PolyUFC: **Poly**hedral Compilation *Meets* Roofline Analysis for **U**ncore **F**requency **C**apping

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

IEEE/ACM CGO, 2026

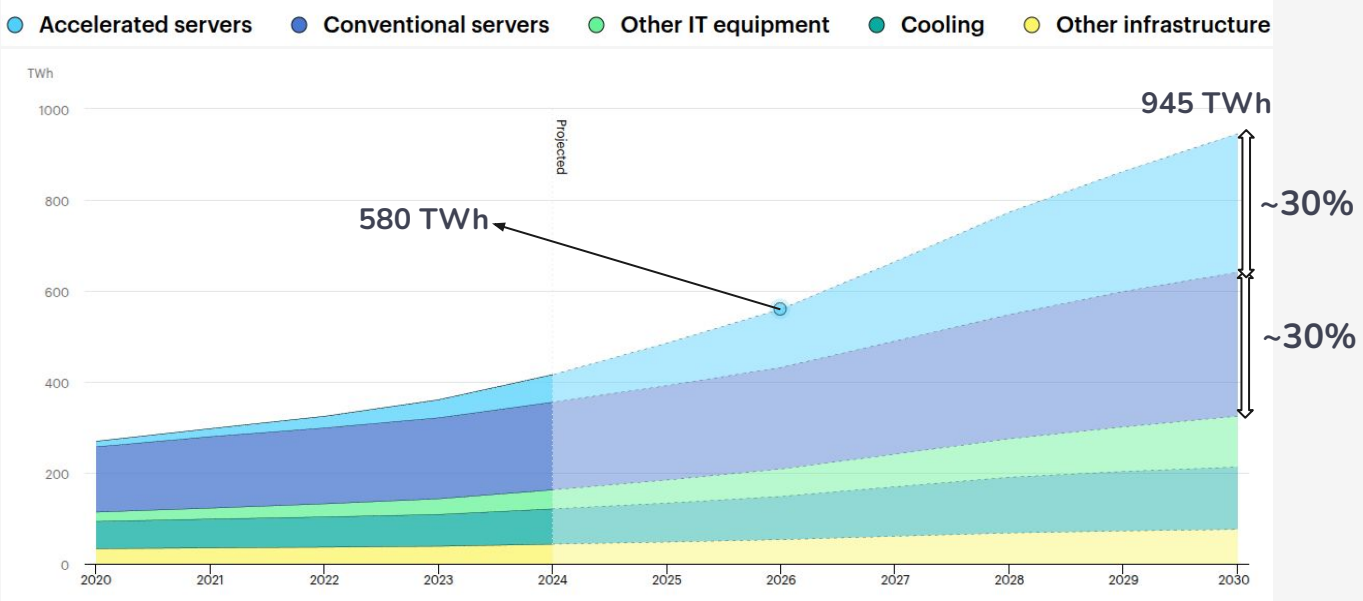
Nilesh Rajendra Shah,
MVVS Manoj Kumar, Dhairya Baxi and
Ramakrishna Upadrasta

Outline

- Motivation and Introduction
  - Opportunities for Power Savings in CPUs
  - Compiler Driven Frequency Capping for Uncore
- Parametric Mathematical Model for Performance/Power
  - Cache Model and Roofline based Characterization
- Experimental Evaluation
  - Characterization, Compile Time, EDP
- Conclusions

1 Motivation and Introduction

Power demand in the age of AI

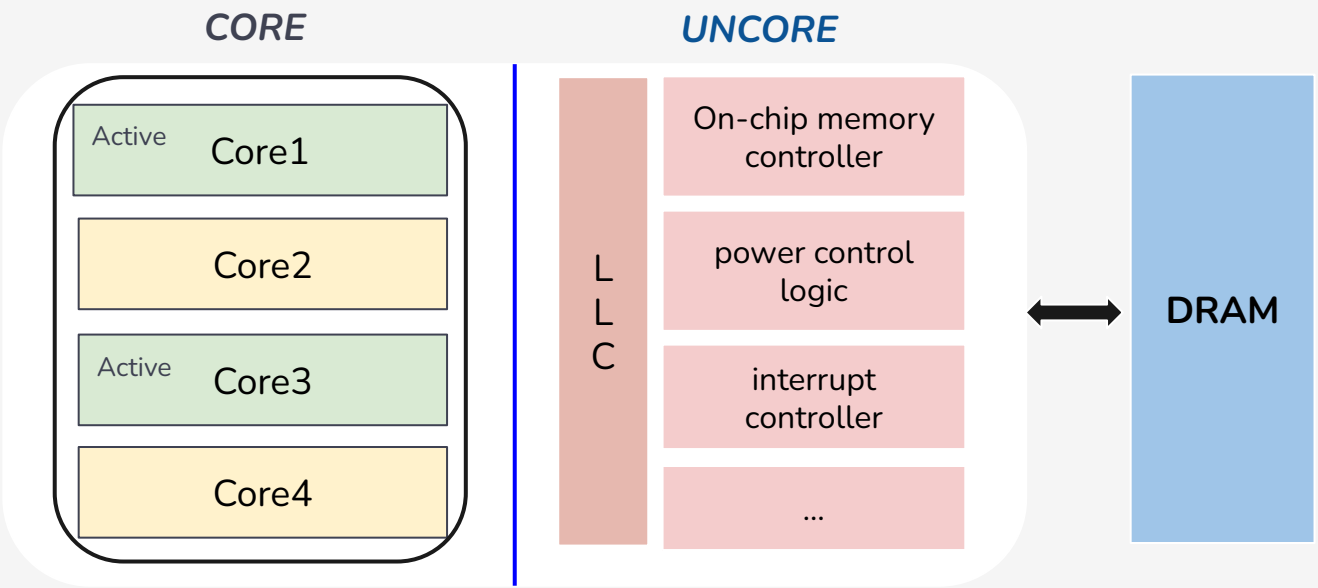


★ **Power consumption** is a primary concern in modern servers and data centers

- ~ **3%** of global electricity by 2030 [1] → ~**0.9%** by conventional servers **(CPU based)**

[1] International Energy Agency (2024): https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai

CPU Components (Core vs. Uncore)



- Minimum required ***uncore*** logic remains **active** as long as **any core** is active
- Uncore power is a key component of overall system power (upto **30%**)

Diversity in Bandwidth Requirements: ML and HPC Workloads

Observations: on ML kernels

➔ Most ML kernels *do not* require **peak bandwidth**

- **Example:** tiled large **`conv2d`**, **`matmul`**`[1,2]`

➔ Some kernels are *strongly* **bandwidth-bound**

- **Example:** tiled **matrix-vector** products, small conv2d [1]

[1] Wang, Yu et al. "A systematic methodology for analysis of deep learning hardware and software platforms." MLSys. 2020.
[2] Zeshan Chishti and Berkin Akin. 2019. Memory system characterization of deep learning workloads. MEMSYS '19

CPUs: Current HW Uncore Power Management

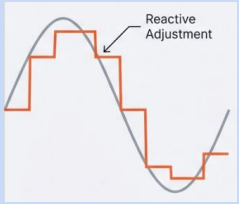
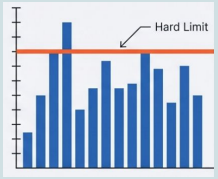
Fact: Hardware makes **uncore** frequency change decisions

- based only on **core** hardware information (active cores, freq, ...)
- **without** analyzing the program!

	Hardware	Compiler
Compute bound	*Over Provision* ❌	*Under Provision* ✔
Bandwidth Bound	*Under Provision* ❌	*Over Provision* ✔

Necessity for a precise compiler based solution!

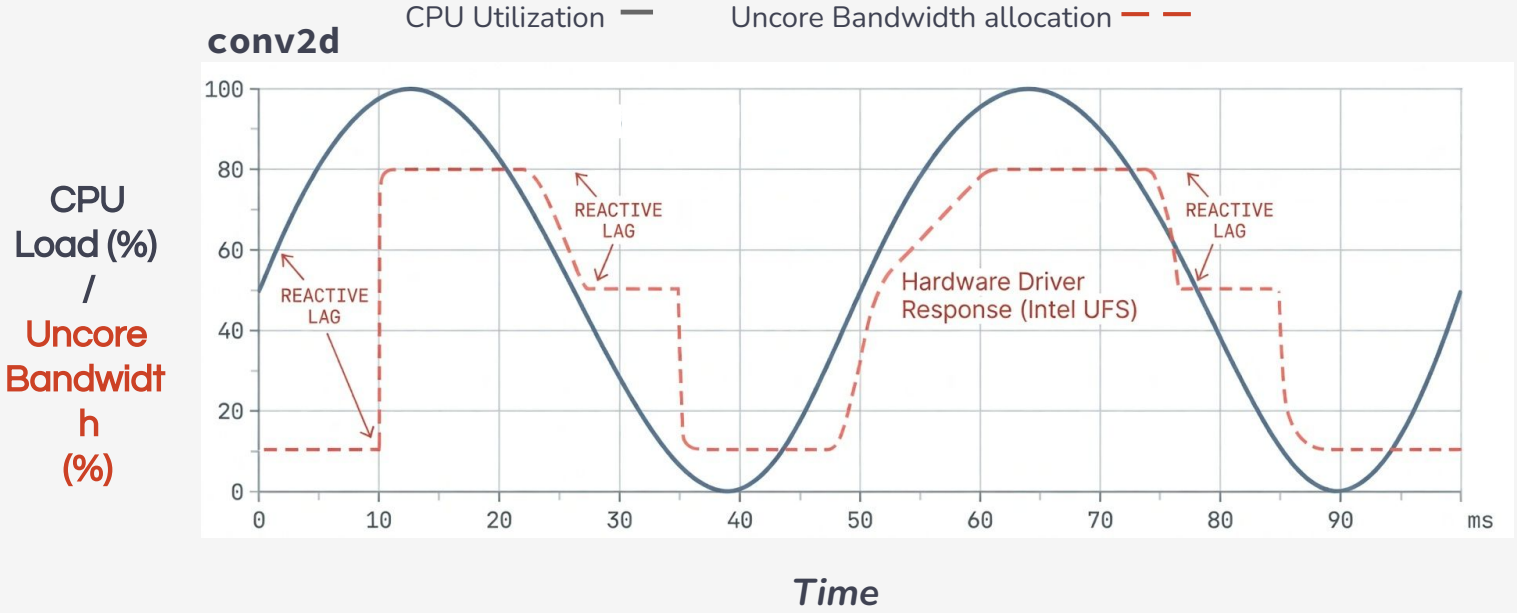
CPUs: Some Power Management Techniques

		CORE	UNCORE
	SCALING	**DVFS/DFS** SW: (Compiler) Jimborean et al. [1]; (Compiler) Bao et al. [2] HW: (Runtime) Haj-Yahya [3], ...	**DUFS** HW: (Runtime) Gholkar el al. [4]
	CAPPING	Power **Capping** Petoumenos et al. [5]	Frequency **Capping** SW: (Compiler) **PolyUFC**

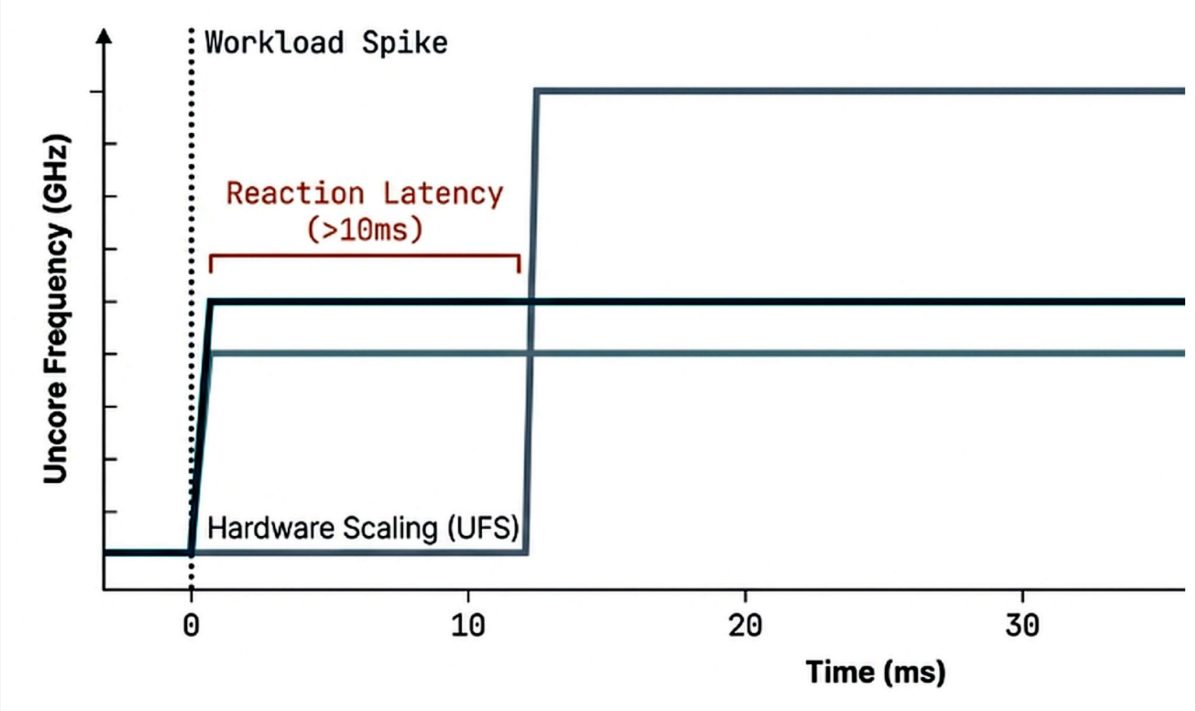
[1] A. Jimborean et al. "*Fix the code. don't tweak the hardware: A new compiler approach to voltage-frequency scaling*", in CGO'14
[2] W. Bao, et al. "*Static and dynamic frequency scaling on multicore cpus,*" in TACO'16.
[3] J. Haj-Yahya et al., "SysScale: Exploiting Multi-domain Dynamic Voltage and Frequency Scaling for Energy Efficient Mobile Processors," in ISCA'20.
[4] N. Gholkar et al."*Uncore power scavenger: a runtime for uncore power conservation on hpc systems,*" in SC '19.
[5] P. Petoumenos et al., "Power Capping: What Works, What Does Not," ICPADS, 2015

Uncore (Hardware) Scaling: How it works?

Hardware adjusts *uncore* scaling frequency f_s (and Voltage) based on CPU **utilization**



Uncore Frequency Capping: How It works?

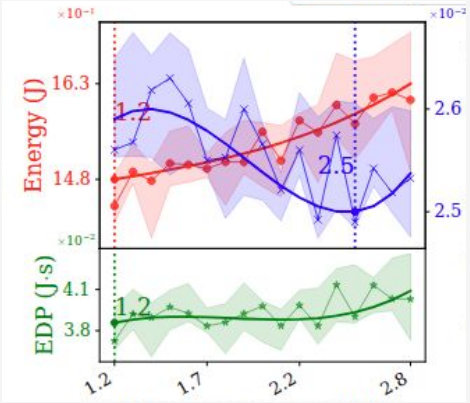
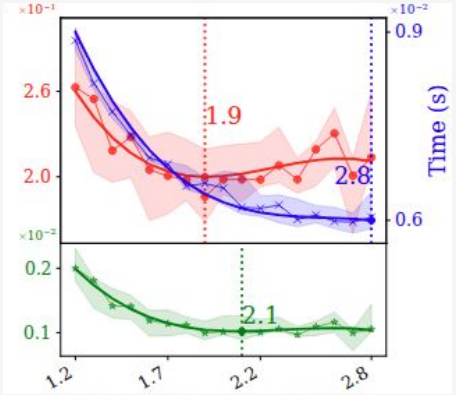


Overprovisioning
(**Max** freq for f_s)

Capping freq (f_c)

Selected freq after Cap

Experiment: Compiler driven UFC

Kernel	`conv2d` (convnext)	`mvt` (Polybench)
Energy/Time TradeOff		
Category	**Compute Bound (CB)**	**Bandwidth Bound (BB)**
LLC Bandwidth Requirement	**Low**	**High**
Compiler based Strategy (using capping)	decrease $\mathbf{f_c}$	increase $\mathbf{f_c}$

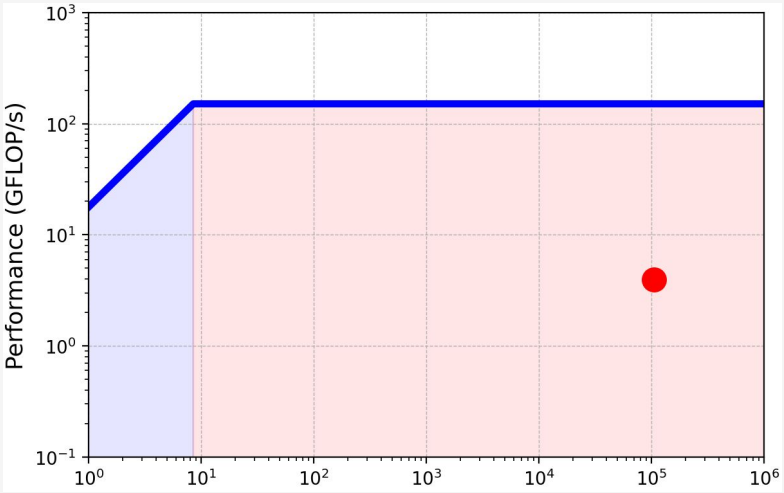
Outline

- Motivation and Introduction
  - Opportunities for Power Savings in CPUs
  - Compiler Driven Frequency Capping for Uncore
- Parametric Mathematical Model for Performance/Power
  - Cache Model and Roofline based Characterization
- Experimental Evaluation
  - Characterization, Compile Time, EDP
- Conclusions

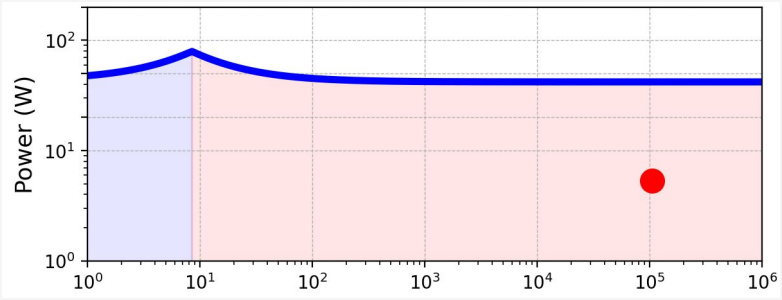
2 Performance and Power Characterization using Roofline Analysis

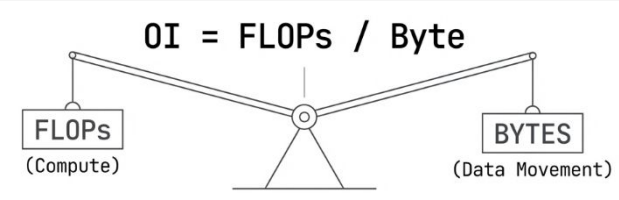
Background: Roofline Analyses for Bottleneck Characterization

Original Roofline Model [1]



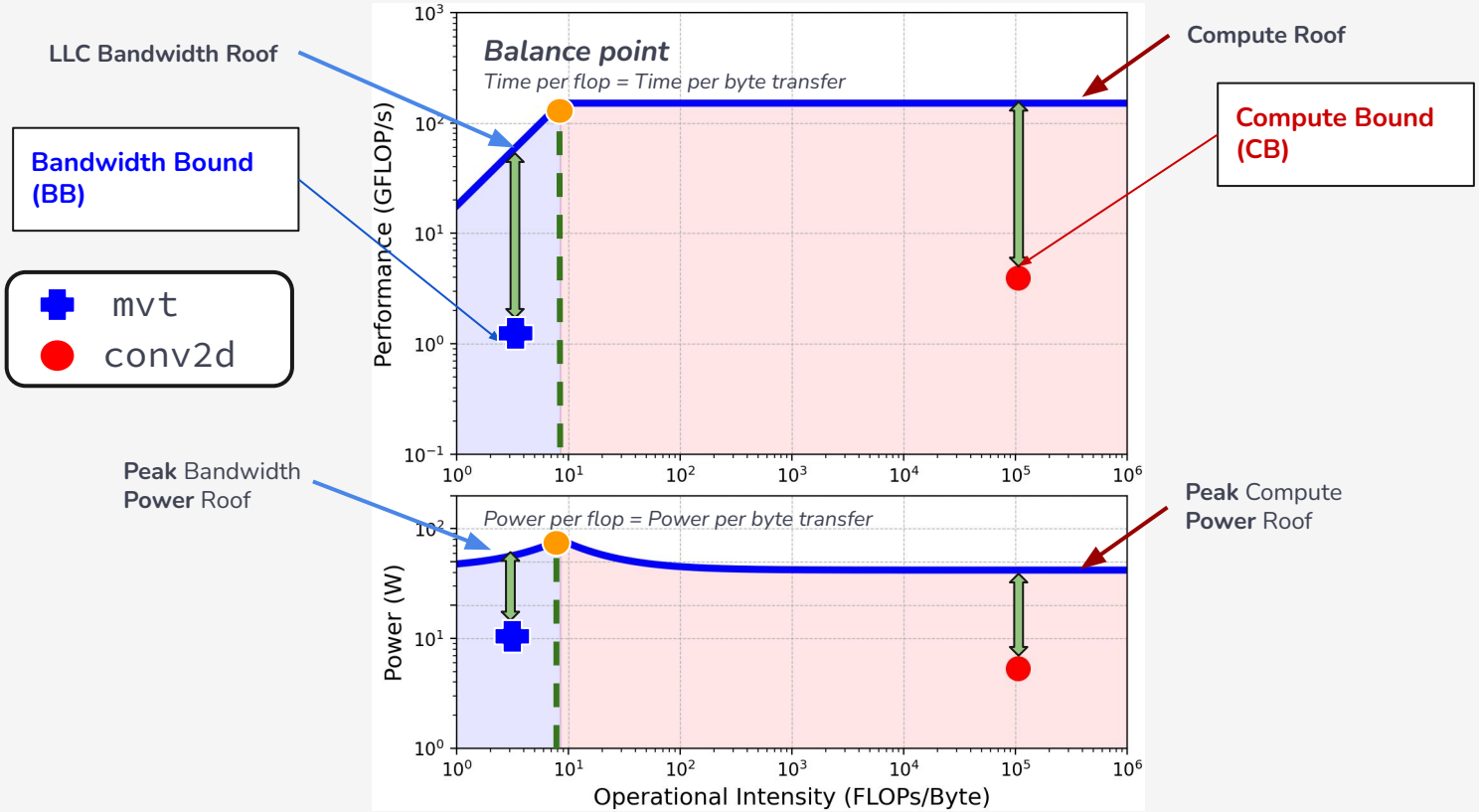
Energy/Power Roofline Model [2]





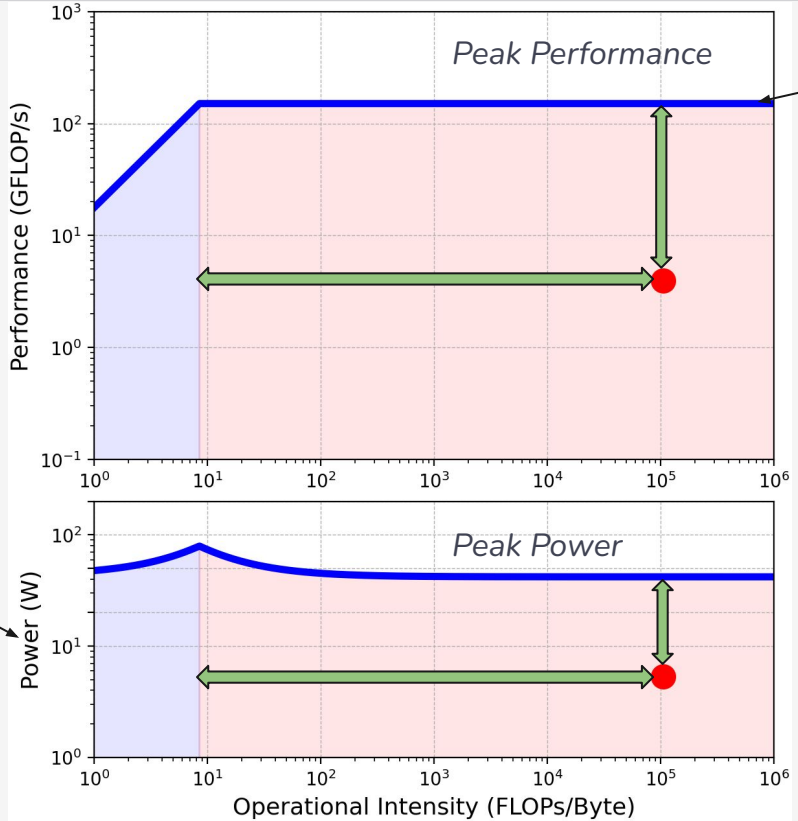
[1] Williams, A. Waterman, and D. Patterson, "*Roofline: an insightful visual performance model for multicore architectures,*" Commun. ACM, 2009.
[2] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "*A roofline model of energy,*" IPDPS, 2013

Background: Roofline Analyses for Bottleneck Characterization



CB: How can Rooflines Analyses help in UFC?

conv2d

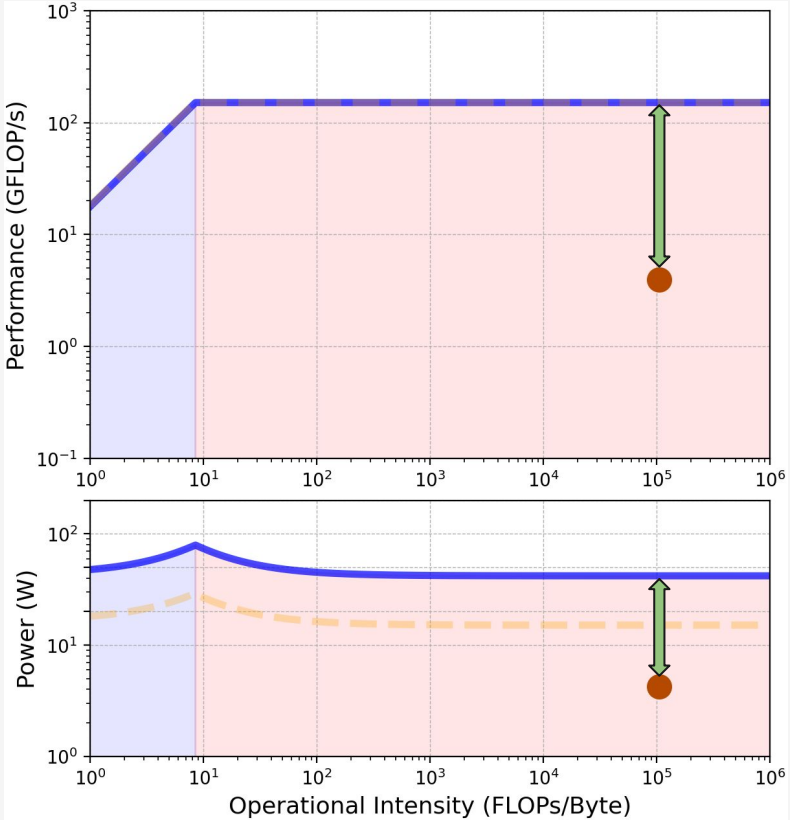


HW Selected Scaling frequency $\mathbf{f_s}$

Dynamic uncore power

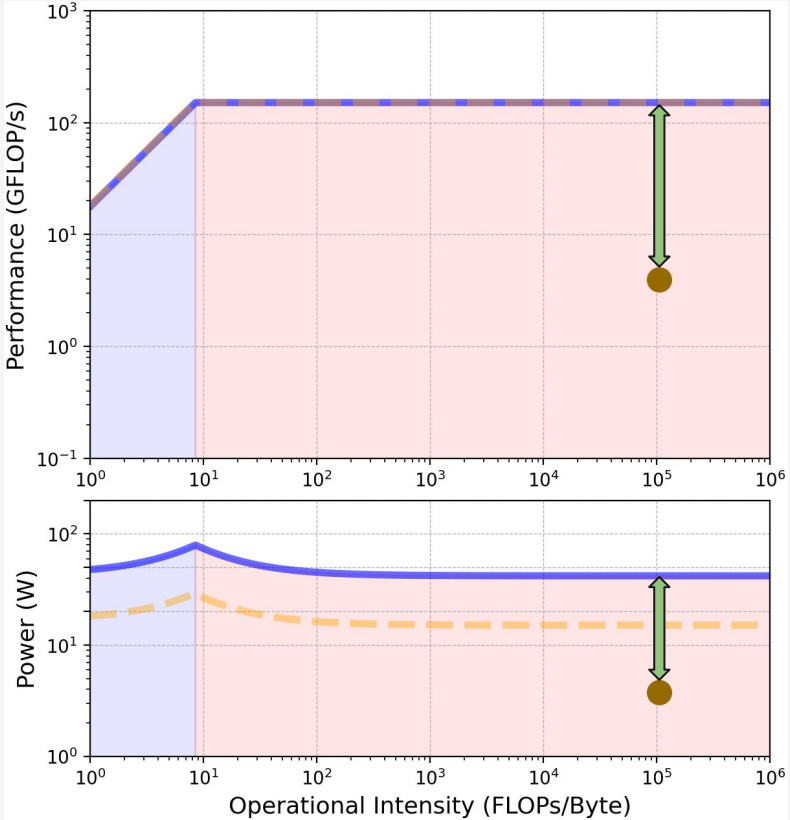
CB: How can Rooflines Analyses help in UFC?

conv2d



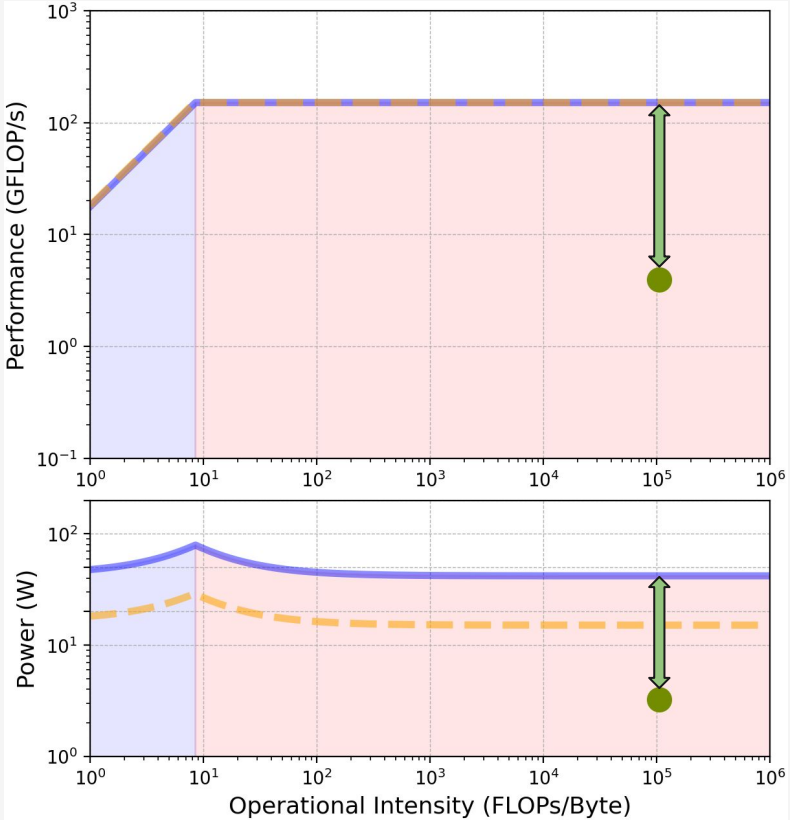
CB: How can Rooflines Analyses help in UFC?

conv2d



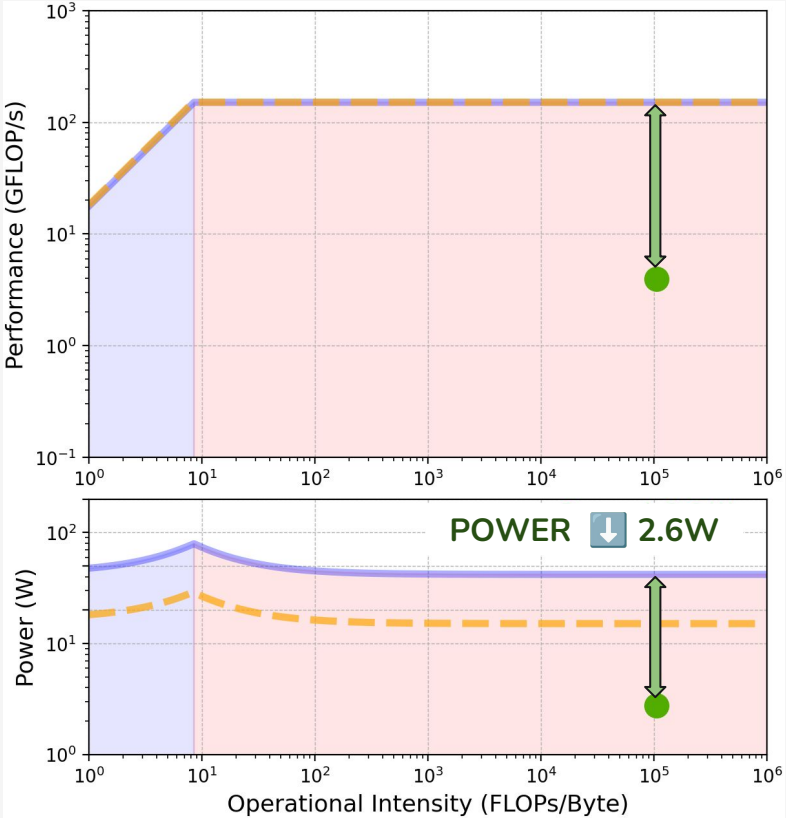
CB: How can Rooflines Analyses help in UFC?

conv2d



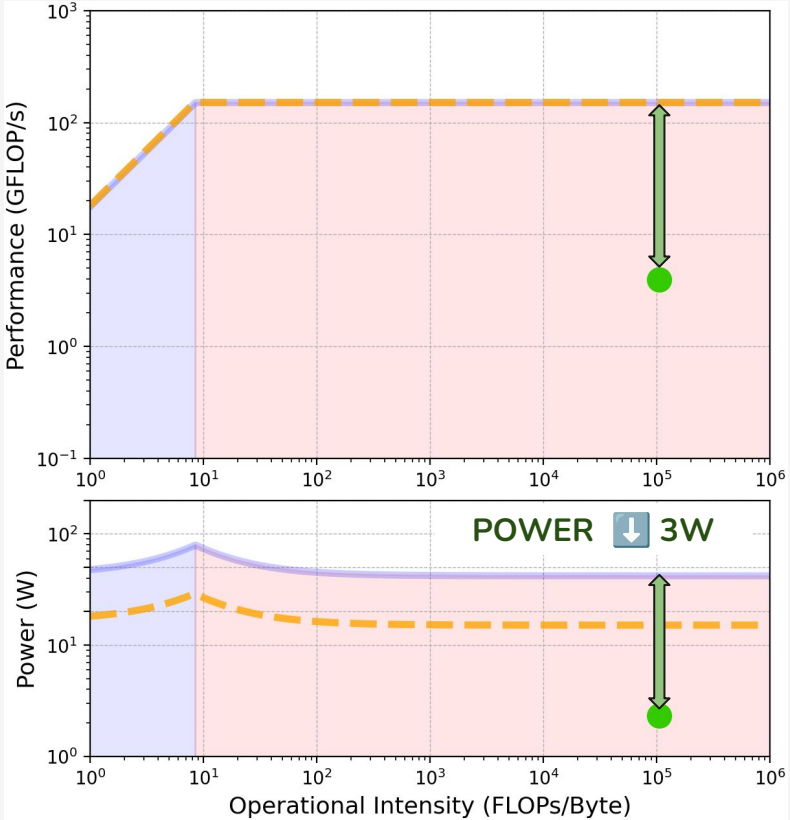
CB: How can Rooflines Analyses help in UFC?

conv2d



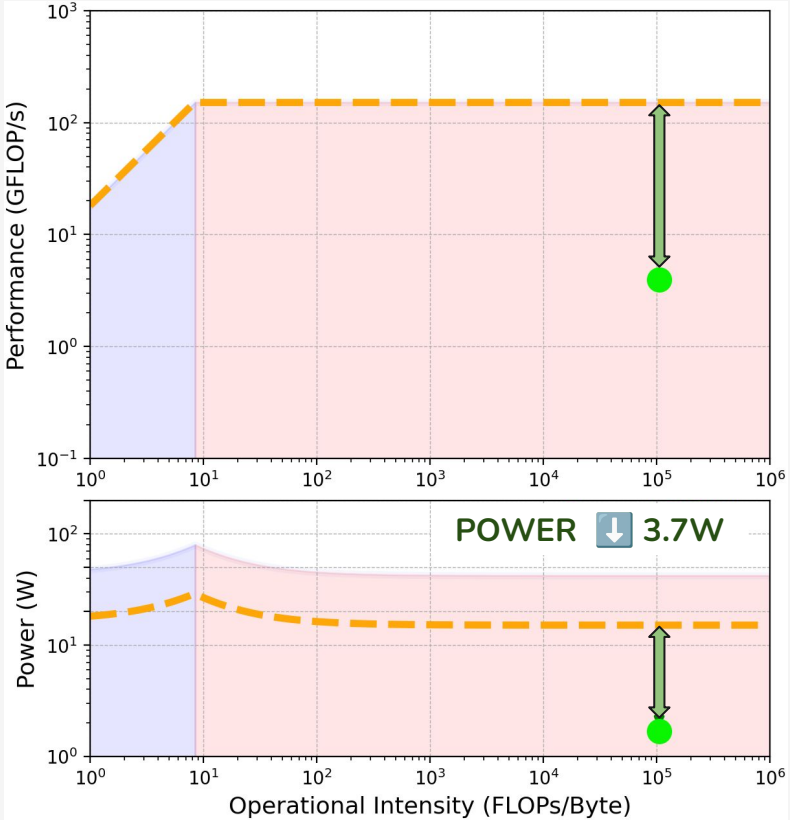
CB: How can Rooflines Analyses help in UFC?

conv2d



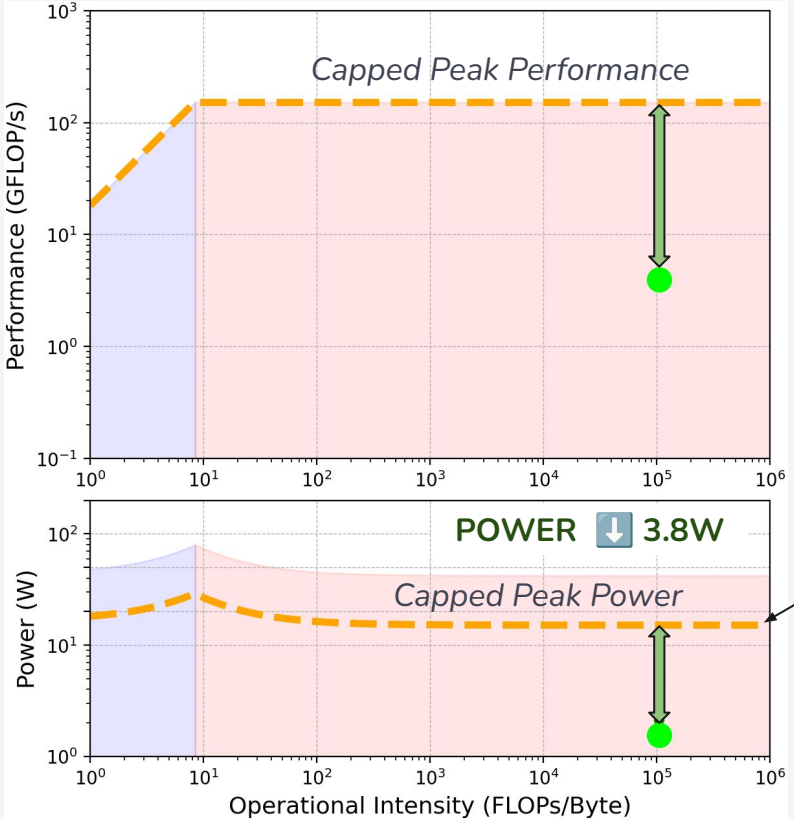
CB: How can Rooflines Analyses help in UFC?

conv2d



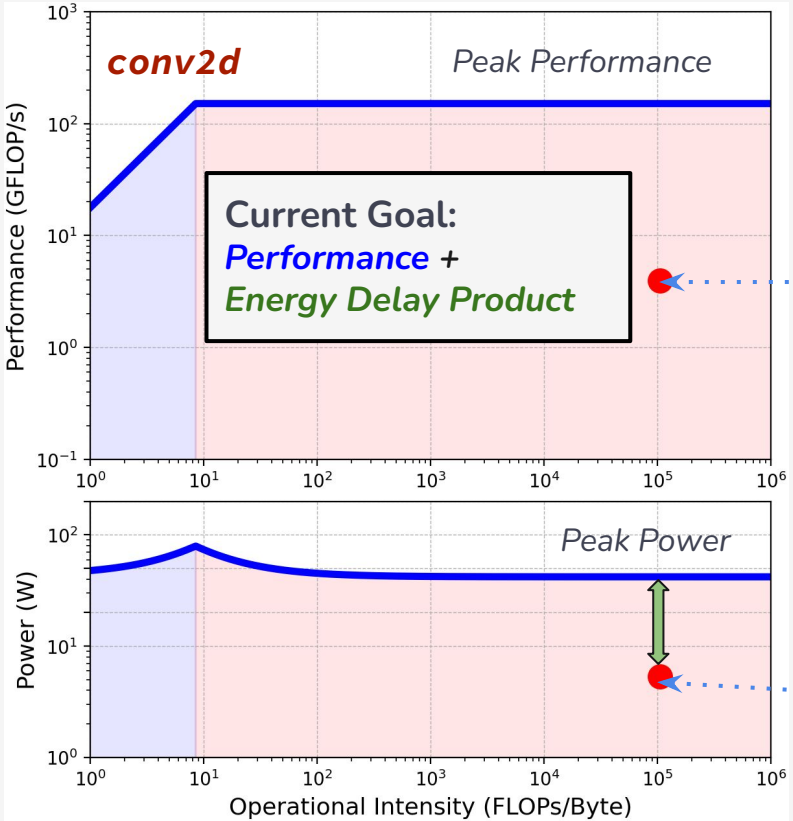
CB: How can Rooflines Analyses help in UFC?

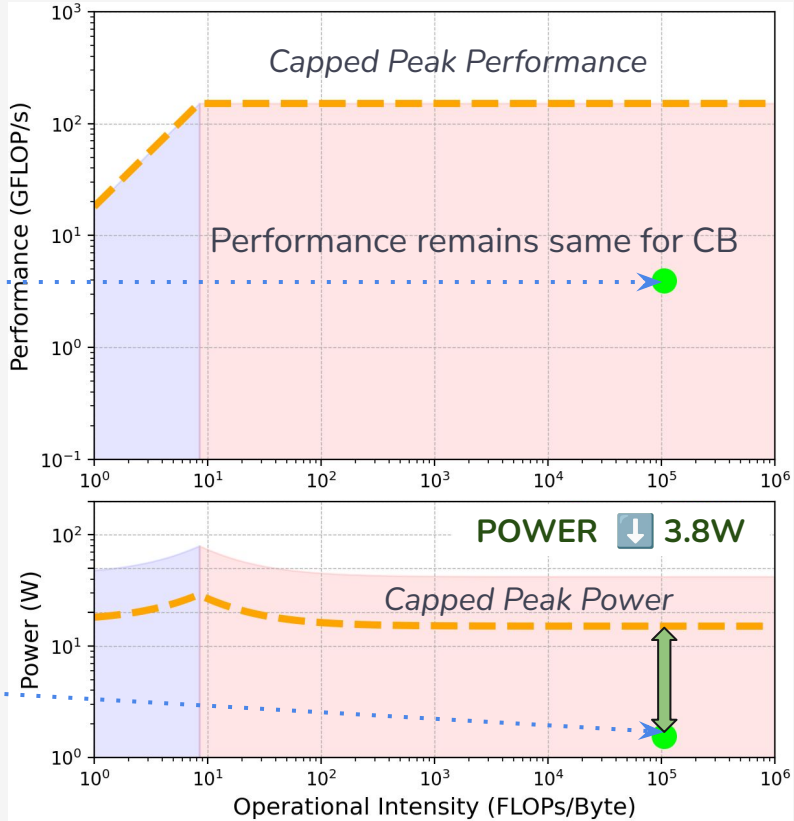
conv2d



Capping frequency $\mathbf{f_c}$

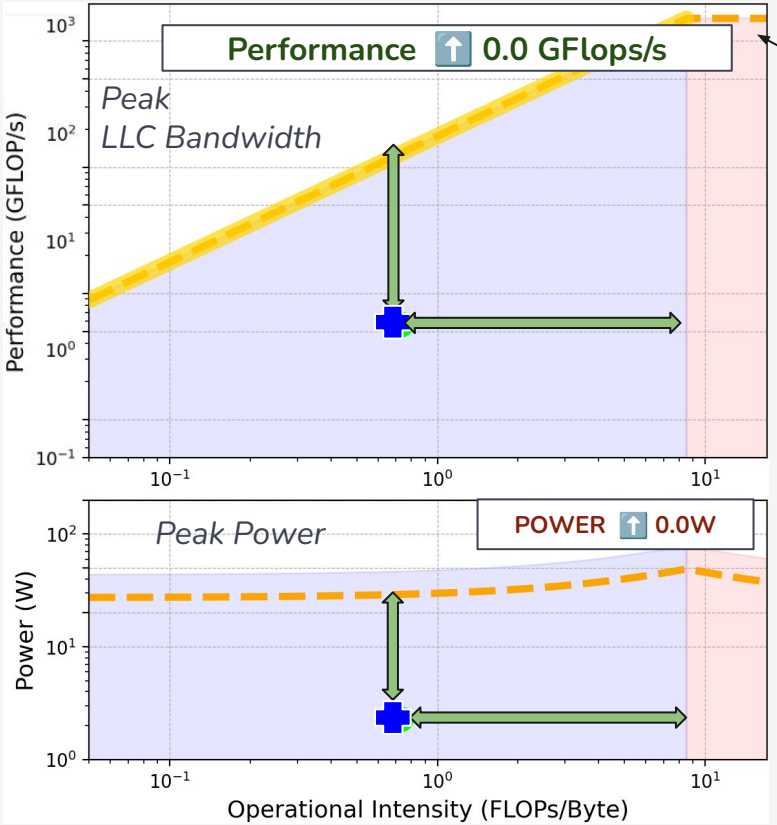
CB: How can Rooflines Analyses help in UFC?





BB: How can Rooflines Analyses help in UFC?

mvt

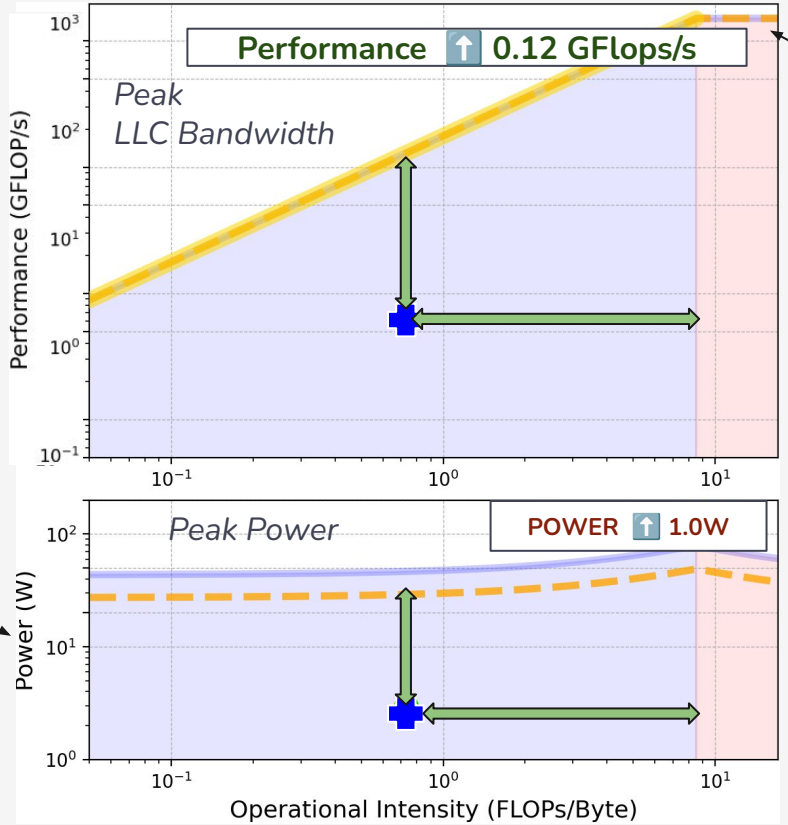


HW Selected Scaling frequency $\mathbf{f_s}$

Dynamic uncore power

BB: How can Rooflines Analyses help in UFC?

mvt

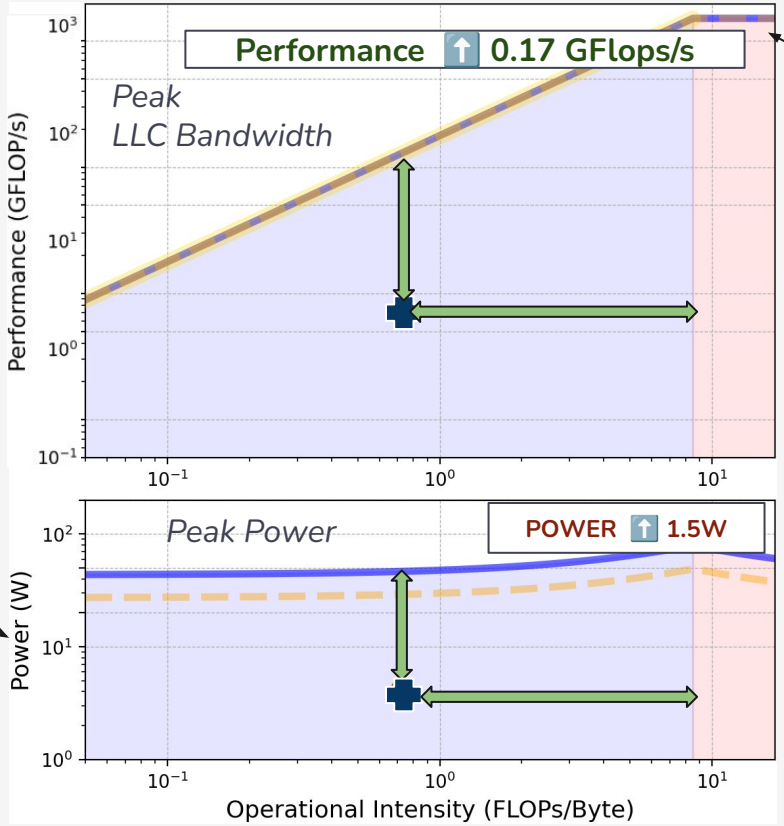


HW Selected
Scaling frequency $\mathbf{f_s}$

Dynamic uncore power

BB: How can Rooflines Analyses help in UFC?

mvt

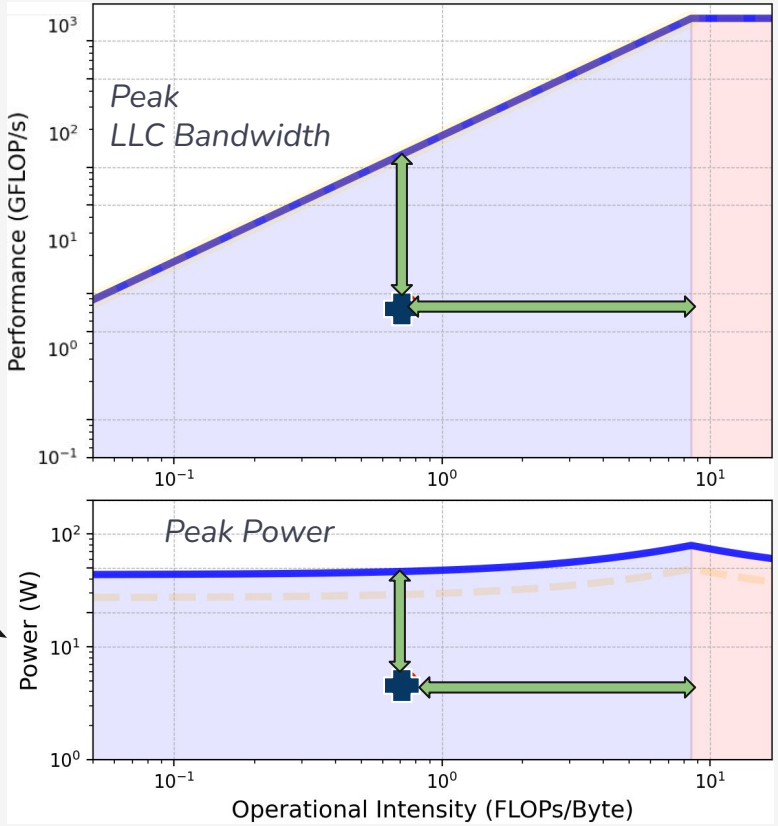


HW Selected
Scaling frequency $\mathbf{f_s}$

Dynamic uncore power

BB: How can Rooflines Analyses help in UFC?

mvt

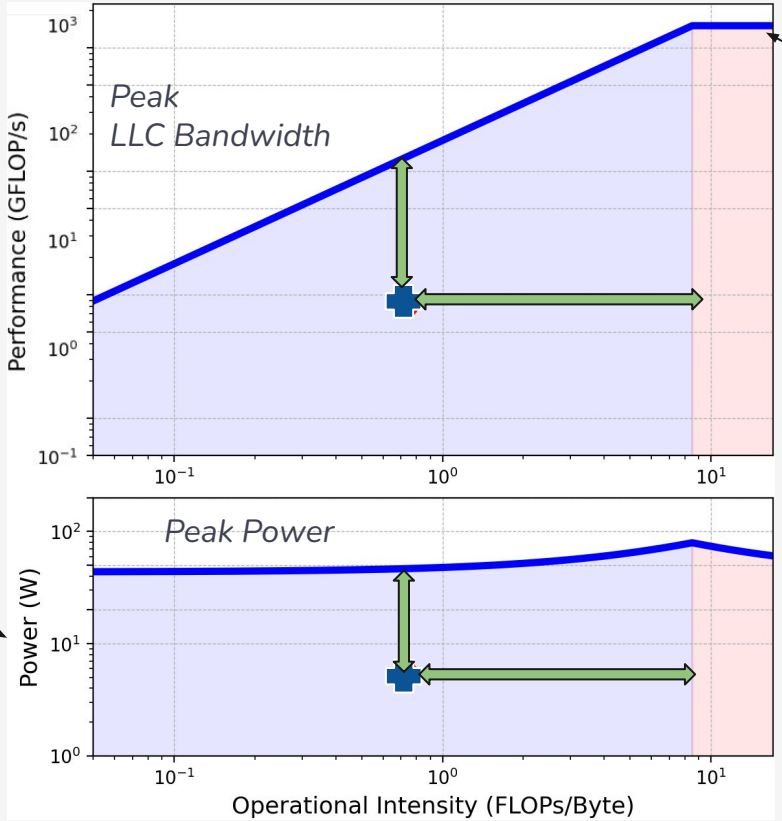


HW Selected
Scaling frequency $\mathbf{f_s}$

Dynamic uncore power

BB: How can Rooflines Analyses help in UFC?

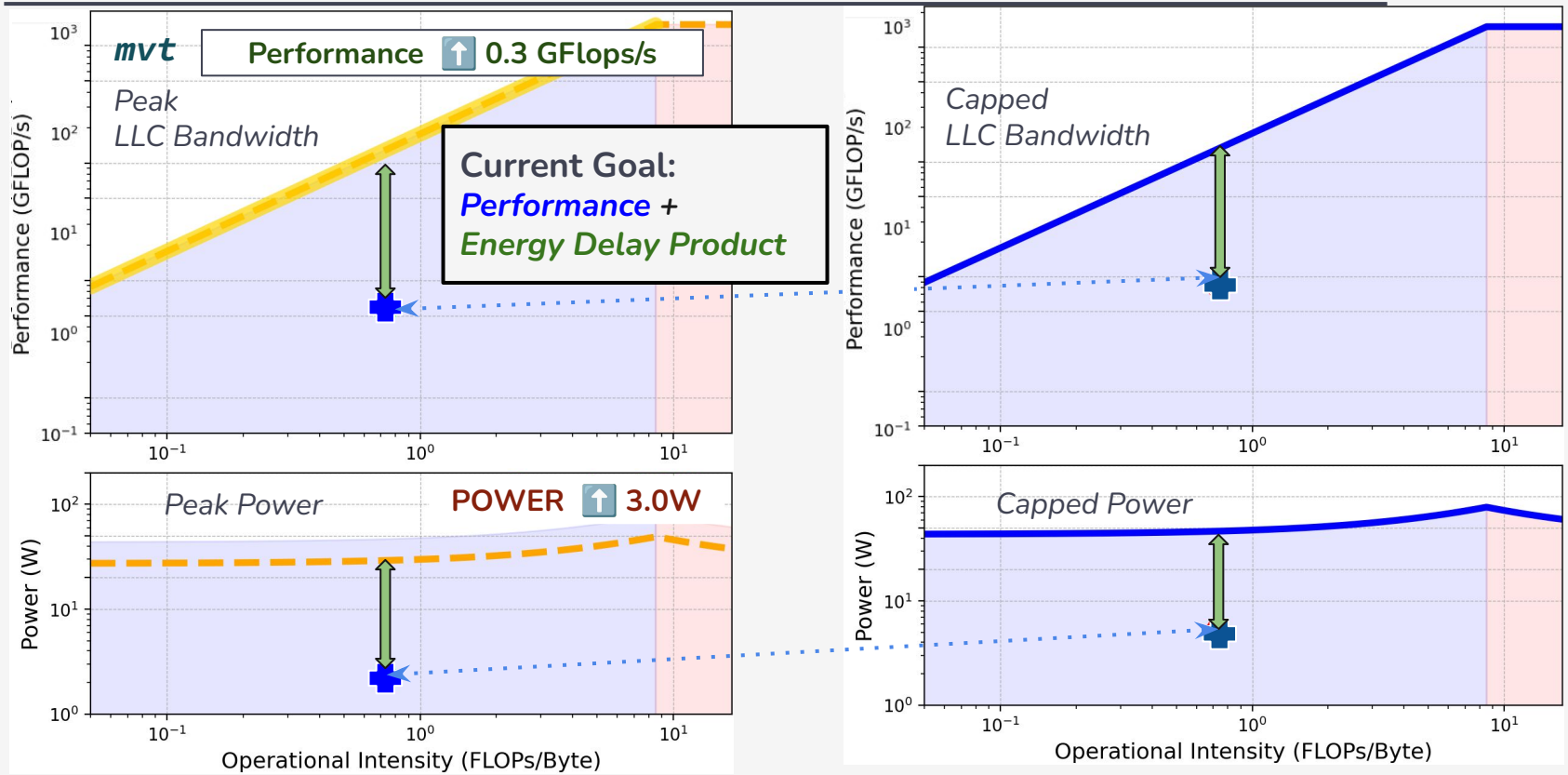
mvt



Compiler selected Capping frequency $\mathbf{f_c}$

Dynamic uncore power

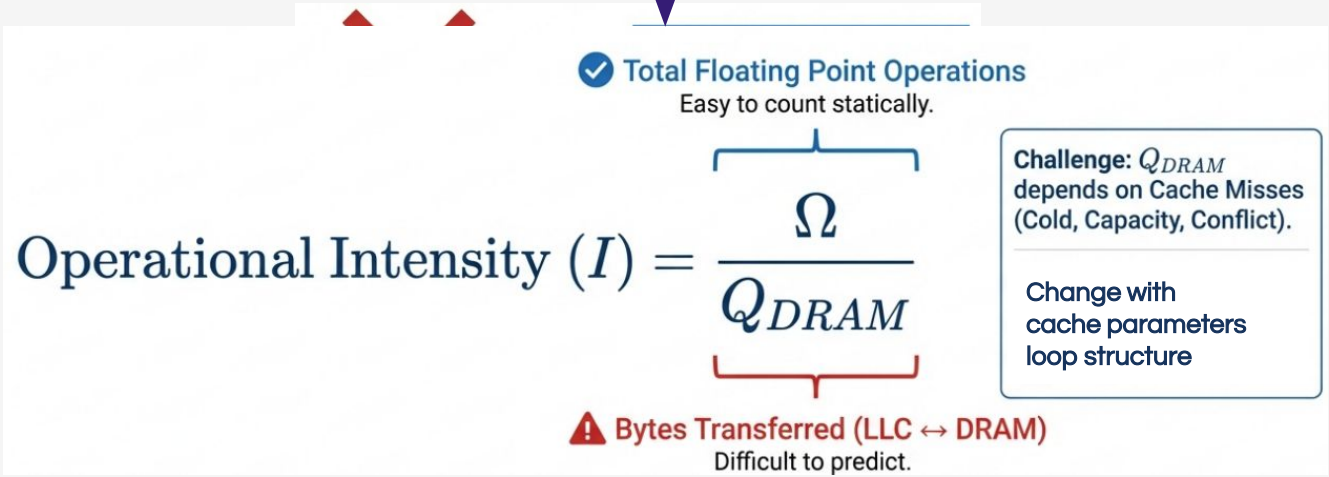
BB: How can Rooflines Analyses help in UFC?



Compile-time Characterization

Can we do ***bottleneck characterization*** (**CB/BB**) using Polyhedral Static Analysis Techniques?

Classify with OI



Compile-time Characterization

Can we do ***bottleneck characterization*** (**CB/BB**) using Polyhedral Static Analysis Techniques?

How to model Performance?

No time overlap!

$$T = T_{compute}(I) + T_{mem}(I, f_c)$$

How to model Total Power?

Model capping frequency for uncore

$$P = P_{static} + P_{core}(I) + P_{uncore}(I, f_c)$$

Operational Intensity (OI) Estimation

➔ Available Techniques

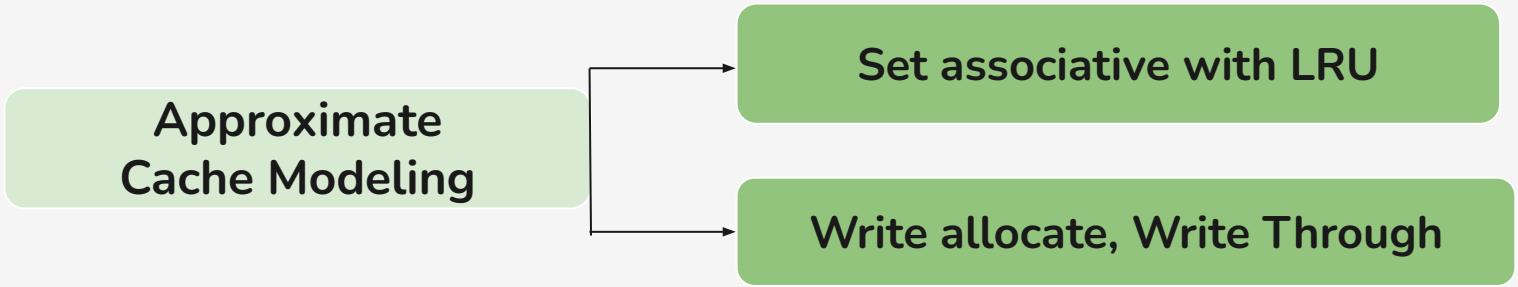
- Use hardware simulation or performance counters
- *Static-analysis based Cache Models:*

  Set-Assoc: Harper et al [TOC'99], PolyFeat [TACO'16], PolyCache [POPL'18]

  Fully-Assoc: HayStack [PLDI'19], BullsEye [TACO'22], Falcon [PLDI'24]

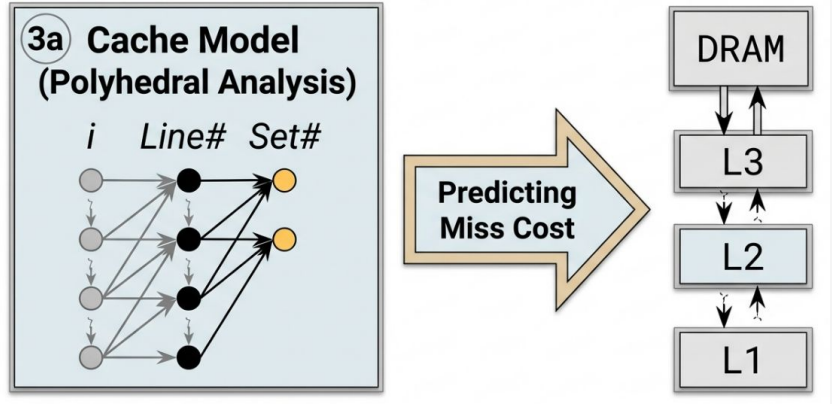
➔ Our Proposal

Approximate multi-level set-associative modeling with LLC shared cache



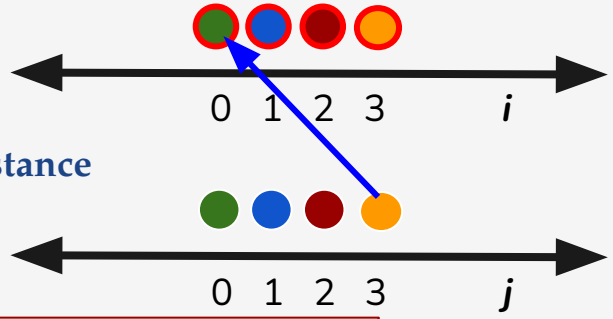
PolyUFC-CM: Set Associative Cache Modeling

- Modeling Associativity: Multi-level caches with *k*-way associativity
- Our Proposal:
  - Consider each cache set as a **LRU** fully-associative cache



LRU Cache

Reuse distance per set




```
double M[4];
for(int i = 0; i < 4; i++)
        S1: M[i] = i;
for(int j = 0; j < 4; j++)
        S2: M[3-j] = j;
```

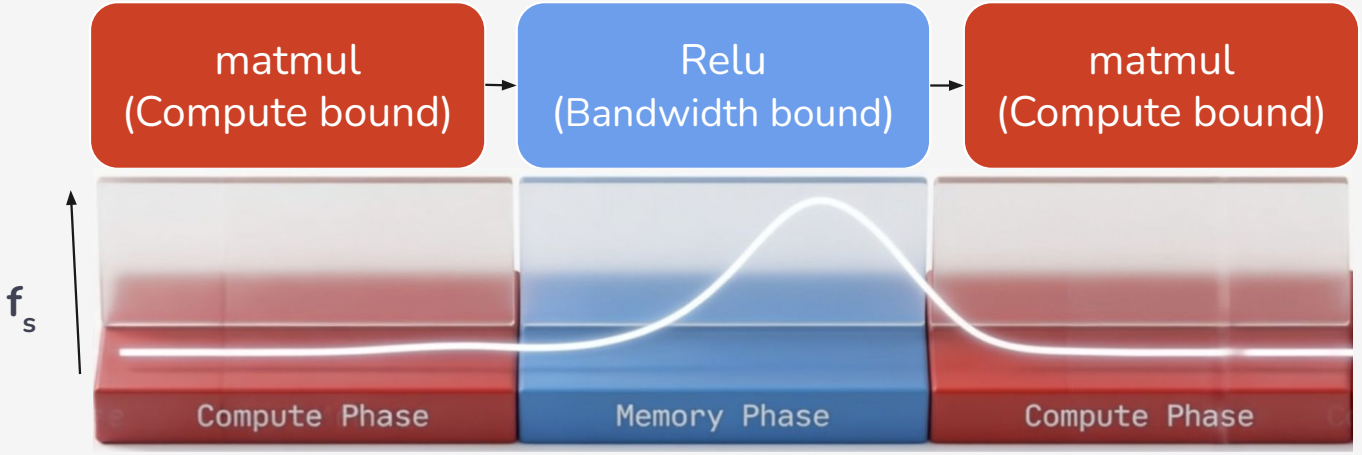
Static analysis

Reuse distance per set

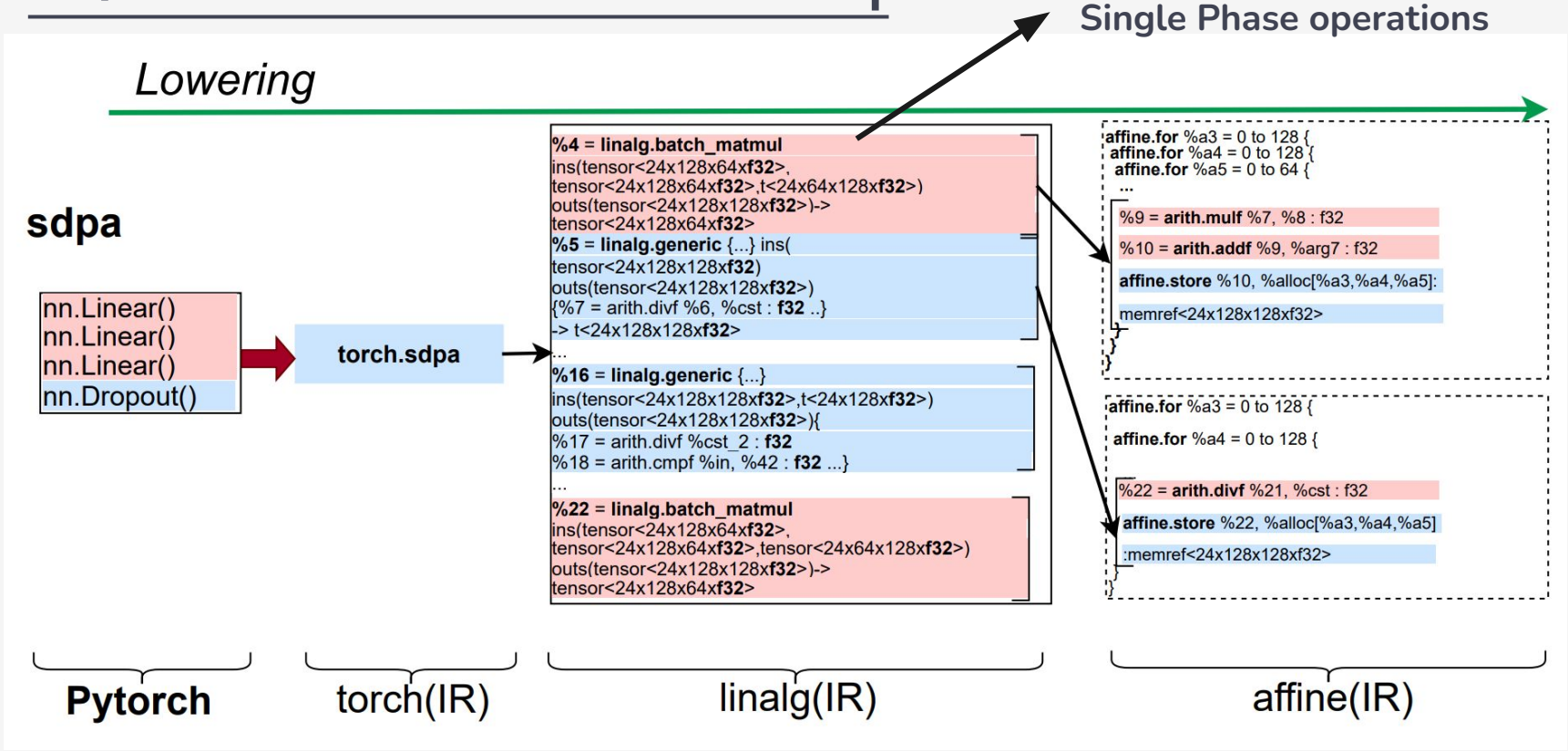
Find pairs that access same memory element in a set

Reuse Distance Map = $(F_{ci} \cap B_{ci}) \circ A_{ci}$

CB/BB Phases in ML kernels

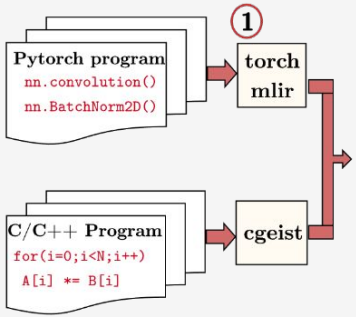


CB/BB Phases in ML kernels: sdpa



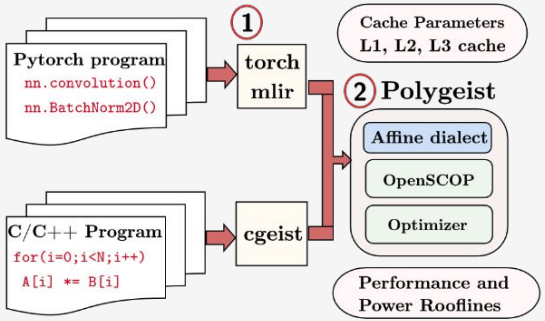
PolyUFC

Compiler Framework for Energy Optimizations



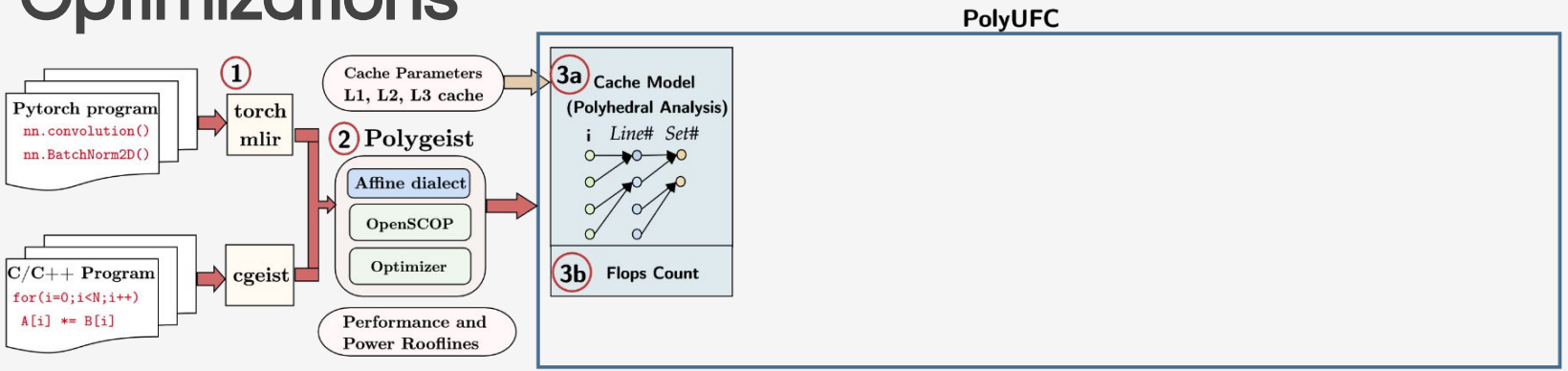
PolyUFC

Compiler Framework for Energy Optimizations



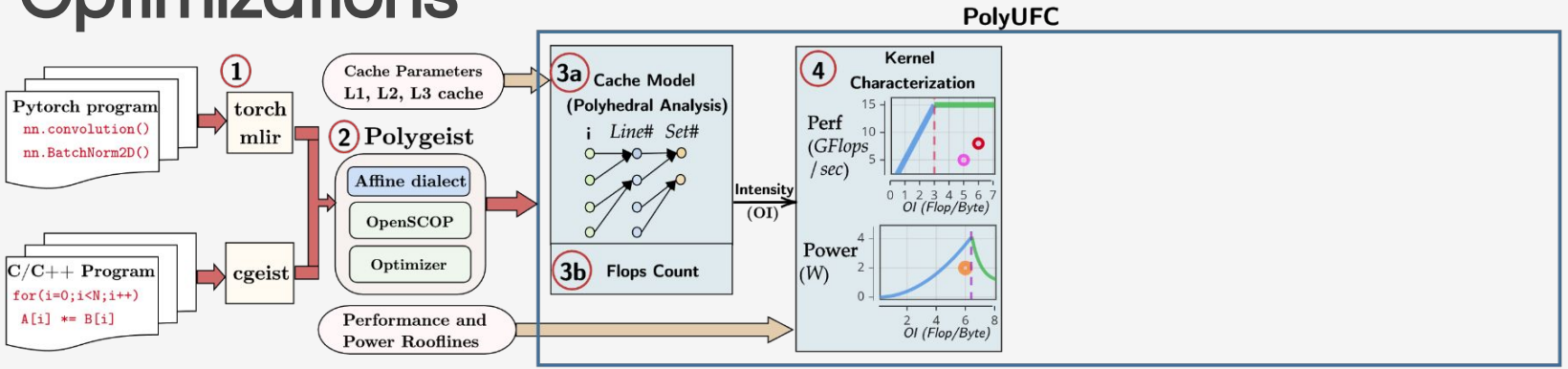
PolyUFC

Compiler Framework for Energy Optimizations



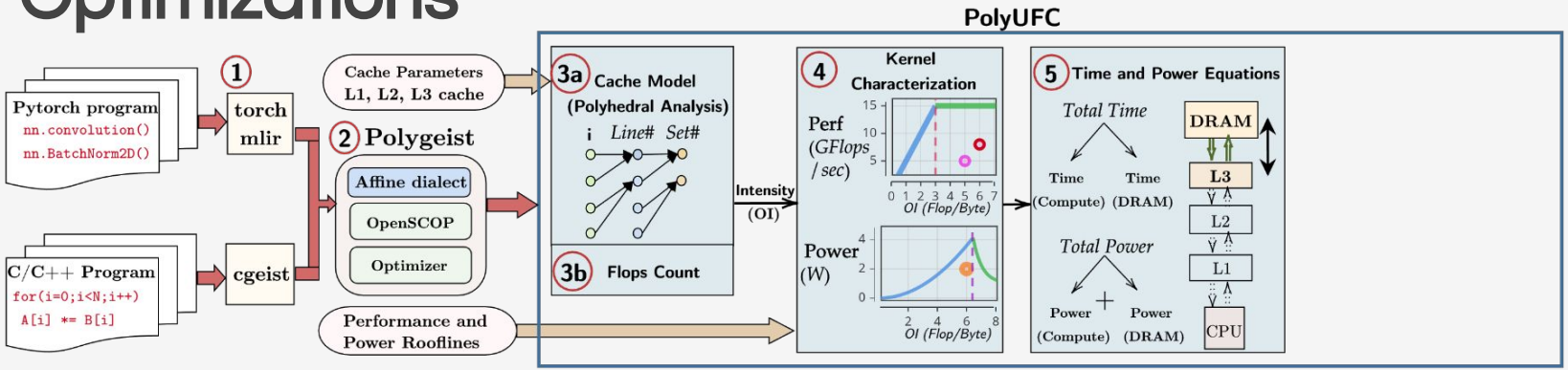
PolyUFC

Compiler Framework for Energy Optimizations



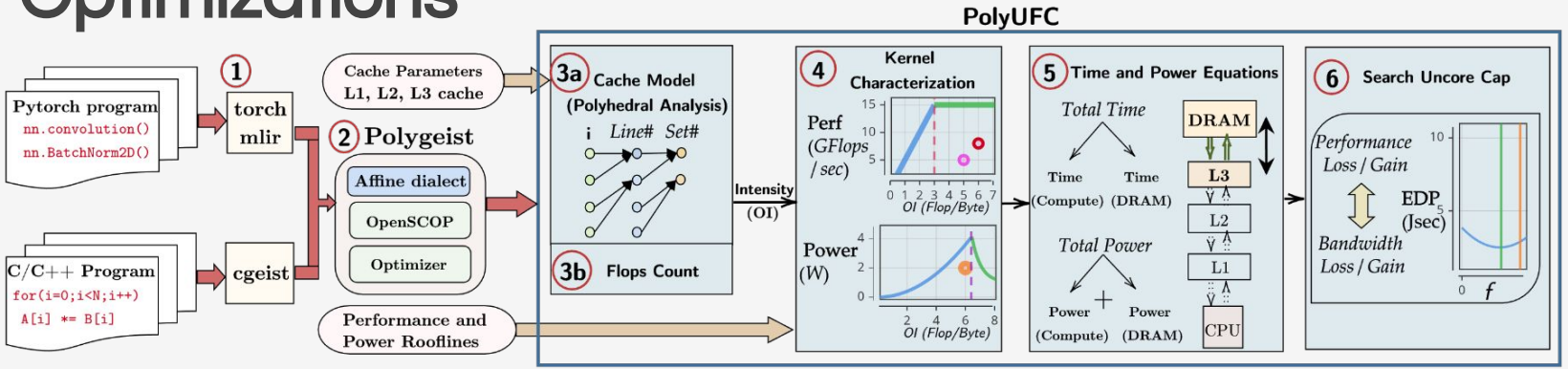
PolyUFC

Compiler Framework for Energy Optimizations



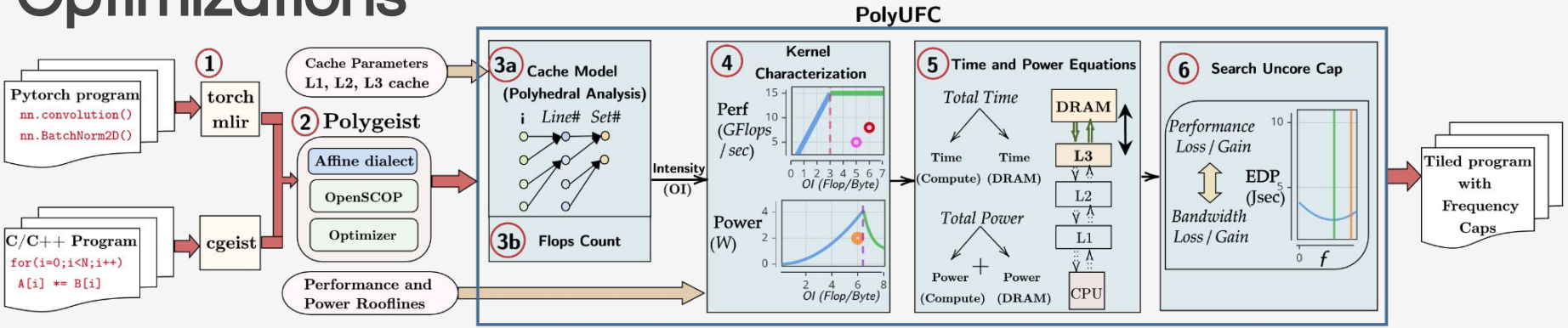
PolyUFC

Compiler Framework for Energy Optimizations



PolyUFC

Compiler Framework for Energy Optimizations



3 Experimental Evaluation

Experimental Setup

CPUs: Intel Broadwell (Xeon 1650-v4) and Raptorlake (i5 13600)

- **Core frequency driver:** Intel P-state scaling with performance governor
- **Uncore frequency driver:** Intel UFS scaling
  (Broadwell f_c Range: **1.2-2.8** GHz, Raptorlake Range: **0.8-4.6** GHz)
- **Optimizer:**
  - Pluto 0.11.4 with **`Polygeist`** (MLIR 18)
    Default tile sizes and fusion heuristic
- **Choice of Objectives**:
  - Performance
  - Energy
  - Energy Delay Product (EDP)
  - Performance and EDP
  - ...

Results: CB/BB Characterization (ML Kernels)

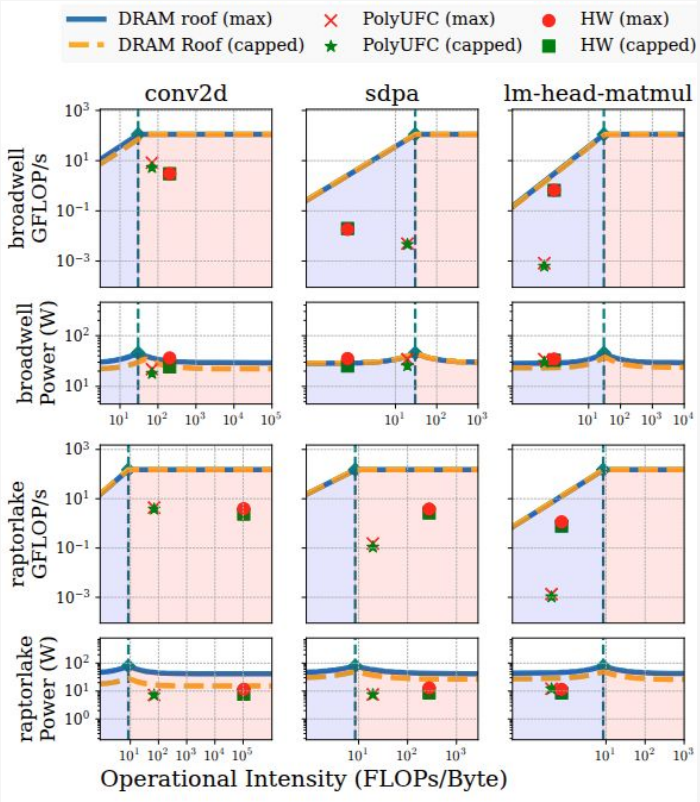
Benchmarks/Kernels

Vision ML Kernels

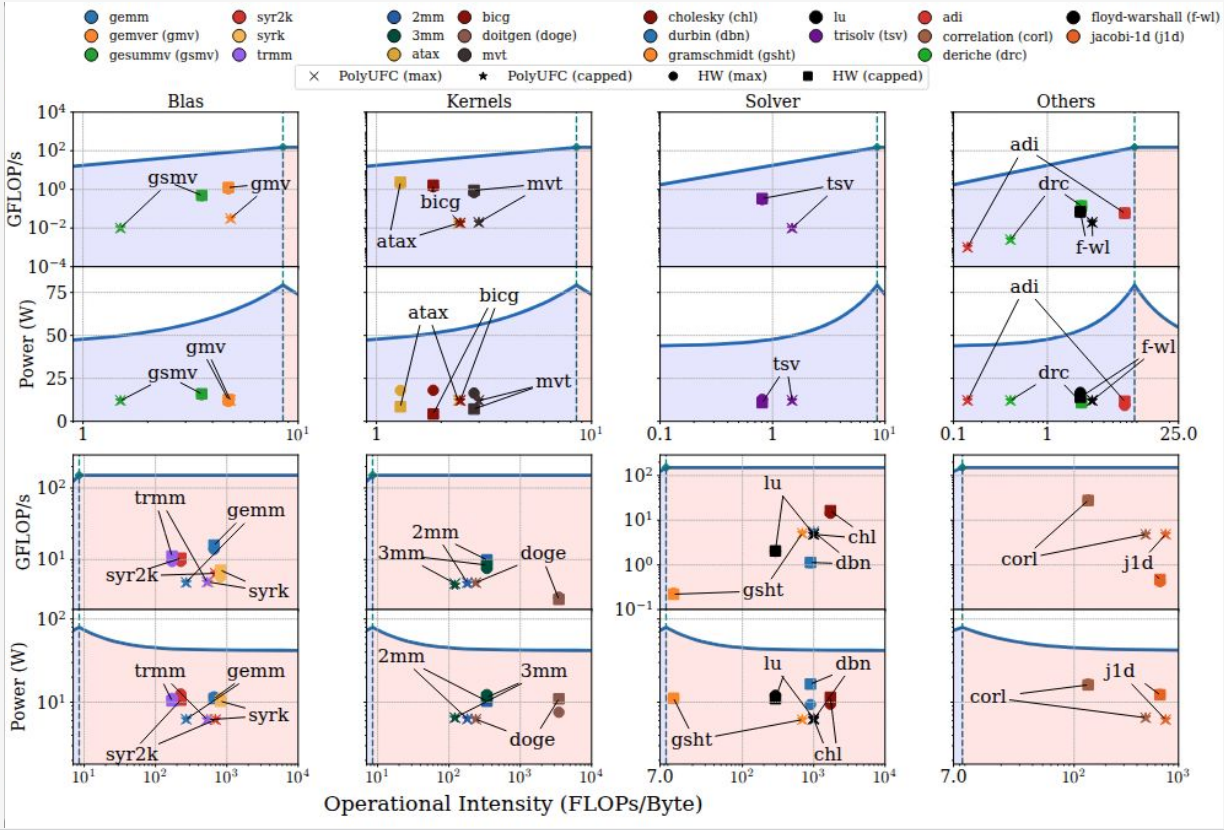
- **conv2d** (AlexNet, ConvNext, Resnet)

NLP Kernels

- **sdpa** (bert, gemma2)
- **lm-head** (gpt-2, llama2)

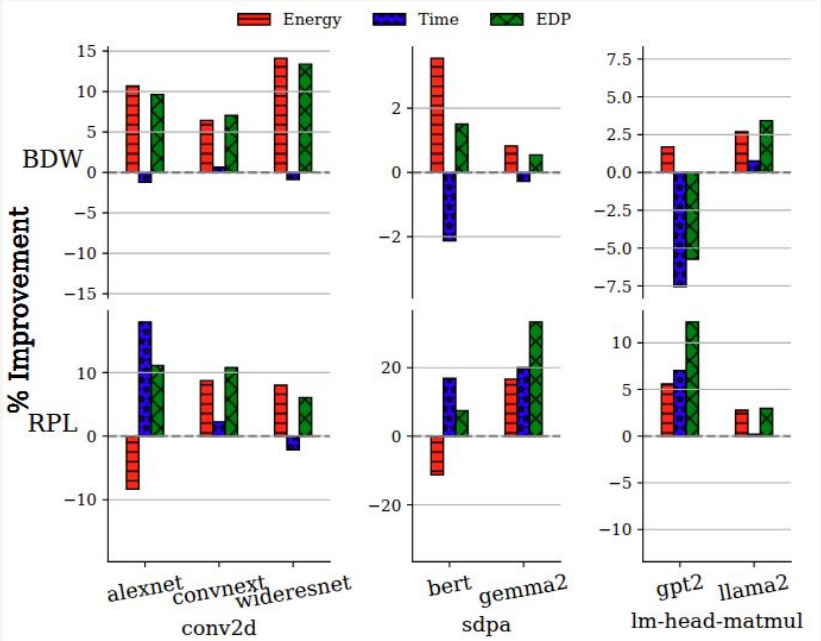


Results: CB/BB Characterization (PolyBench)

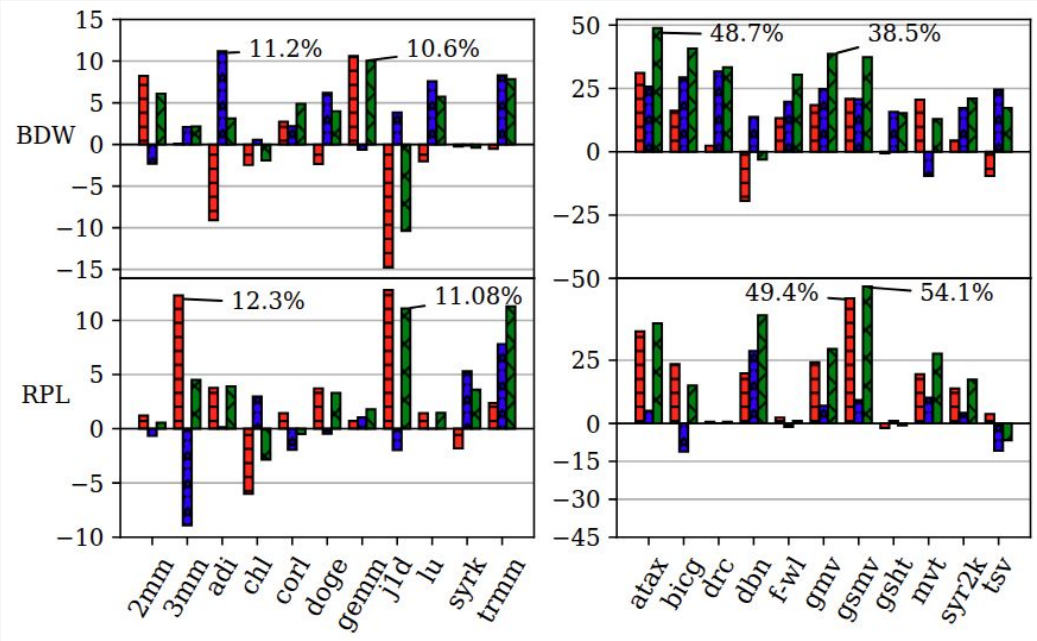


Results: EDP, Perf, Energy Trade-offs (ML kernels)

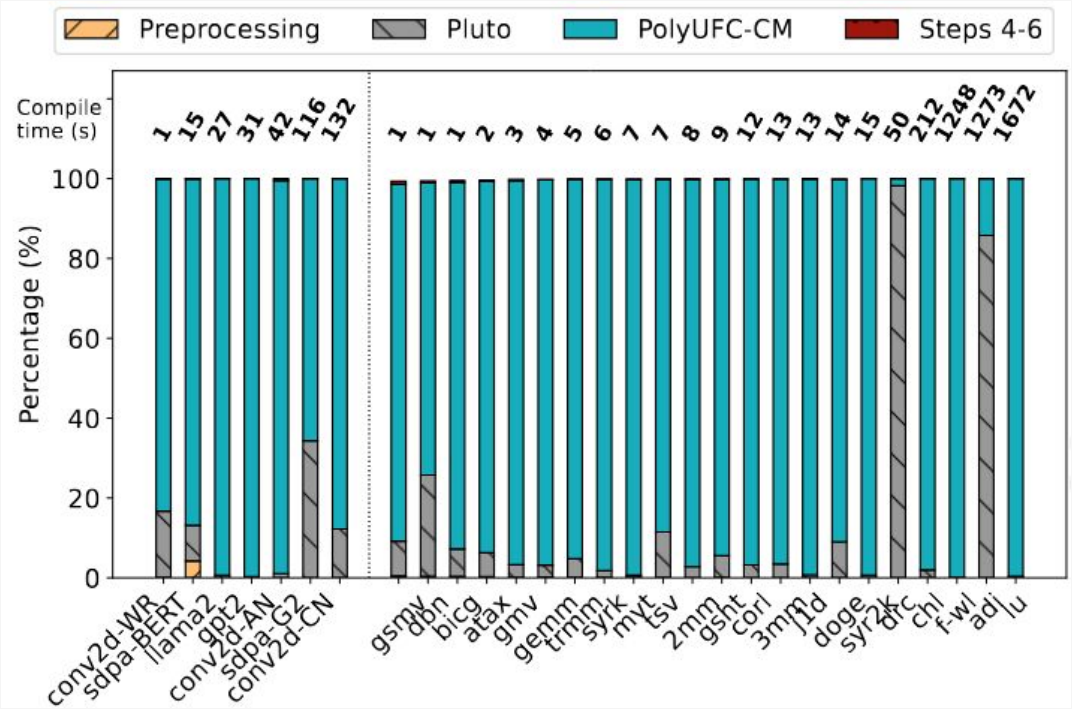
Current Goal:
Performance **+**
Energy Delay Product



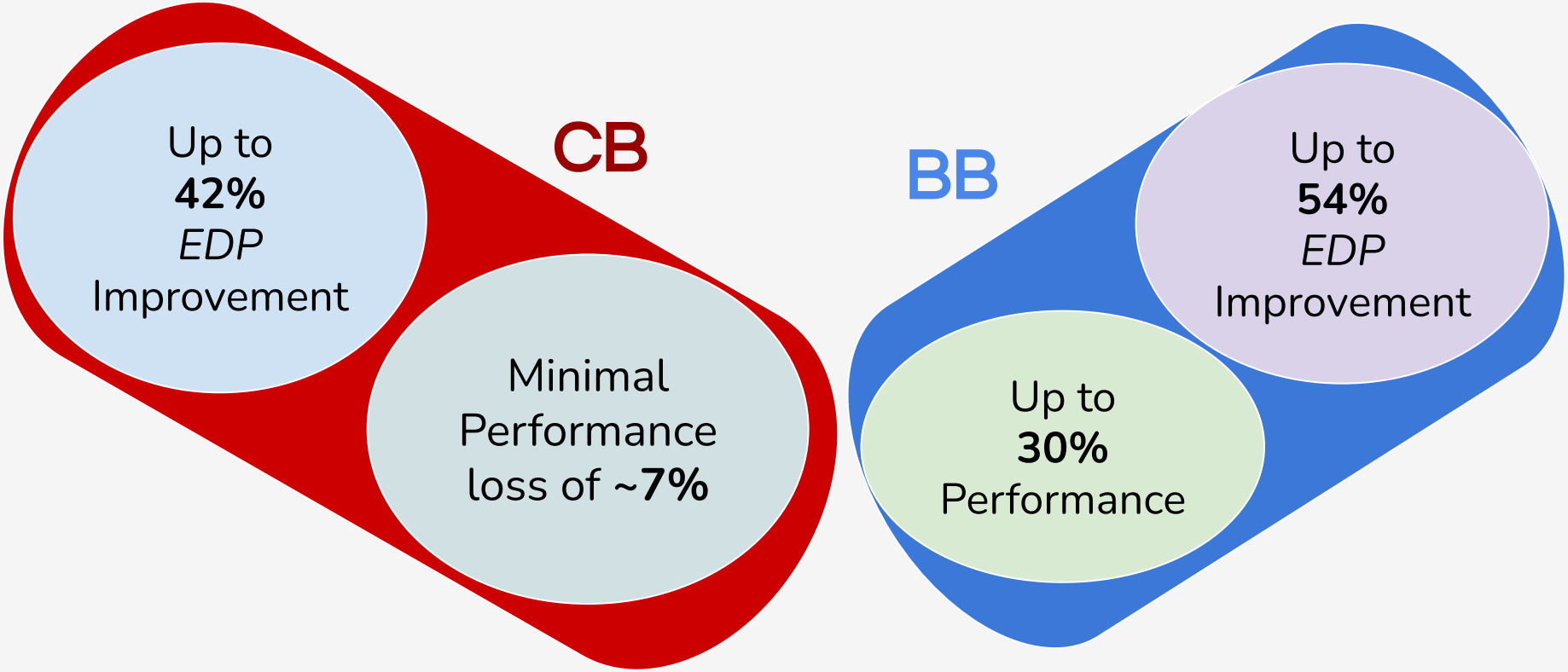
Results: EDP, Perf, Energy Trade-offs (PolyBench)



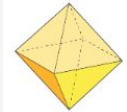
Results: Compile-time Analysis



Results: Performance and EDP



Conclusions

	**Automatic detection of** *over* and *under* provisioning in uncore domain
	**Polyhedral compilation based** characterization of ML codes (CB/BB)
LLC	**Problem-size independent** set-associative cache modeling
	**MLIR** based affine loop-nest level code generation
	**Detects Phase transitions and EDP Improvements** of up to **54%** on BB, **42%** on CB

PolyUFC

Thank You!

Questions?

https://compilers.cse.iith.ac.in/
https://compilers.cse.iith.ac.in/projects/polyufc
https://compilers.cse.iith.ac.in/projects/polyufc-supplementary