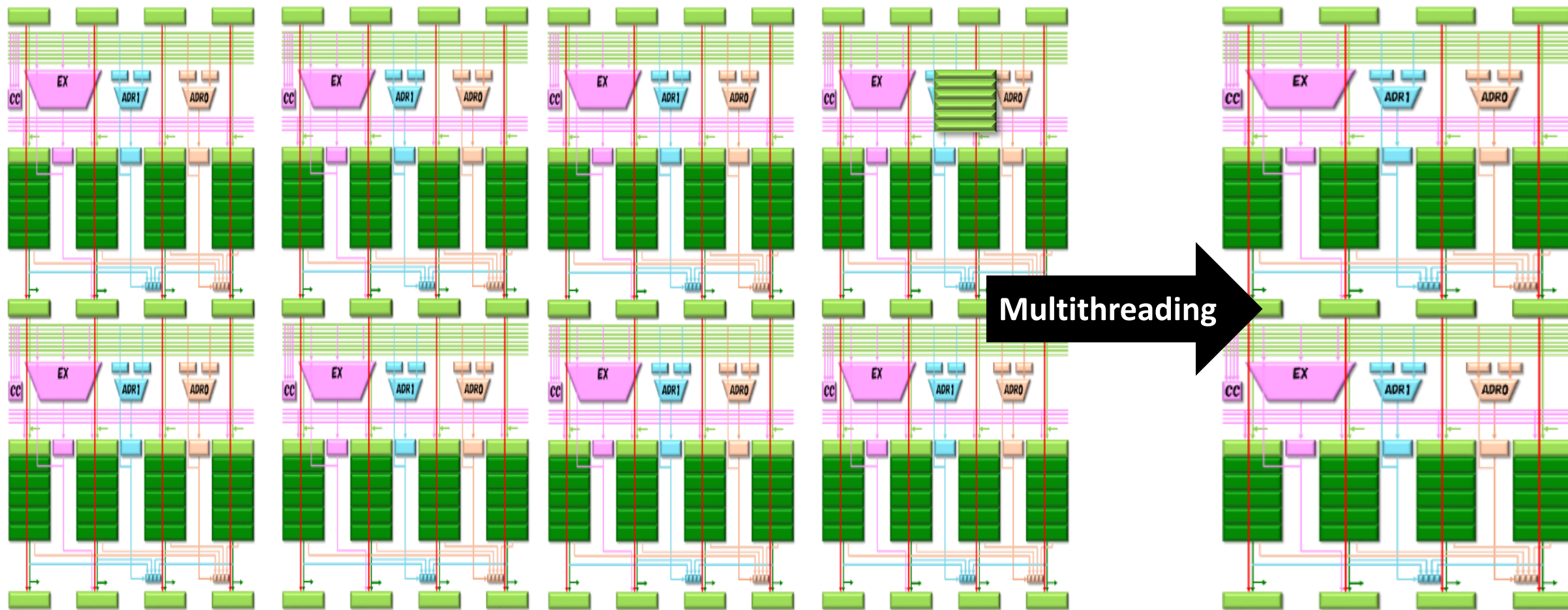
Demonstrates CGRAs as the optimal trade-off between FPGAs and ASICs .

CGLA is NOT magic

Just discovering the right combination of existing technologies.

Program Model	Load(pull)-ALU-Store	Load(pull)-ALU-Store	VLD(pull)-ALU-VST	Anywhere LD(push)-ALU-ST
	Manycore	GPGPU (coalescing)	Long Vector Extension (full-coalescing)	CGLA (Coarse Grained Linear Array)



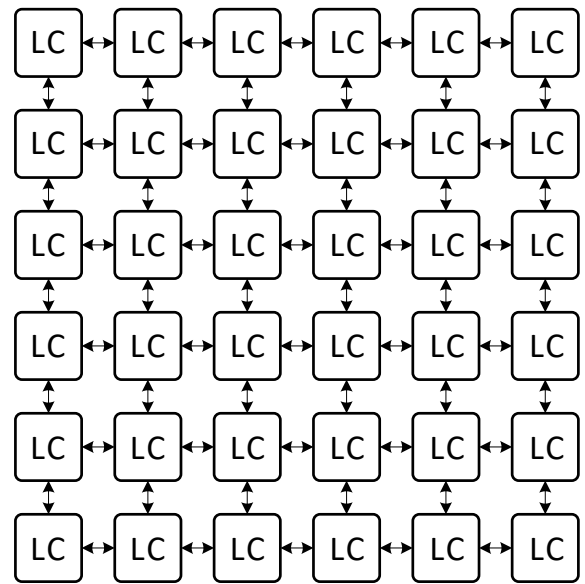


Logical unit

physical unit

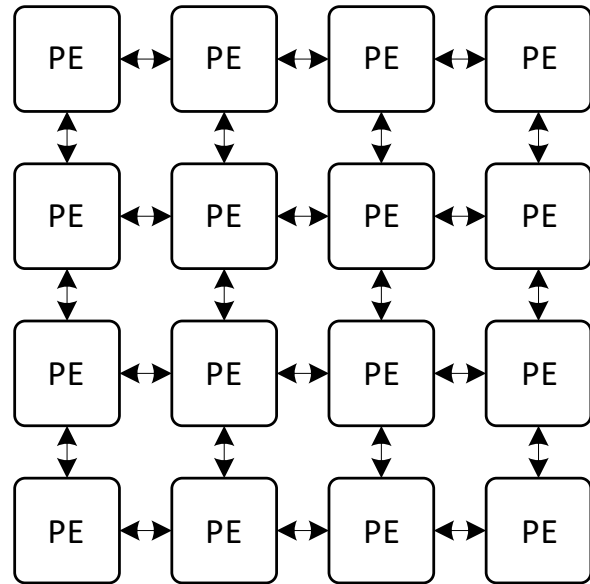
FPGA

Field Programmable Gate Array



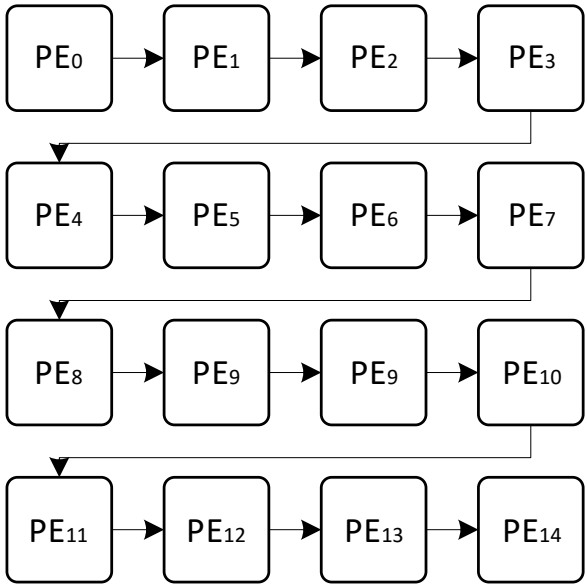
CGRA

Coarse-Grained Reconfigurable Array



CGLA

Coarse-Grained Linear Array



	FPGA		CGRA		CGLA
Unit	LUT	>	PE	=	PE
Routing flexibility	High	>	Middle	>	Low-Middle
Compilation time	6h-1d	>	10m-60m	>	1-10s

[5] Akabe, Tomoya, Vu Trung Duong Le, and Yasuhiko Nakashima. "IMAX: A Power-efficient Multilevel Pipelined CGLA and Applications."