PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System

Yintao He Haiyu Mao Christina Giannoula
Mohammad Sadrosadati Juan Gómez-Luna Huawei Li
Xiaowei Li Ying Wang Onur Mutlu

ASPLOS 2025

SAFARI

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

KING'S College LONDON

UNIVERSITY OF TORONTO

ETH Zürich

NVIDIA

Executive Summary

Observation: Large Language Model (LLM) decoding kernels have **different and dynamically changing** computation and memory bandwidth demands at runtime

Problem: Existing heterogeneous LLM systems have two shortcomings:
- **Static scheduling** that fails to dynamically cater to changing kernel demands
- **Support only one type of Processing-In-Memory (PIM) device** with a certain computation throughput and memory bandwidth capability

Goal: Design a **heterogeneous system** that caters to different and dynamically changing computation and memory demands in LLM decoding

Key Idea: Enable **online dynamic task scheduling** on a heterogeneous architecture via online identification of LLM decoding kernel properties

Key techniques: A new computing system called **PAPI:**
- **Dynamic LLM kernel scheduling** to the most suitable hardware units at runtime
- **Hybrid PIM units** to meet the diverse LLM kernel demands

Key Results: PAPI outperforms a state-of-the-art PIM-enabled LLM computing system and a pure PIM system by **1.8X** and **11.1X**, respectively

Outline

1 **Background**

2 Observations & Motivation

3 PAPI's Overview

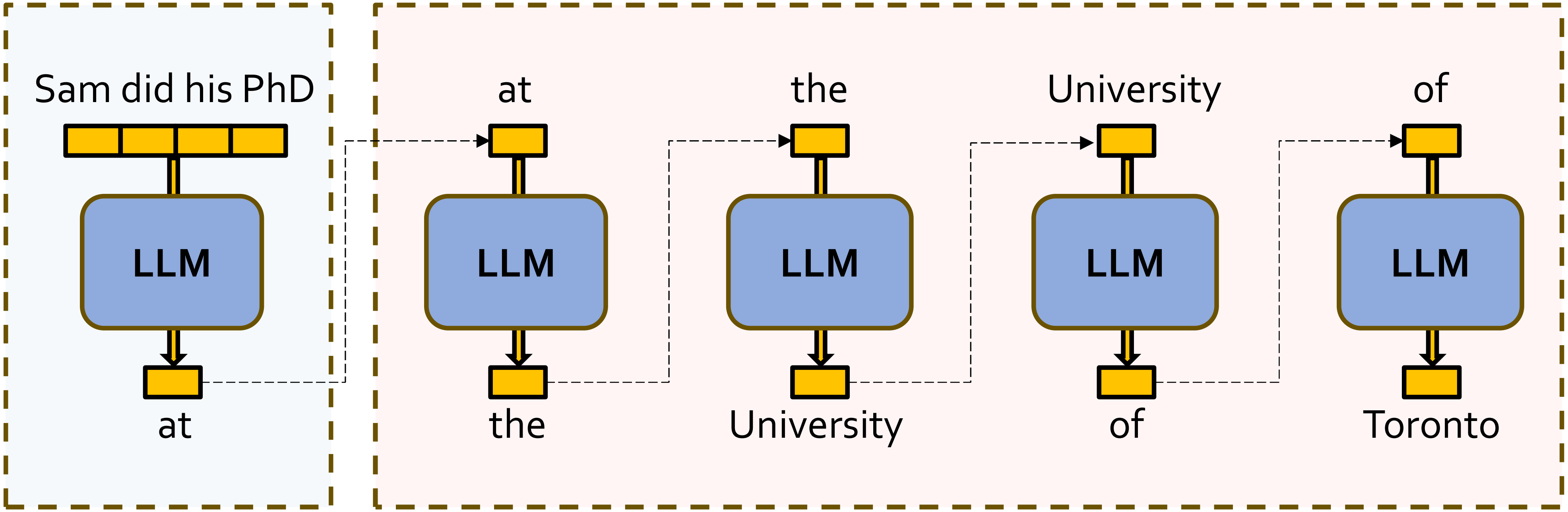
4 PAPI's Implementation

5 Evaluation

6 Conclusion

LLM Inference

An example:



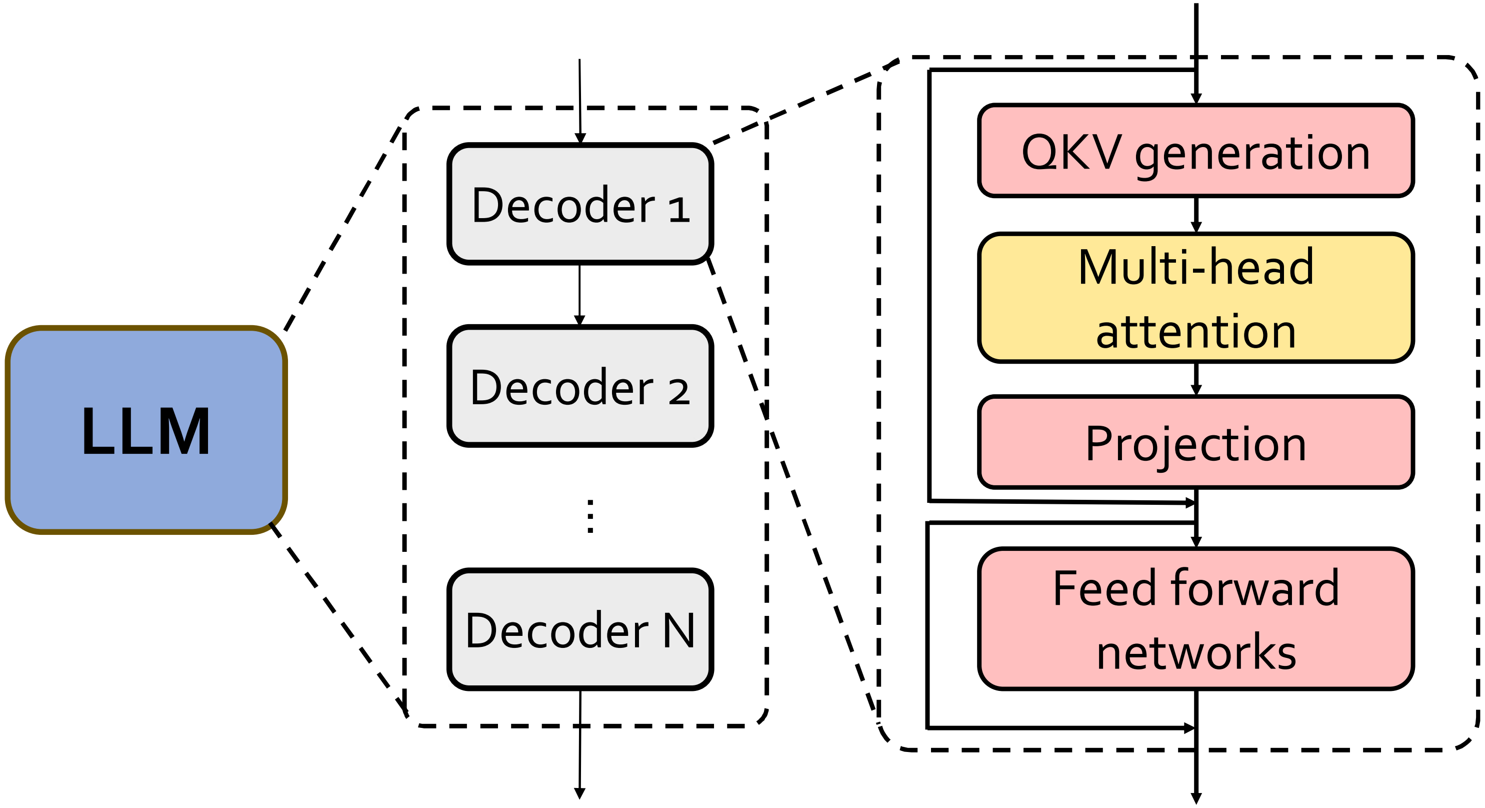
Prefilling

(Encodes contextual information from the input in parallel)

Decoding

(Generates output tokens **in serial or parallel**)

LLM Structure

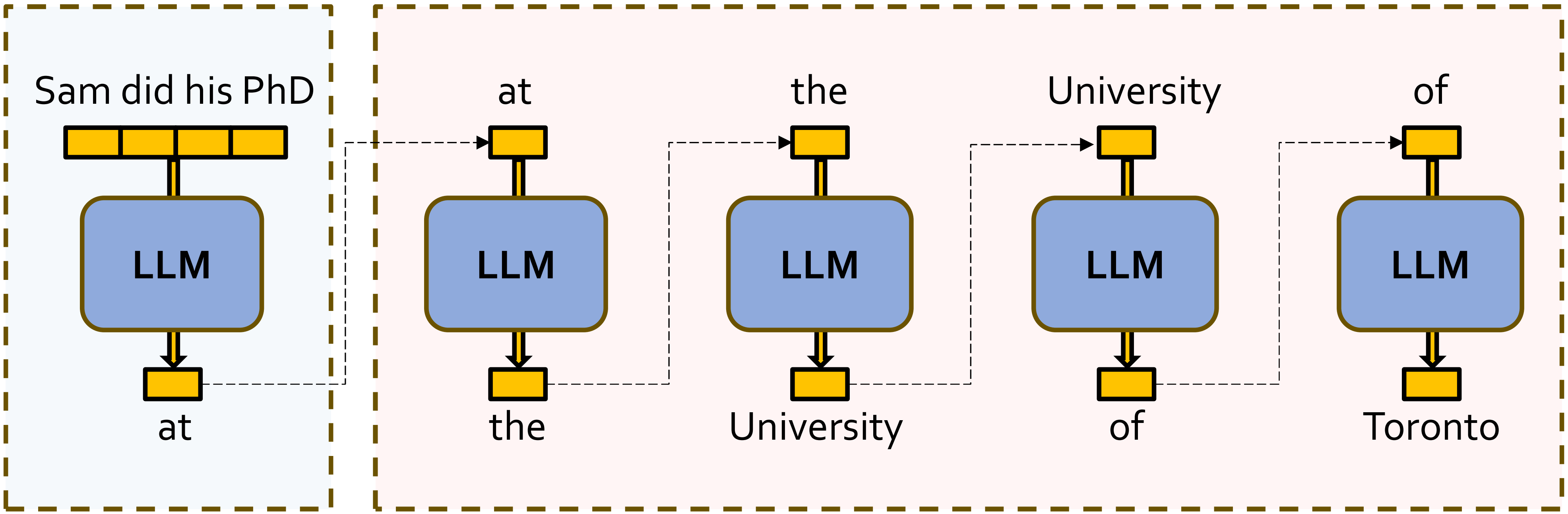


Attention kernels
- Encoded from input tokens
- Different data across requests

Fully-connected (FC) kernels
- Pretrained by LLM training
- Used for token generation

LLM Inference

An example:



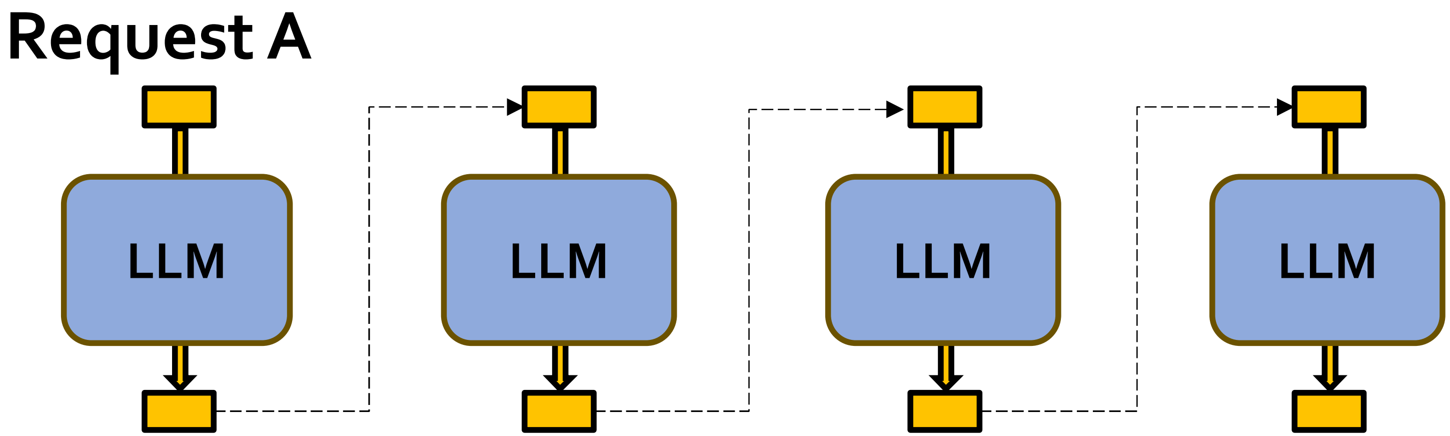
Prefilling

(Encodes contextual information from the input in parallel)

Decoding

(Generates output tokens **in serial or parallel**)

Serial Decoding

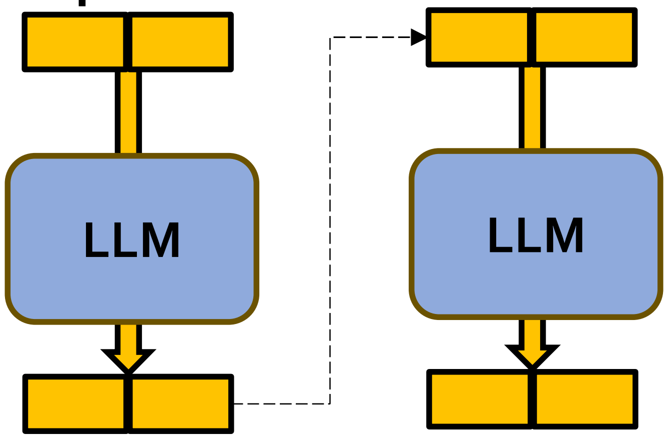


- Low hardware utilization
- Low throughput

Parallel Decoding

Decode tokens of one request in parallel

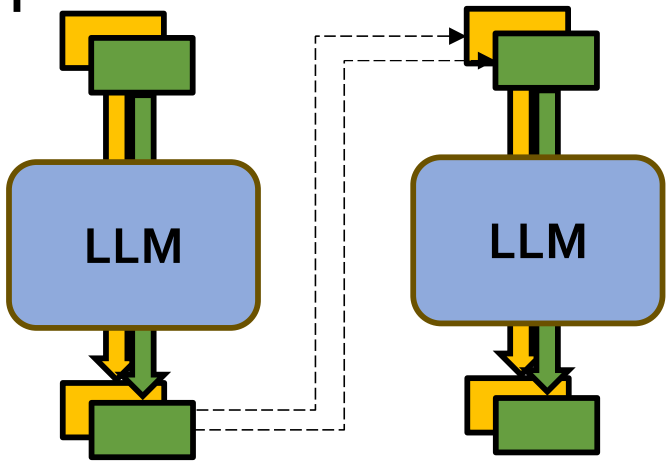
Request A



Token-Level Parallelism (TLP)

Decode different requests in parallel

Requests A & B



Request-Level Parallelism (RLP)

- Higher hardware utilization
- Higher throughput

Do TLP and RLP benefit all kernels in LLM decoding?

Outline

1 Background

2 Observations & Motivation

3 PAPI's Overview

4 PAPI's Implementation

5 Evaluation

6 Conclusion

Key Observations

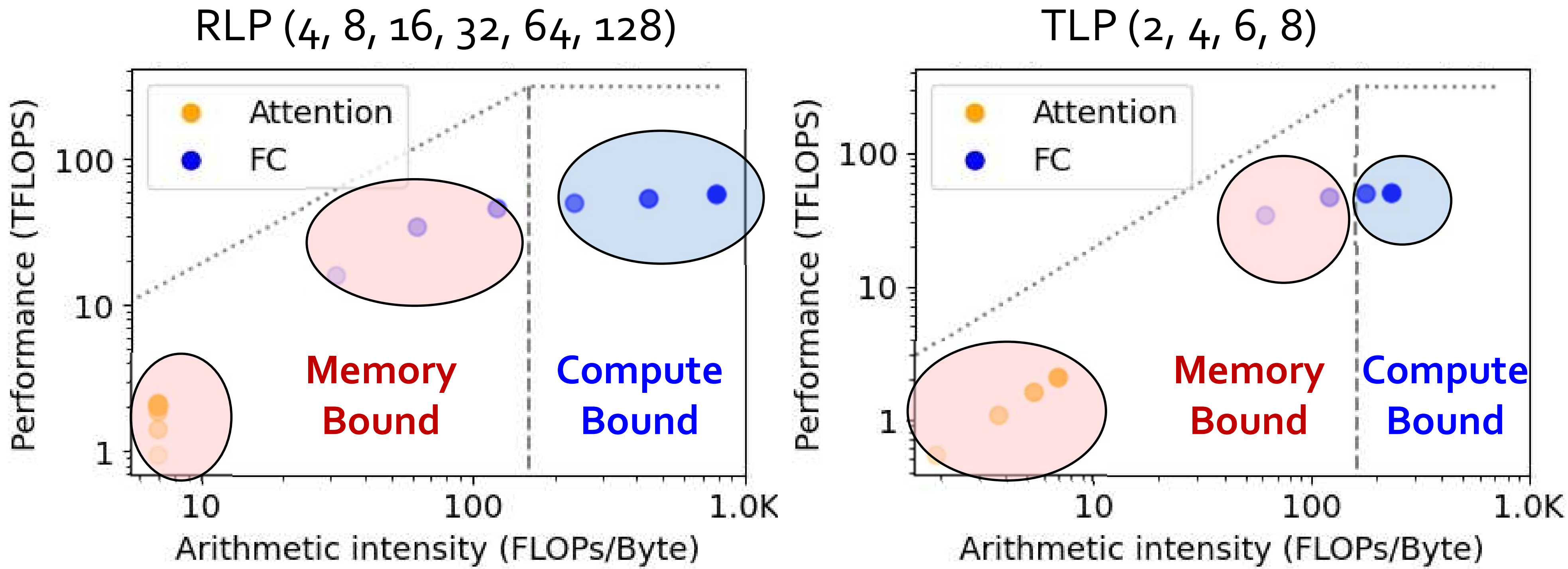
1 LLM kernels have **different computation and memory bandwidth demands** across **different RLP & TLP levels**

2 **Memory-bound kernels** exhibit **different computation demands** depending on kernel type

3 LLM kernels have **dynamically changing RLP and TLP levels**

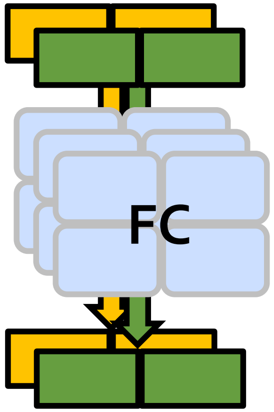
1. Different Computation and Memory Bandwidth Demands due to RLP/TLP

Roofline model of LLM kernels with **six RLP and four TLP configurations** on an NVIDIA A100 **GPU system**:



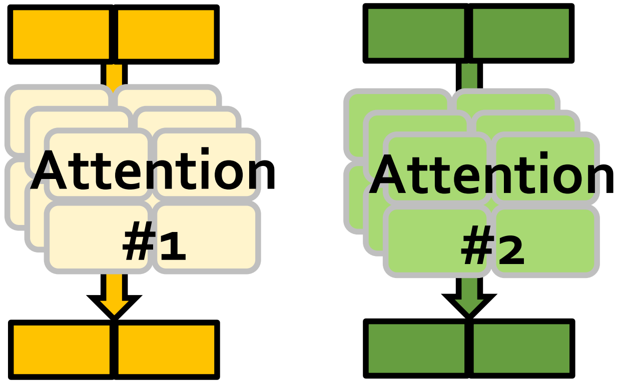
LLM kernels have different computation and memory bandwidth demands across different RLP & TLP levels

Why Different Computation & Memory Demands in Parallel Decoding?



- **FC kernels** benefit from **RLP & TLP**

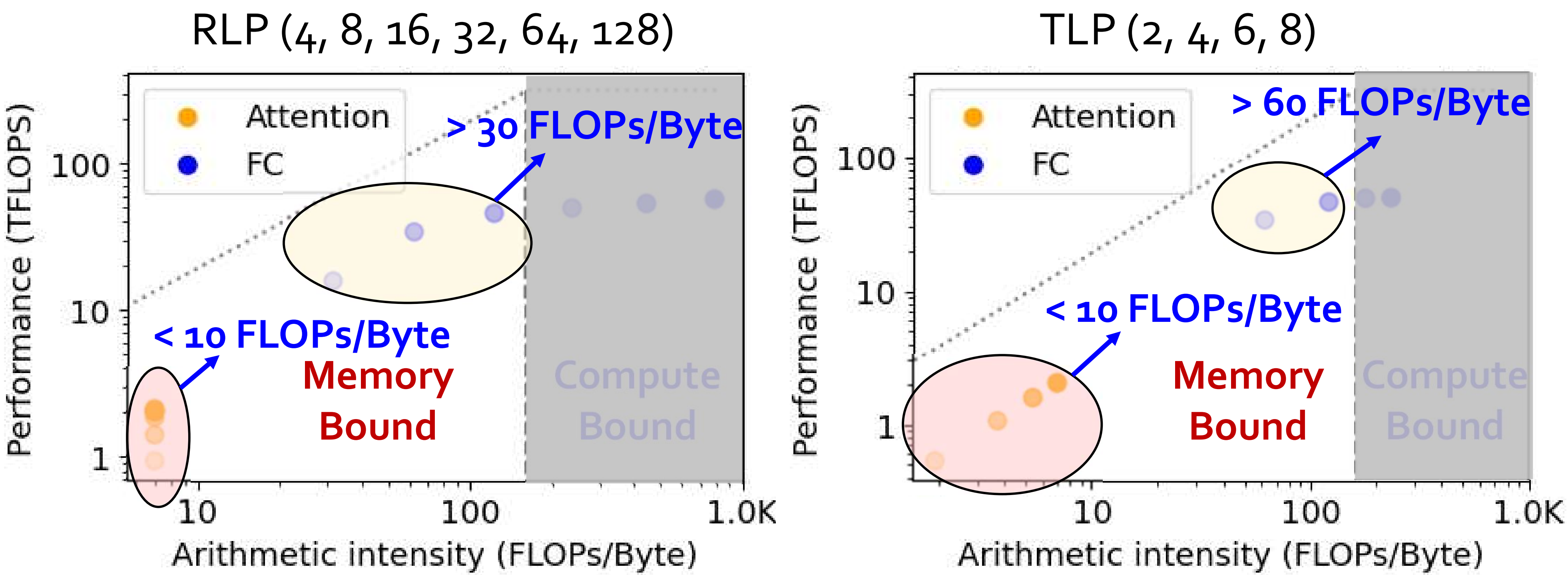
Compute-Bound



- **Attention kernels** benefit from **TLP**
- **TLP** is usually **much smaller** than RLP

Memory-Bound

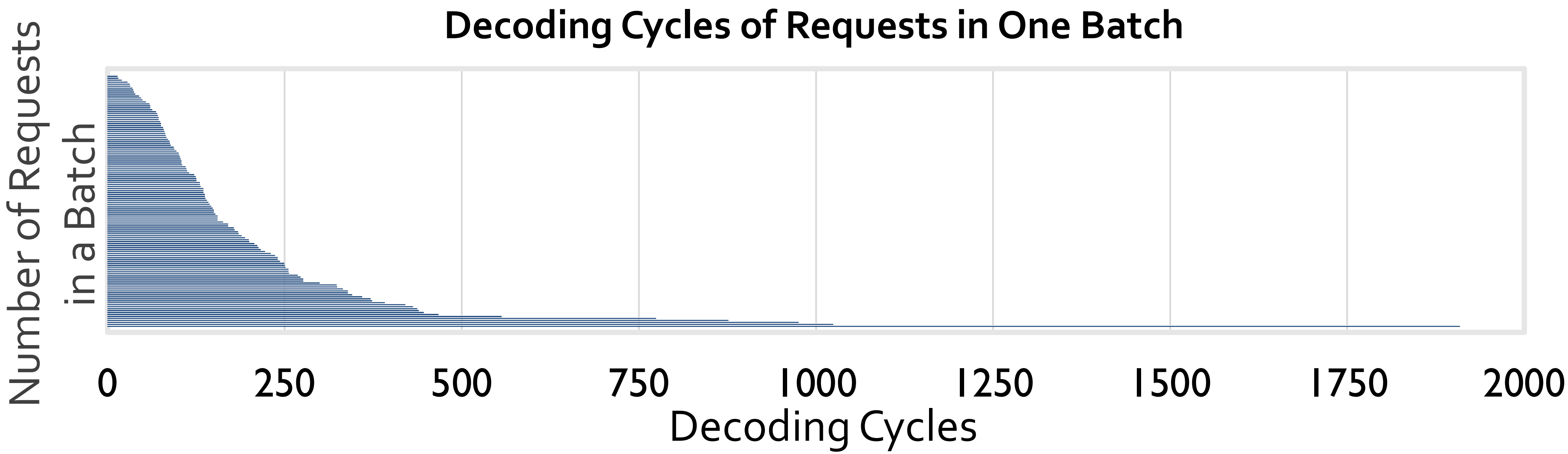
2. Different Computation and Memory Bandwidth Demands due to Kernel Type



Memory-bound kernels exhibit **different computation demands** depending on kernel type

3. Dynamically Changing RLP and TLP Levels

- **Parallelism levels** (RLP & TLP) **vary dynamically** in real-world scenarios
  - E.g., request-level parallelism (RLP) **decreases at runtime** when using static batching



In the Paper: Analysis of Dynamic Parallelism Levels

- **Initial RLP:**
  - Service level objective
  - Memory capacity limits
  - Dynamic batching

- **Runtime RLP:**
  - Static batching
  - Mixed continuous batching

- **TLP:**
  - Speculative decoding

LLM kernels have **dynamically changing RLP and TLP levels**

In the Paper: Analysis of Dynamic Parallelism Levels

PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System

Yintao He[1,2] Haiyu Mao[3,4] Christina Giannoula[5,6,4] Mohammad Sadrosadati[4]
Juan Gómez-Luna[7] Huawei Li[1,2] Xiaowei Li[1,2] Ying Wang[1] Onur Mutlu[4]
[1]SKLP, Institute of Computing Technology, CAS [2]University of Chinese Academy of Sciences [3] King's College London
[4]ETH Zürich [5]University of Toronto [6]Vector Institute [7] NVIDIA

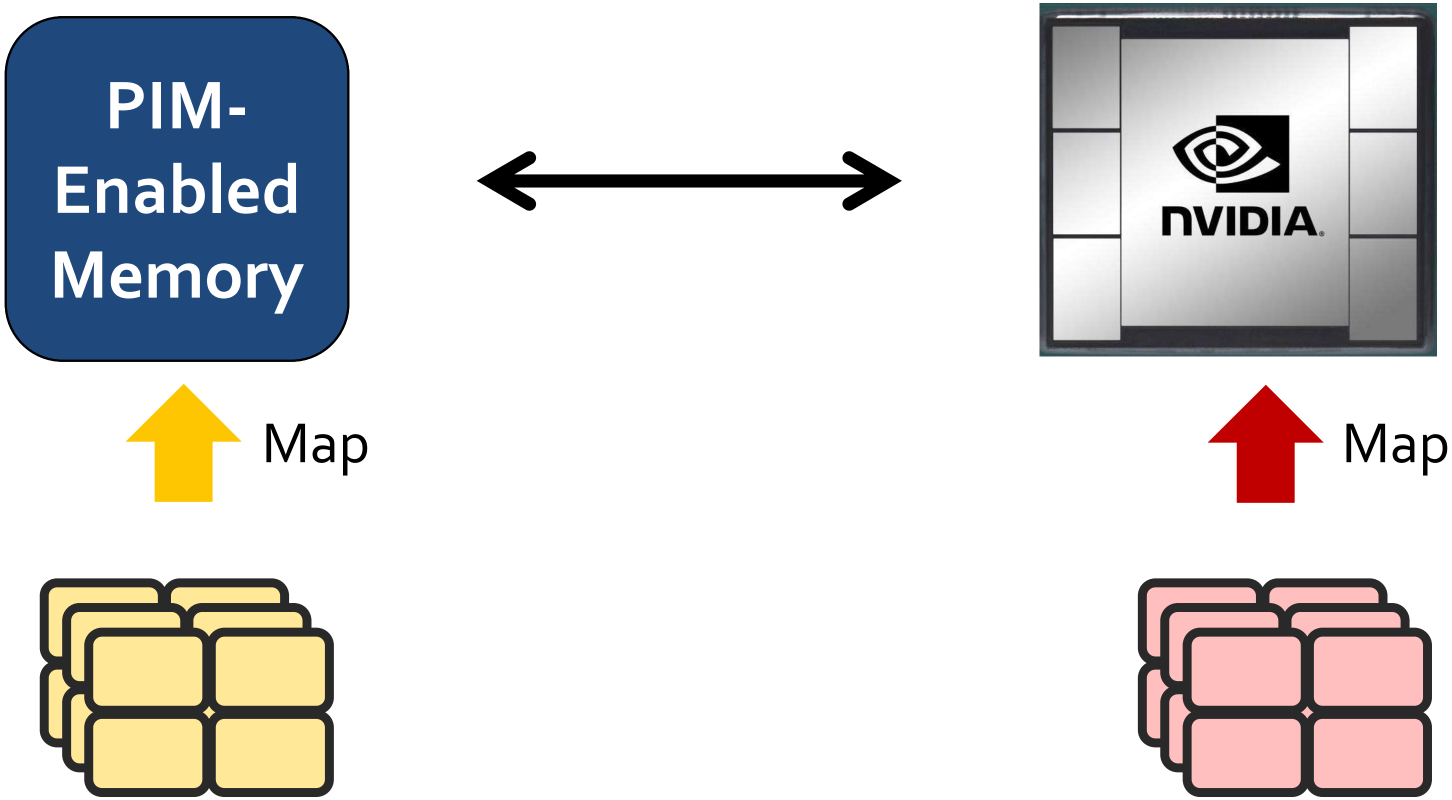
https://arxiv.org/pdf/2502.15470

State-of-the-Art in LLM Inference

A PIM-enabled LLM computing system:

Memory-centric computing device
(e.g., HBM-PIM)

Computation-centric accelerator
(e.g., GPU)



Attention kernels

FC kernels

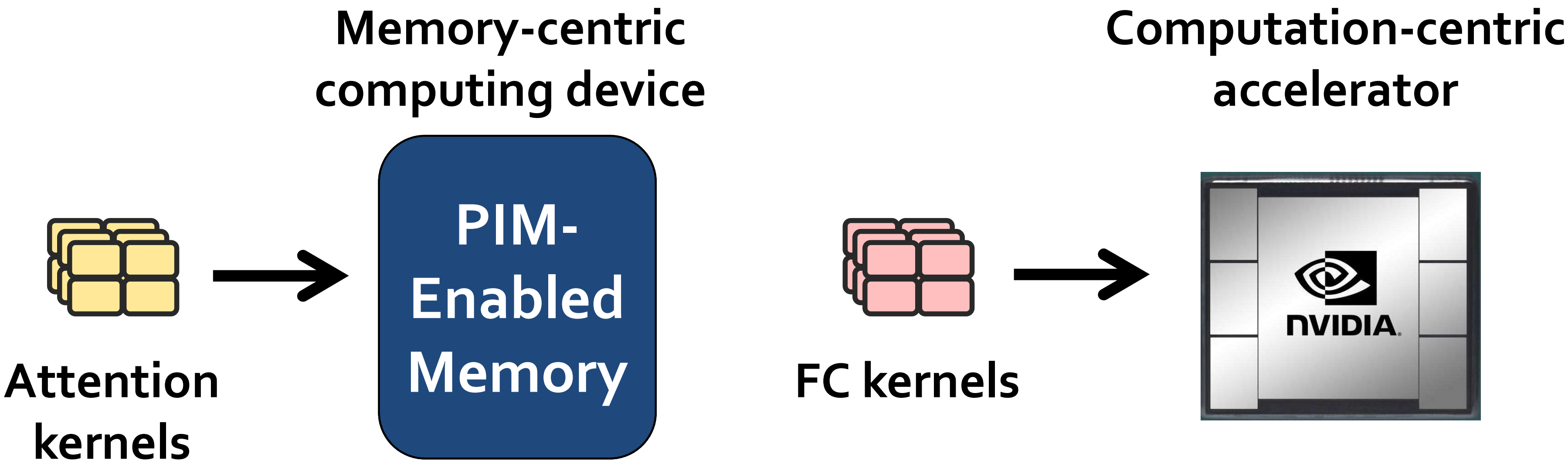
Major Shortcomings

1 **Static scheduling** leads to **sub-optimal** performance across **different parallelism levels**

2 Prior approaches support **only one type of PIM device** with a **certain computation and memory bandwidth capability**

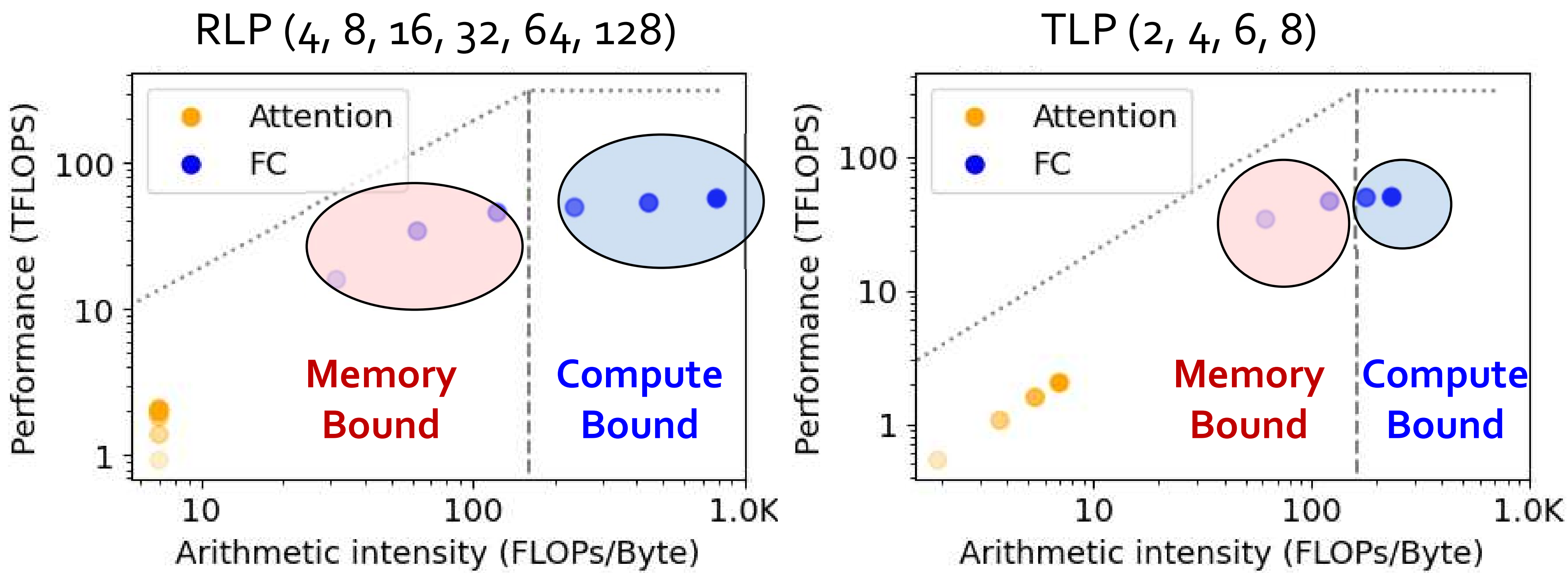
Shortcoming 1: Static Scheduling (I)

State-of-the-art typically uses **static scheduling**:



Shortcoming 1: Static Scheduling (II)

- Static scheduling works **well** for **memory-bound attention kernels**
- Static scheduling **fails** for **FC kernels** that switch between being **compute-bound or memory-bound**



Static scheduling leads to sub-optimal performance across different parallelism levels

Shortcoming 2: One-Size-Fits-All Approach

Prior works leverage
only **one type of PIM device** with
a **fixed computation and memory bandwidth**

Memory-bound FC kernels and attention
kernels have **varying computation**
and memory bandwidth demands

Prior approaches support only one type of PIM device
with a certain computation and memory bandwidth capability

Our Goal

Design a heterogeneous system that caters to **varying parallelism levels** in real-world LLM inference with **different and dynamically changing computation and memory demands**

Outline

PAPI's Key Idea

Enable **online dynamic task scheduling** in a **heterogeneous PIM-enabled architecture** via online identification of kernel properties in LLM decoding

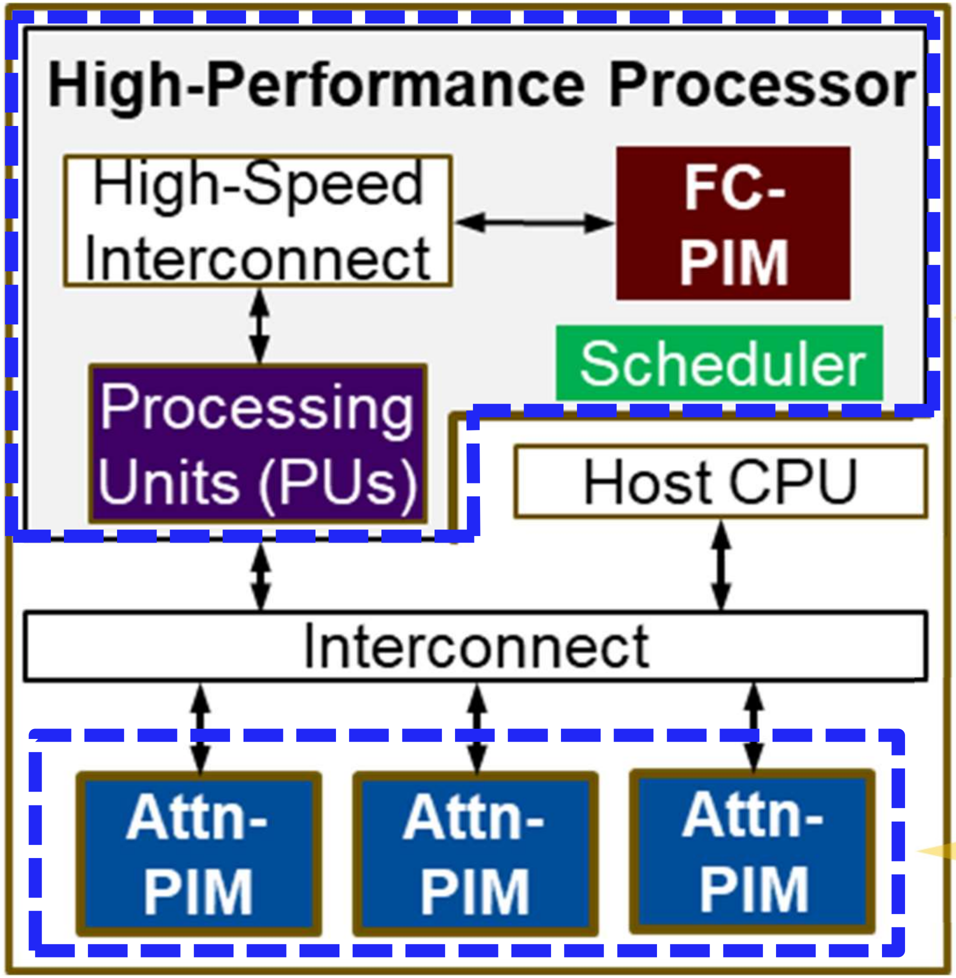
PAPI's Key Components

A new PIM-enabled computing system design

Hybrid PIM units
to cater to different parallelism levels of FC and attention kernels

Dynamic LLM kernel scheduling
to cater to dynamically changing parallelism levels

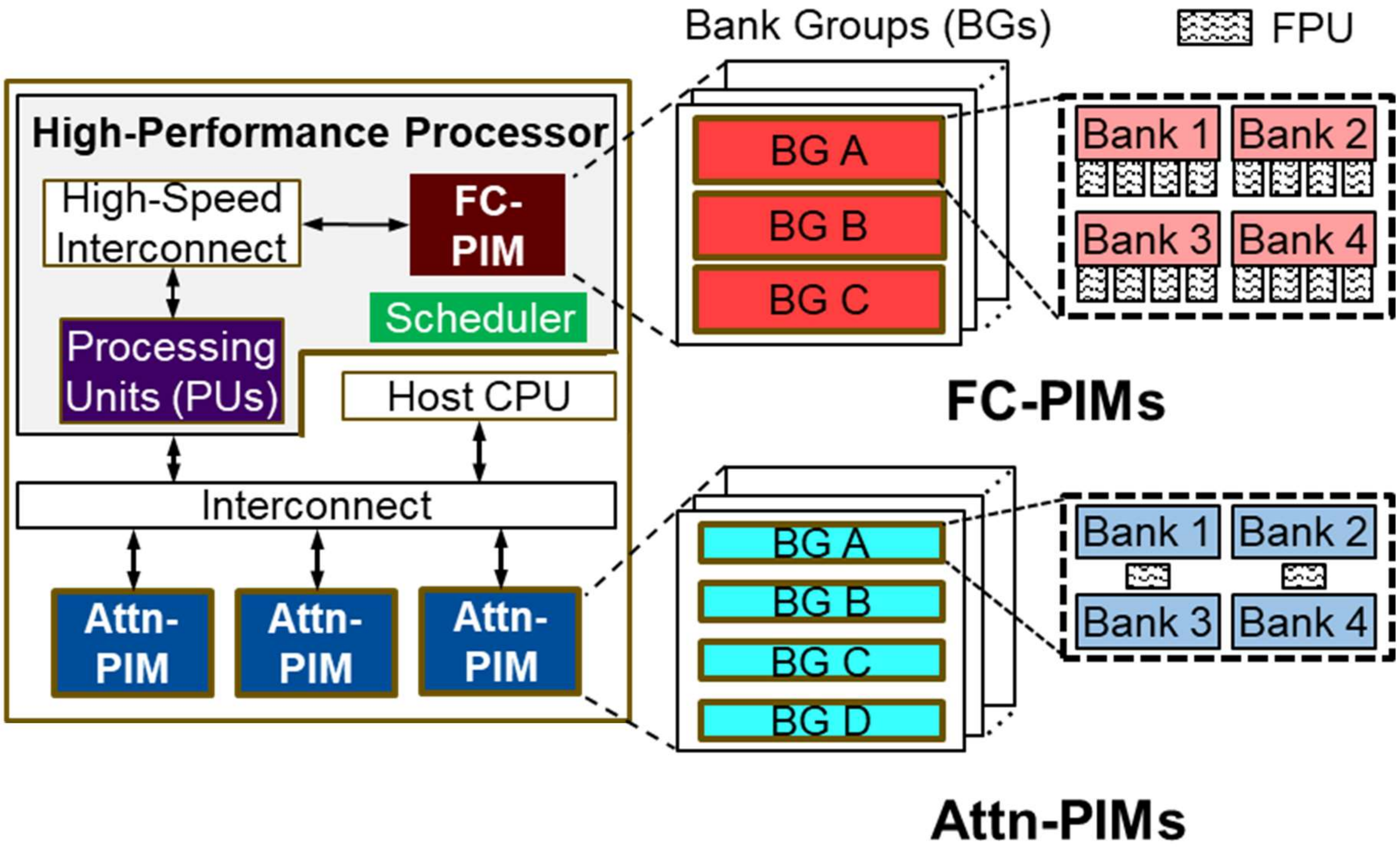
PAPI's Architecture



Handles memory-bound or compute-bound **FC kernels**
- Execution of FC kernels
- Dynamic scheduling

Handles memory-bound **attention kernels**

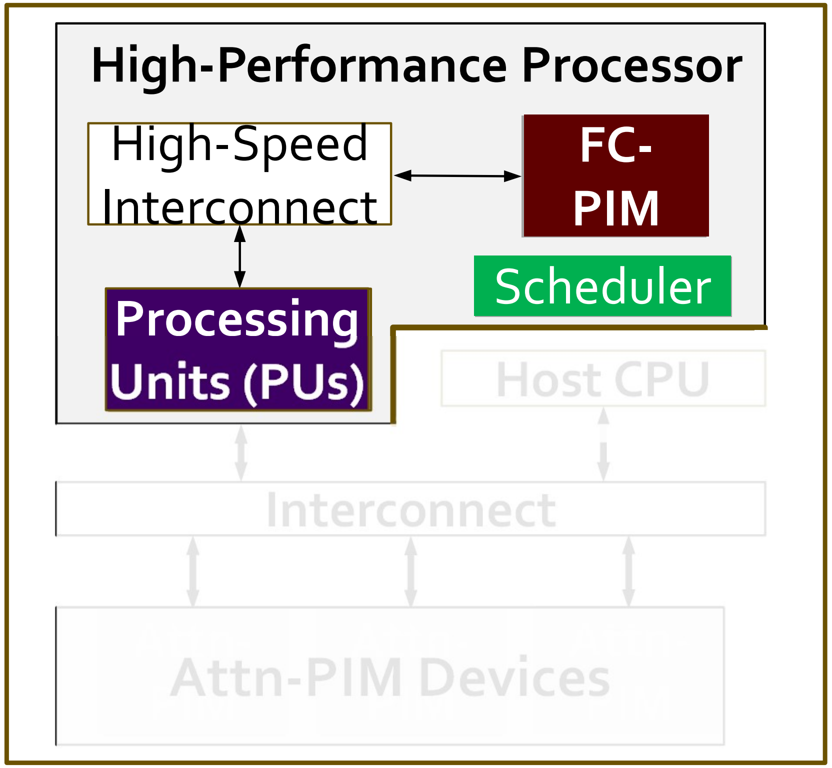
PAPI's Architecture



Hybrid PIM units handle memory-bound FC & attention kernels with **different computational and memory demands**

Outline

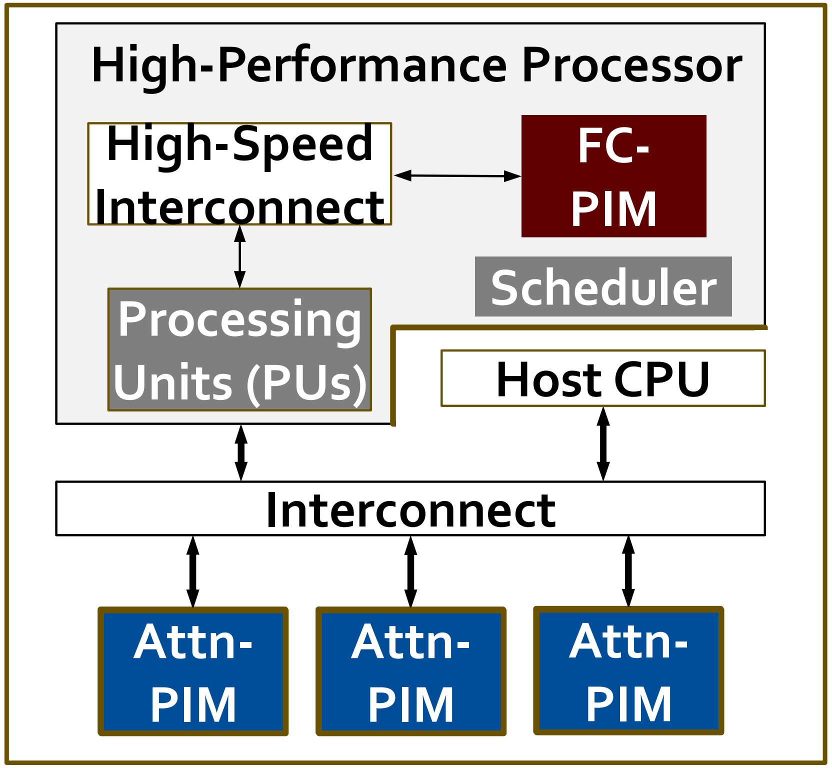
High-Performance Processor



When FC kernels are compute-bound:
Assign FC kernels to PUs

When FC kernels are memory-bound:
Assign FC kernels to FC-PIM

Hybrid PIM Units (I)



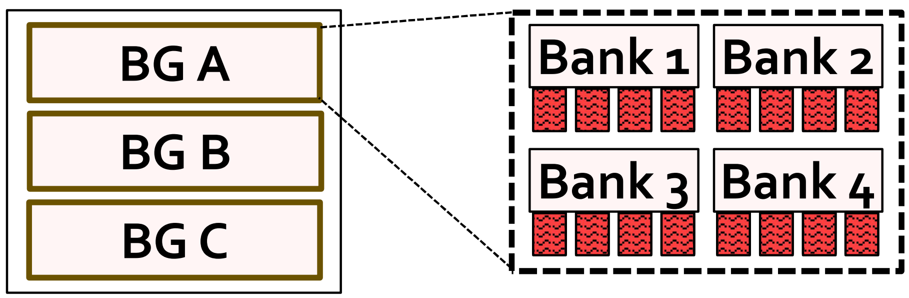
FC-PIM device placed in the High-Performance Processor

Attn-PIM devices store KV cache; **separated from** the High-Performance Processor

Hybrid PIM Units (II)

Floating-Point Processing Units (FPU)

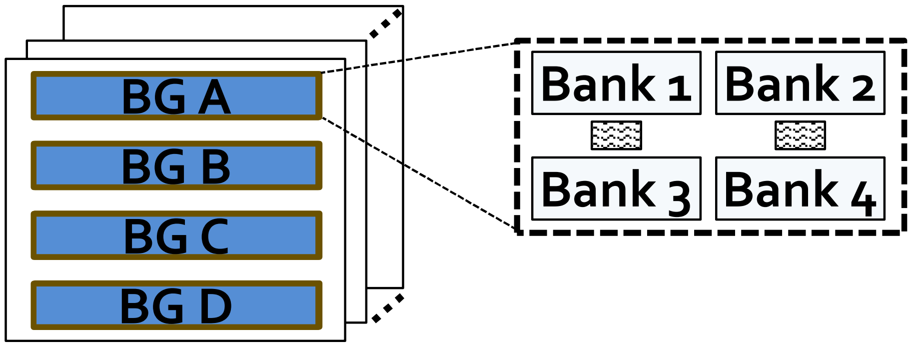
Bank Groups (BGs)



FC-PIM

More FPUs per Bank

Higher Computation Capability
to cater to FC kernels



Attn-PIMs

More Bank Groups per Stack
More Attn-PIM Devices

Higher Memory Capacity
to cater to attention kernels

PAPI Runtime Scheduler

Offline: identify memory-boundedness threshold

① Monitor Parallelism Levels

- RLP & TLP

② Arithmetic Intensity Predictor

- Estimate arithmetic intensity of FC kernels
- Compare with memory-boundedness threshold

③ Schedule the FC Kernels

- Map FC kernels to either FC-PIM or PUs

Outline

Evaluation Methodology

Performance and Energy Analysis:

- Simulation using AttAcc [ASPLOS'24] and Ramulator 2 [IEEE CAL'23]

Baselines:

- **AttAcc** [ASPLOS'24]
- **GPU+HBM-PIM** (NVIDIA A100 GPU + Samsung's HBM-PIM)
- **PIM-only** (PIM devices in AttAcc)

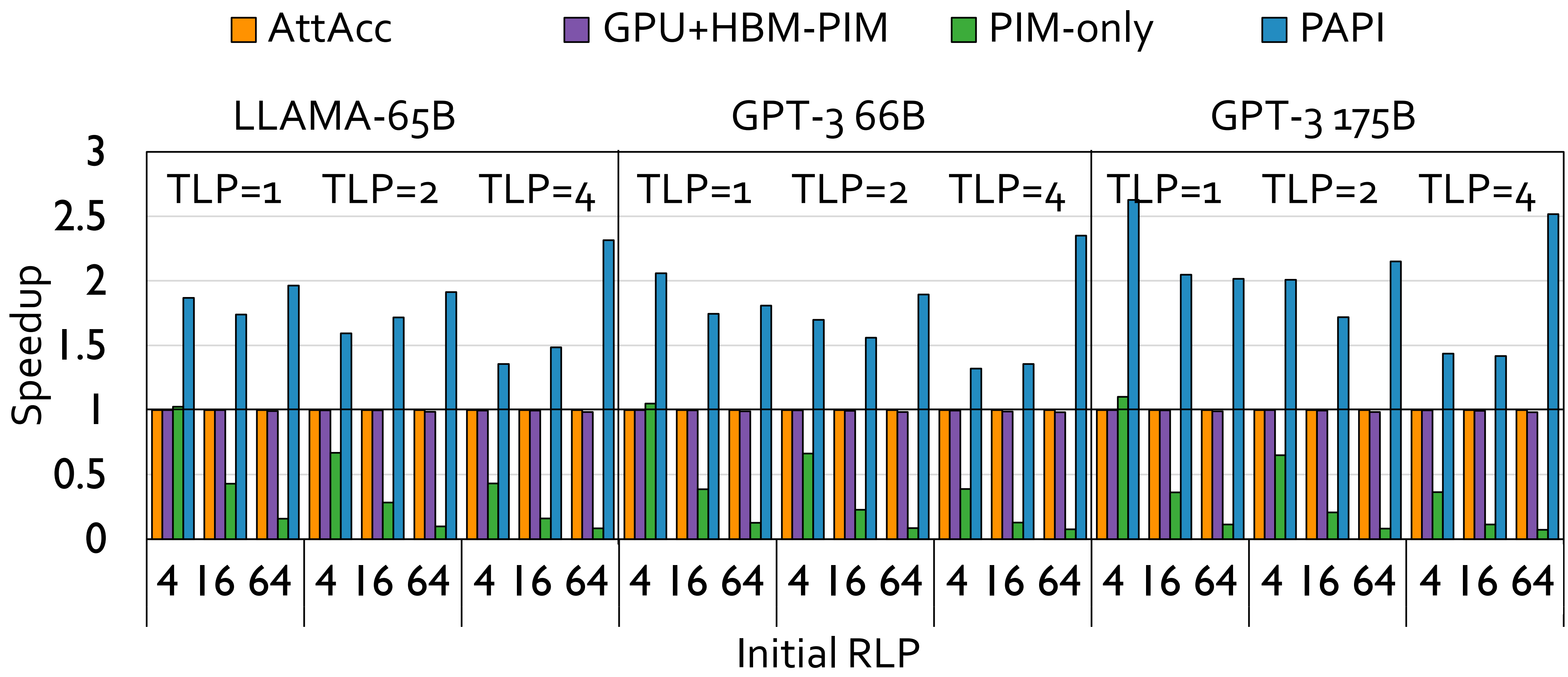
Workloads: Three transformer-based LLMs

- LLaMA-65B, GPT-3 66B, GPT-3 175B

Datasets: Dolly

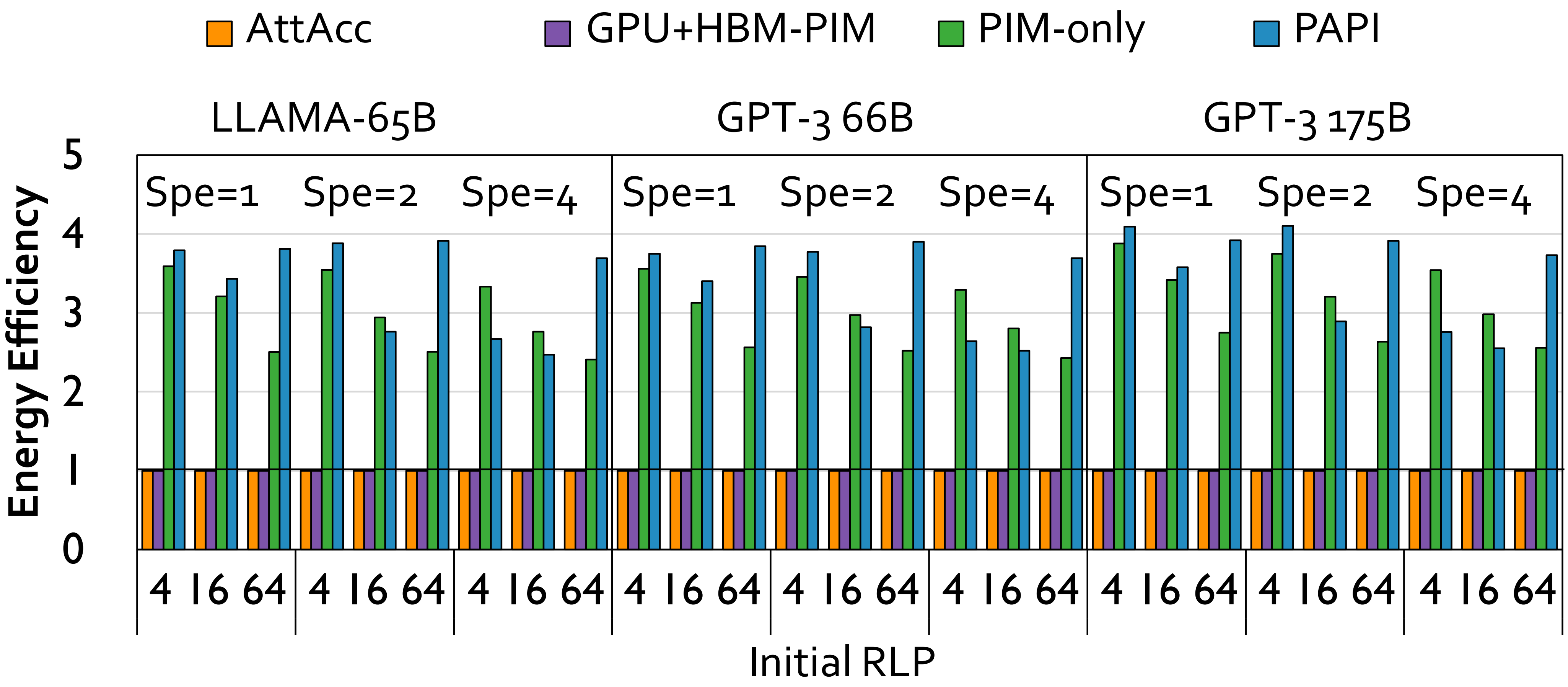
- Creative-writing tasks
- General-QA tasks

Performance Analysis



PAPI improves performance by 1.8X, 1.9X, and 11.1X compared to AttAcc, GPU+HBM-PIM, and PIM-only, respectively

Energy Analysis



PAPI improves energy efficiency by 3.4X, 3.4X, and 1.2X compared to AttAcc, GPU+HBM-PIM, and PIM-only, respectively

More in the Paper

- **Details on PAPI's implementation**
  - PAPI's heterogeneous architecture
  - PAPI's runtime scheduler
  - System integration
  - Data partitioning across PIM devices (both Attn-PIM & FC-PIM)

- **Detailed evaluation results**
  - PAPI's speedup across different RLP & TLP levels
  - Ablation study for PAPI's speedup

- **Area/power analysis**

More in the Paper

PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System

Yintao He[1,2] Haiyu Mao[3,4] Christina Giannoula[5,6,4] Mohammad Sadrosadati[4]
Juan Gómez-Luna[7] Huawei Li[1,2] Xiaowei Li[1,2] Ying Wang[1] Onur Mutlu[4]
[1]SKLP, Institute of Computing Technology, CAS [2]University of Chinese Academy of Sciences [3] King's College London
[4]ETH Zürich [5]University of Toronto [6]Vector Institute [7] NVIDIA

https://arxiv.org/pdf/2502.15470

Outline

1 Background

2 Observations & Motivation

3 PAPI's Overview

4 PAPI's Implementation

5 Evaluation

6 Conclusion

Conclusion

Key Findings

1. LLM kernels have different computation and memory bandwidth demands across different RLP & TLP levels

2. Memory-bound kernels exhibit different computation demands depending on kernel type

3. LLM kernels have dynamically changing RLP and TLP levels

Conclusion

Key Contribution

PAPI

A new **PIM-enabled heterogeneous** system design that caters to **varying demands** of LLM kernels by scheduling them **dynamically** to computation-centric processing units and hybrid PIM units

Key Results

PAPI largely improves both performance and energy efficiency over best prior LLM decoding system

- **1.8**× speedup
- **3.4**× energy efficiency increase

PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System

Yintao He Haiyu Mao Christina Giannoula
Mohammad Sadrosadati Juan Gómez-Luna Huawei Li
Xiaowei Li Ying Wang Onur Mutlu

ASPLOS 2025

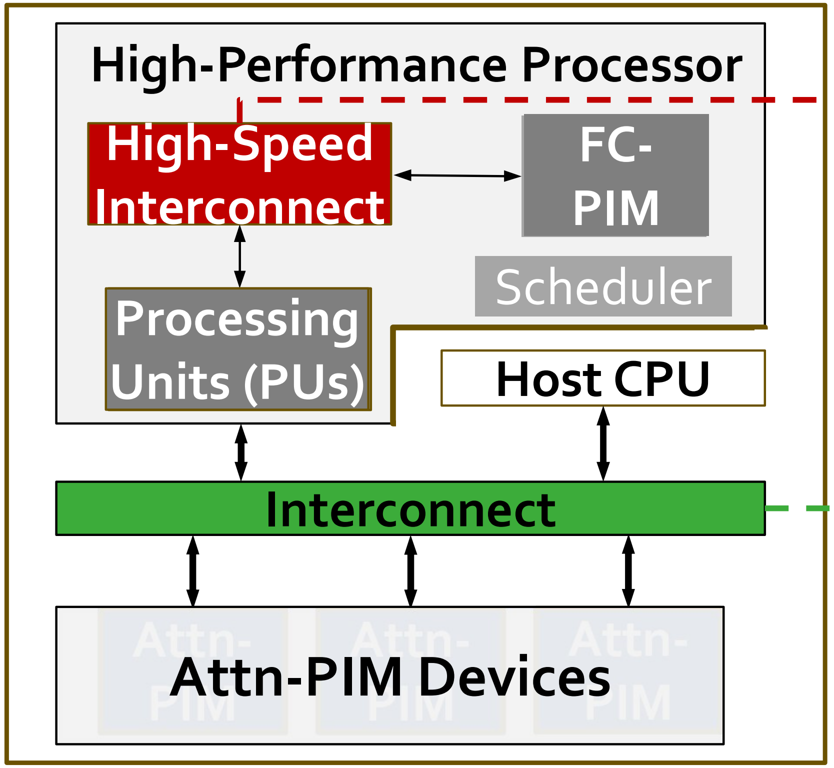
SAFARI

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES
KING'S College LONDON
UNIVERSITY OF TORONTO
ETH Zürich
NVIDIA

Backup Slides

- Interconnections in PAPI
- Identify memory-boundedness threshold
- Energy breakdown & power analysis
- Estimated arithmetic intensity
- Execution time breakdown in LLM decoding
- The process of dynamic scheduling

Interconnections in PAPI



FC-PIM units & PUs:
High-speed interconnect

FC kernels with **large weight parameters' transfers** from FC-PIM to PUs

Attn-PIM & high-performance processor:
a **standard** interconnect like PCIe

Attention kernel involves
small data transfers (Byte-level Q vector)

Identify memory-boundedness threshold

We evaluate when FC kernels becomes memory-bound by testing different configurations

① Run FC kernels

- With different TLP and RLP levels
- On PUs and FC-PIM unit, respectively

② Measure arithmetic intensity and execution time for each case

③ Figure out threshold

- Under what conditions FC-PIM unit **faster** than PUs

Energy Breakdown & Power Analysis

- **DRAM access** costs the **most energy consumption** when executing the FC kernels
- Leveraging **data reuse** can reduce the number of **DRAM access**

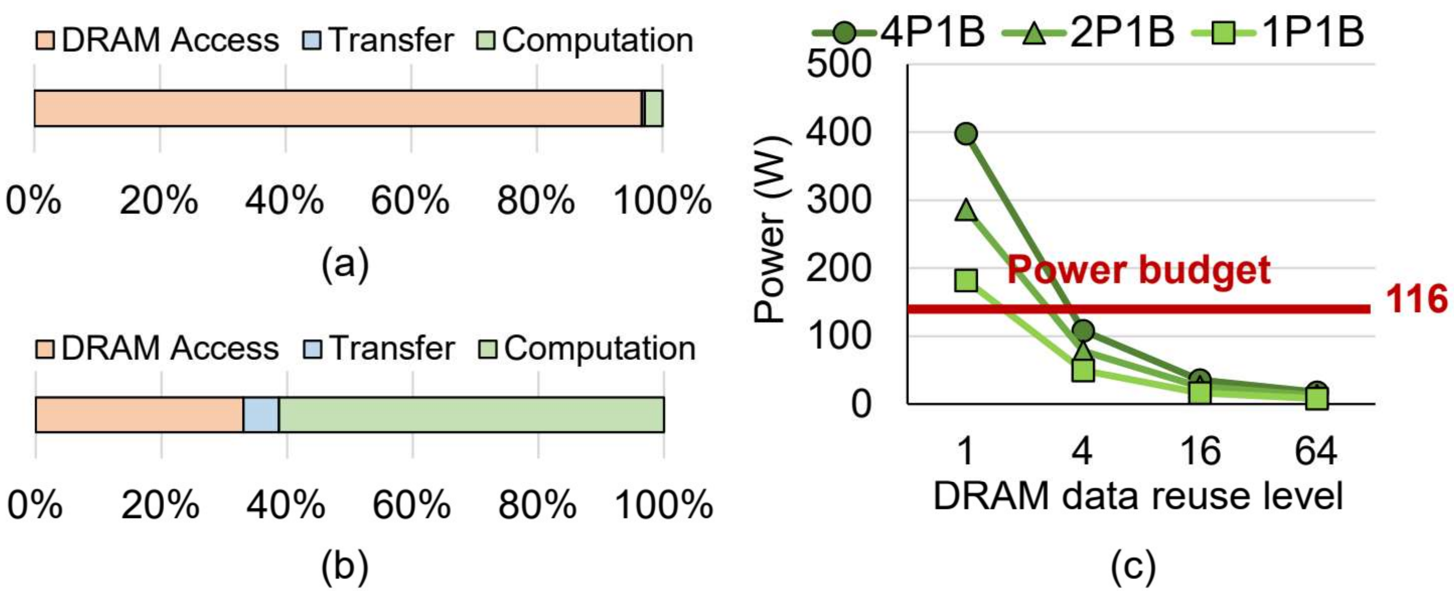


Figure 7: (a) Energy breakdown of PIM for executing the FC kernel with no DRAM data reuse. (b) Energy breakdown of PIM for executing the FC kernel when one DRAM access (i.e., an activated DRAM row) is used 64 times for computation (i.e., data reuse level = 64). (c) Power consumption of PIM architecture with different data reuse levels and different numbers of FPUs per bank.

Estimated Arithmetic Intensity

Arithmetic Intensity ≈ RLP × TLP

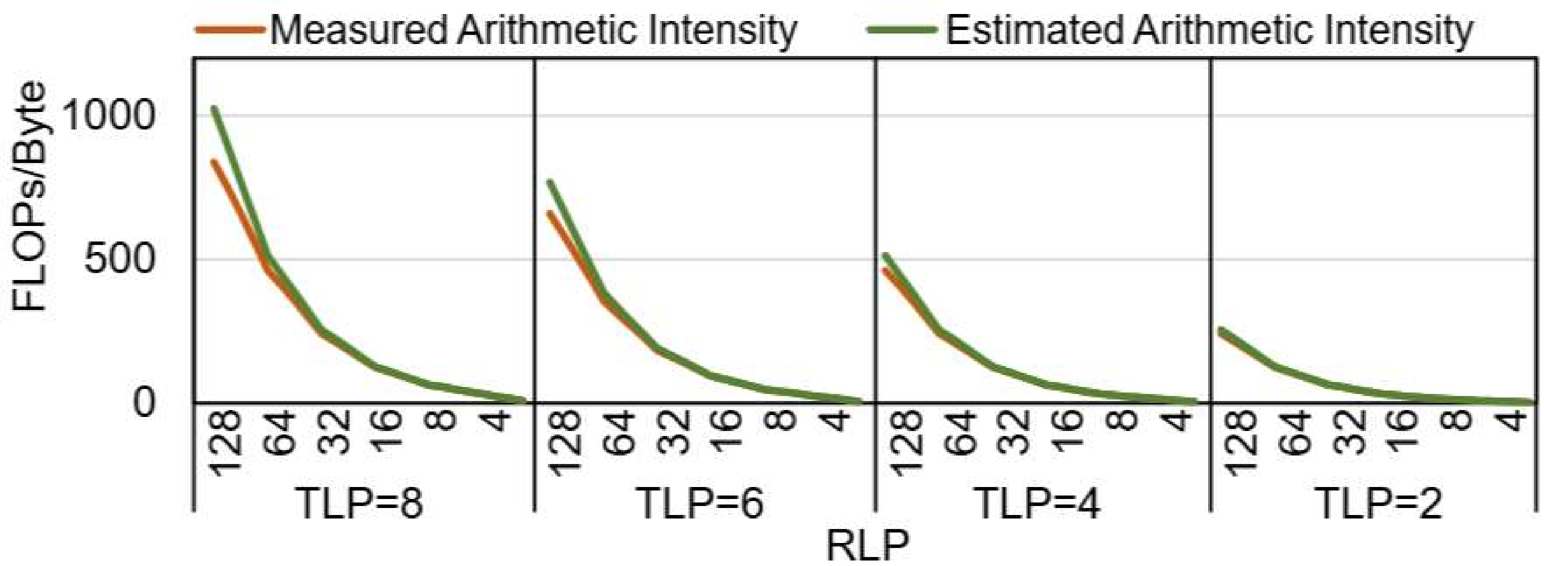


Figure 6. Actual measured arithmetic intensity and the estimated arithmetic intensity for FC kernels in the GPT-3 66B model.

Execution Time Breakdown in LLM Decoding

- LLM decoding in **pure PIM system**:
  - **Attention kernels: 0.3~1%** of the execution time
  - **FC kernels dominate** the total execution time
- Similar within the **PIM-enabled heterogeneous** LLM systems

The **execution time breakdown** per token in the decoding stage
Of **LLaMA-65B model (Initial RLP = 4, TLP=4)**

It is valuable to **speedup FC kernels**

The Process of Dynamic Scheduling

- Assume the memory-boundedness threshold **α=3** in this case

Output tokens of requests

Today	is	sunny
It	is	a
Have	a	nice
How	are	you
Here	is	a

RLP	5	5	5
TLP	1	1	1
Estimated value	5	5	5
Reschedule	×	×	×
RESULT	-	-	-