

Morpheus

Extending the Last Level Cache Capacity in GPU Systems
with Idle GPU Core Resources

Sina Darabi, Mohammad Sadrosadati, **Joël Lindegger**,
Negar Akbarzadeh, Mohammad Hosseini, Jisung Park,
Juan Gómez-Luna, Hamid Sarbazi-Azad, Onur Mutlu

ETH zürich



SAFARI
SAFARI Research Group

December 23rd 2022

بیژوهشگاه دانش‌های بنیادی (مرکز تحقیقات فیزیک نظری و ریاضیات)
IPM
INSTITUTE FOR RESEARCH IN FUNDAMENTAL SCIENCES



Motivation & Goal



Memory-bound GPU applications
leave **some cores under-utilized**



Our Goal:

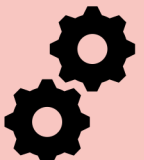
Leverage the **under-utilized cores**
to **boost** the **performance** and **energy efficiency**
of **memory-bound applications**

Our Work



Morpheus

First technique that leverages some **GPU cores' private memories** to **extend** the total GPU **last-level cache (LLC) capacity**

A black icon of two interlocking gears, representing hardware or mechanical components.

Morpheus employs **(1)** a **software helper kernel** in cores and **(2)** a **new hardware unit** in the LLC partitions

A black icon of a magnifying glass with a pulse line inside the lens, symbolizing analysis or search.

Morpheus significantly **improves performance** and **energy efficiency** of **memory-bound GPU applications** and **enables a 4x larger LLC** with the same LLC hardware

Outline

1 Introduction

2 Morpheus: Overview

3 Morpheus Controller

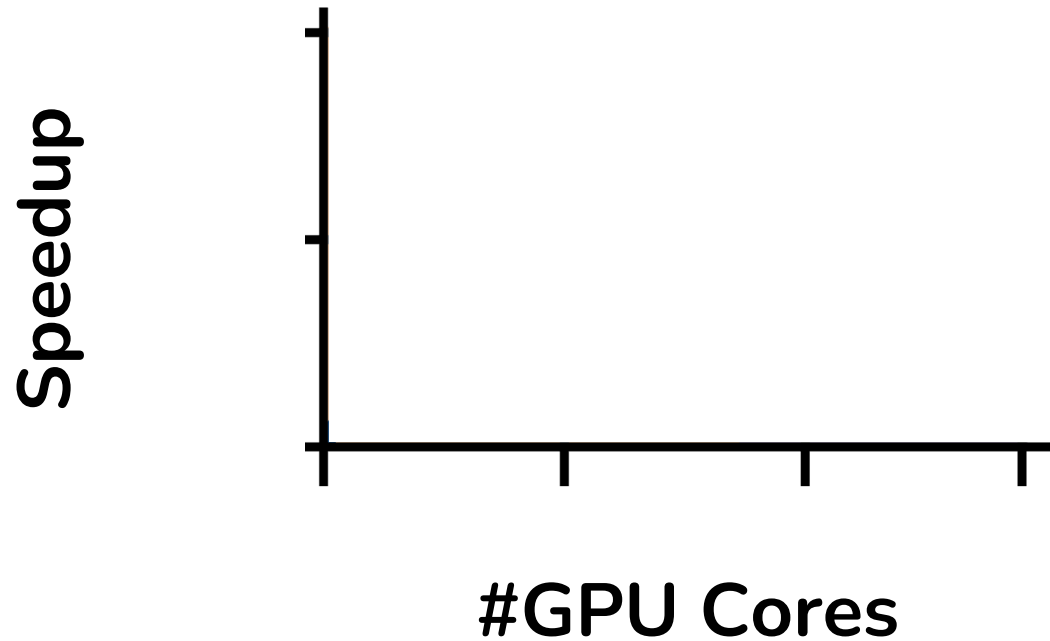
4 Extended LLC Kernel

5 Hit/Miss Prediction

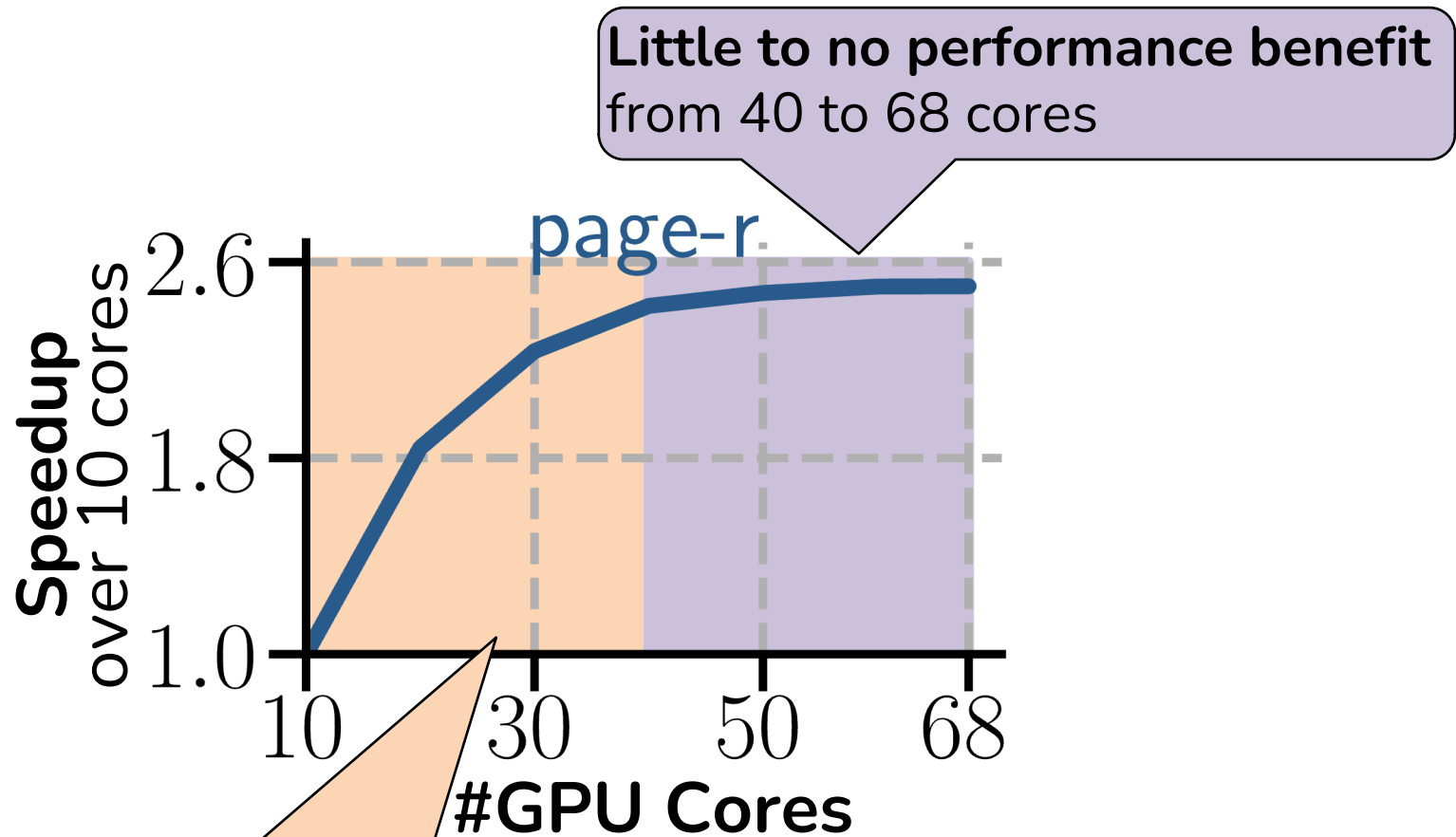
6 Evaluation

7 Conclusion

Memory-Bound GPU Applications



Memory-Bound GPU Applications

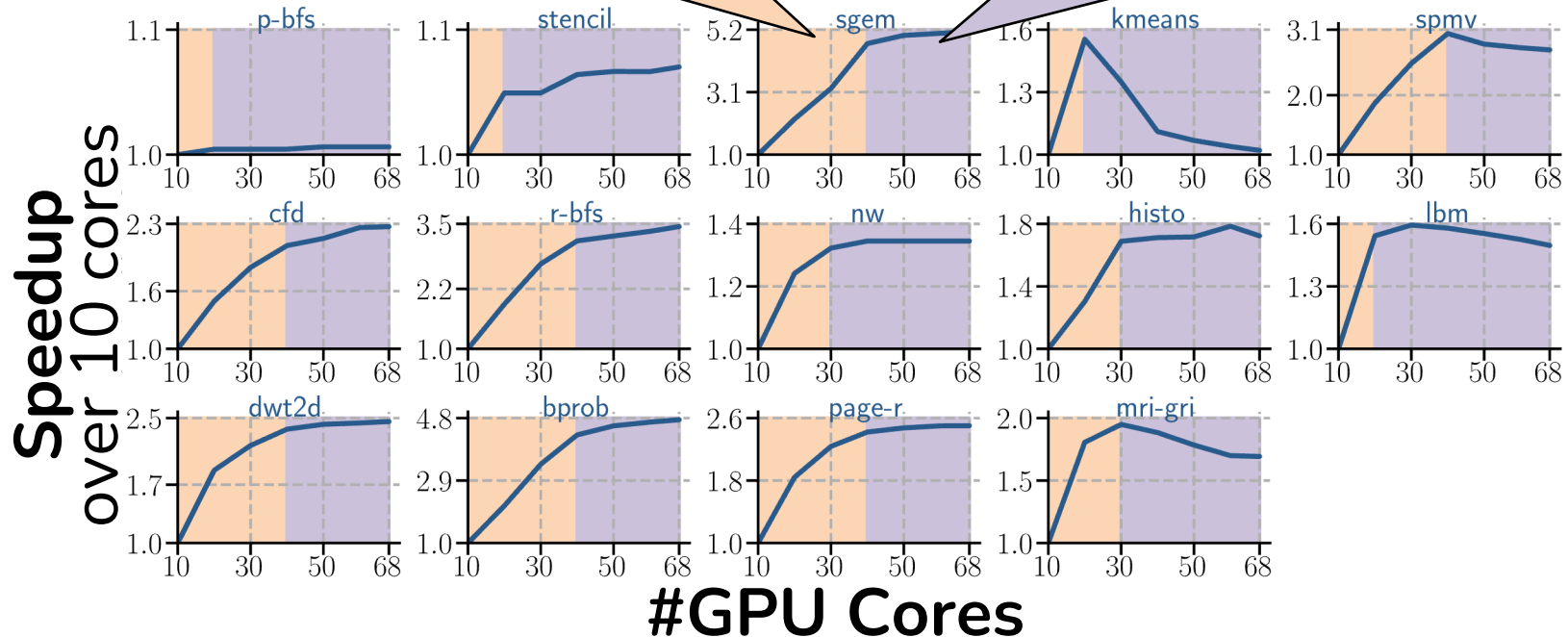


Good performance benefit up to 40 cores

Memory-Bound GPU Applications

Good performance benefit

Little to no performance benefit

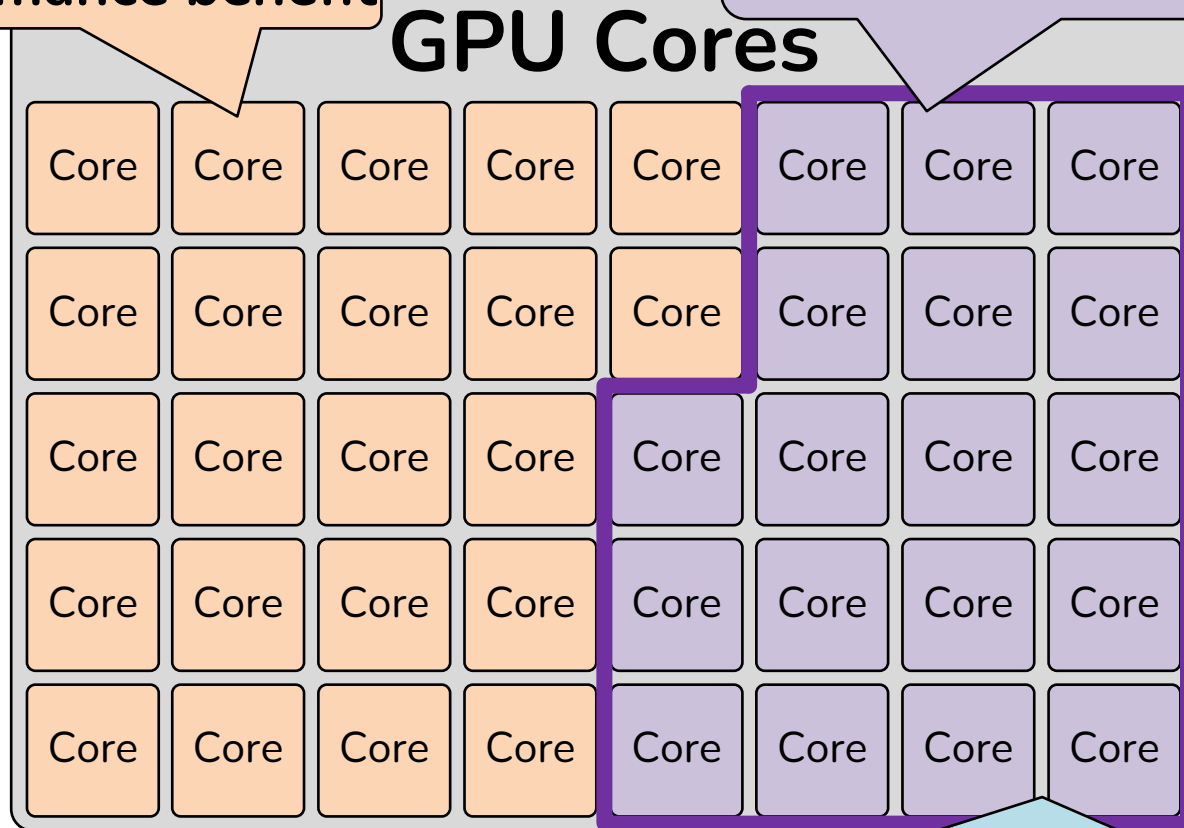


Conventional GPUs do **not benefit** (significantly) from all cores for **memory-bound** applications

Our Goal

Good performance benefit

Little to no performance benefit
for memory-bound applications



Our Goal

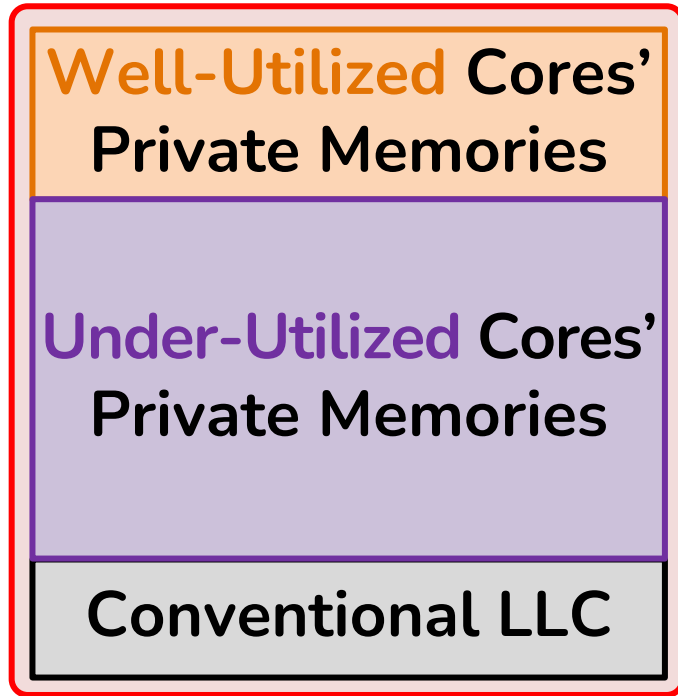
Leverage the **under-utilized cores**
to **boost** the **performance** and **energy efficiency**
of **memory-bound applications**

Outline

- 1 Introduction
- 2 Morpheus: Overview**
- 3 Morpheus Controller
- 4 Extended LLC Kernel
- 5 Hit/Miss Prediction
- 6 Evaluation
- 7 Conclusion

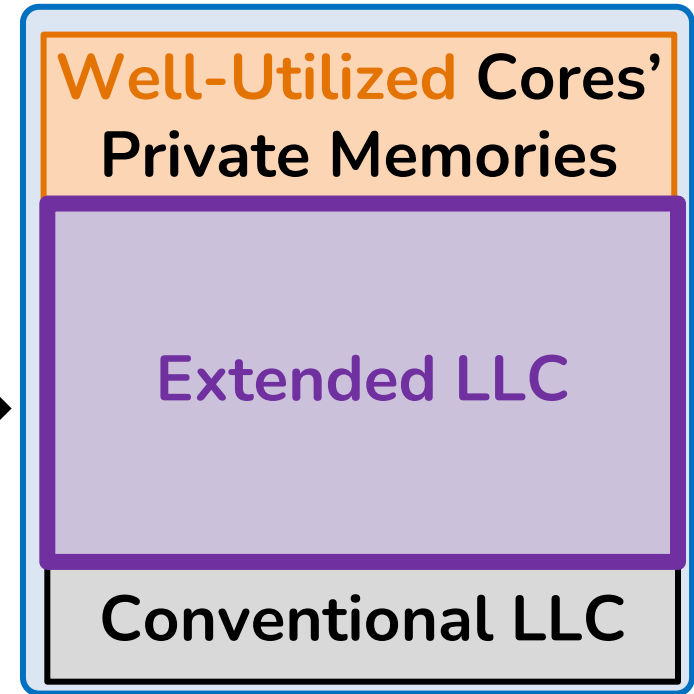
Key Idea

On-Chip Memory
of **Conventional GPU**



Morpheus
→

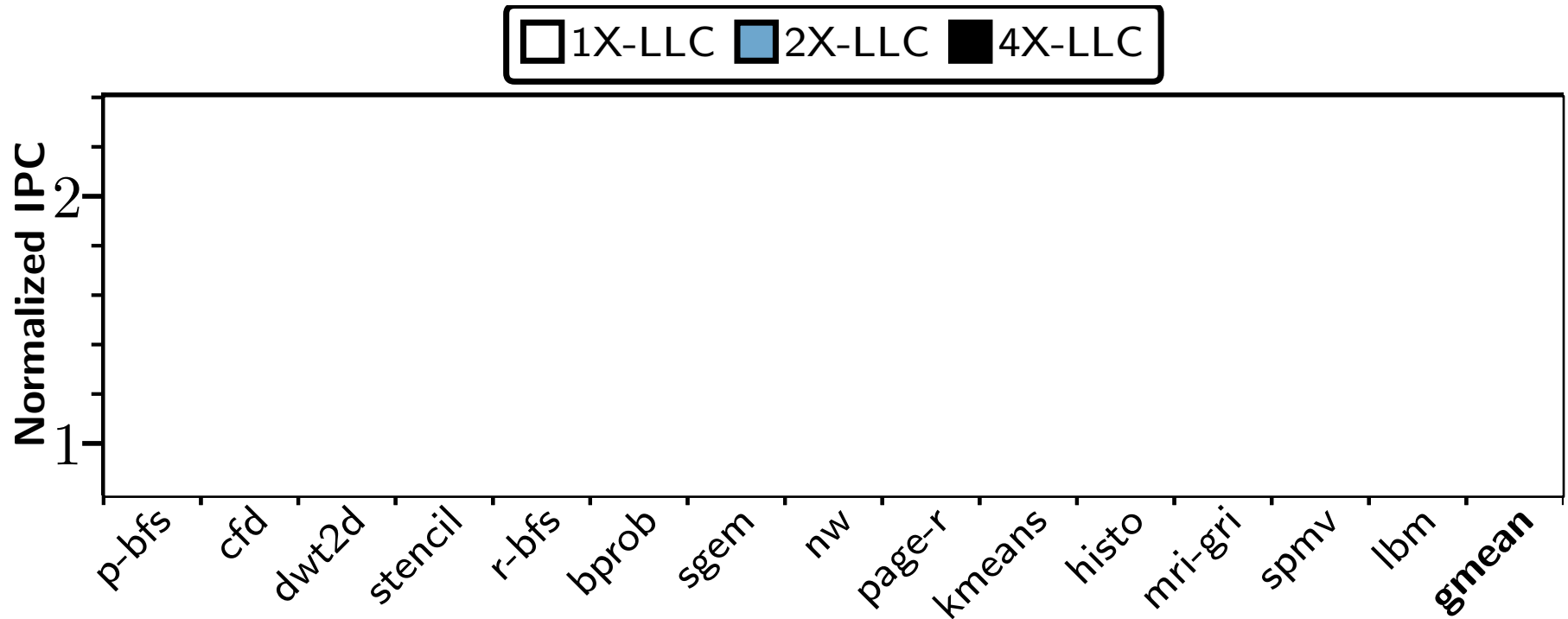
On-Chip Memory
of **Morpheus-enabled GPU**



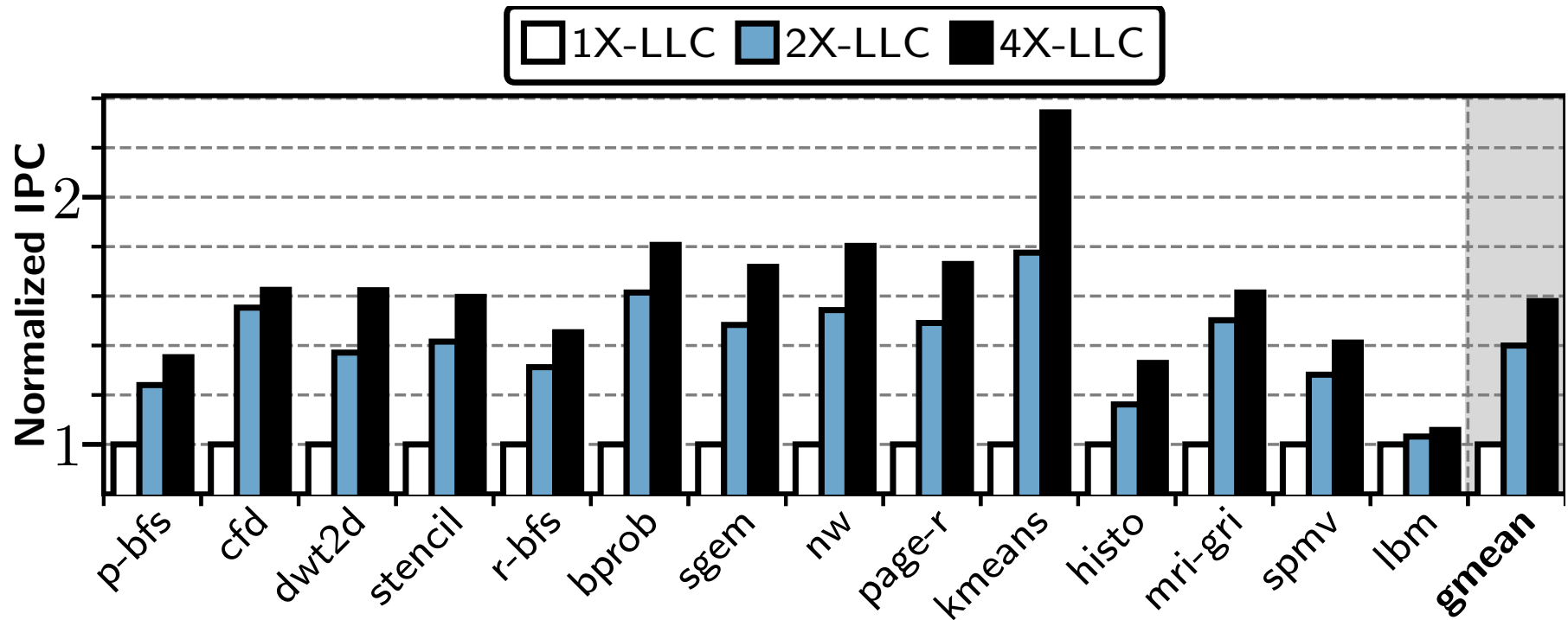
Morpheus' Key Idea:

Repurpose under-utilized GPU cores' **private memory**
to **extend** the **GPU's LLC capacity**

Performance Benefits of a Larger LLC



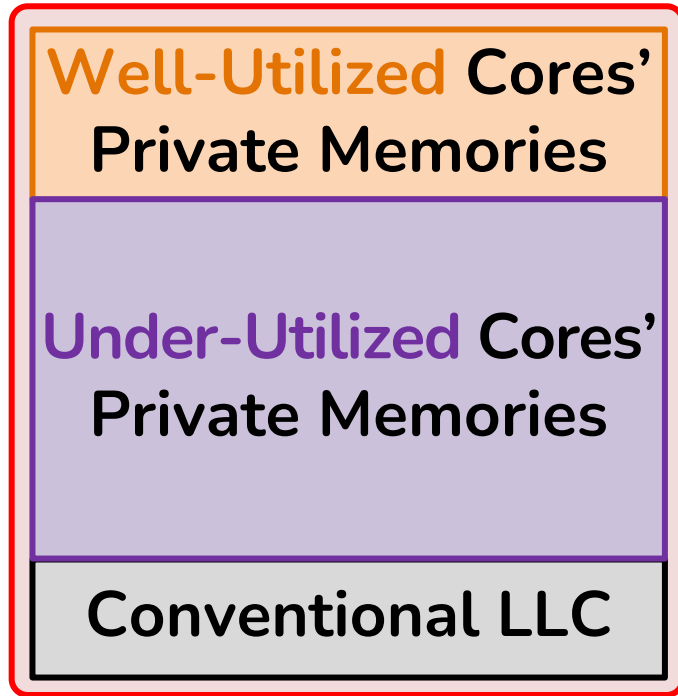
Performance Benefits of a Larger LLC



More LLC generally improves performance

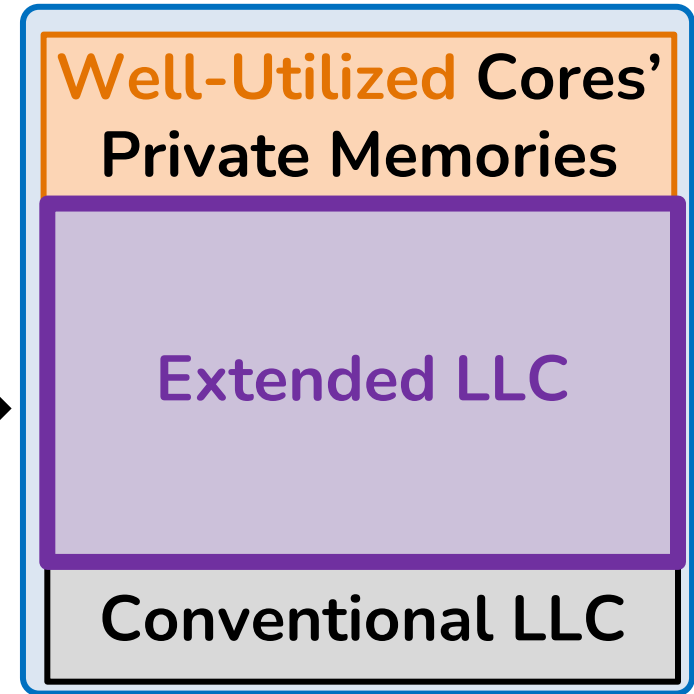
Key Idea

On-Chip Memory
of **Conventional GPU**



Morpheus
→

On-Chip Memory
of **Morpheus-enabled GPU**



Morpheus' Key Idea:

Repurpose under-utilized GPU cores' **private memory**
to **extend** the **GPU's LLC capacity**

GPU Core Execution Modes in Morpheus

GPU Core

Compute Mode

Operates Conventionally

Application Kernel

Application Data

Execution Units

L1 Cache

Shared Memory

Register File

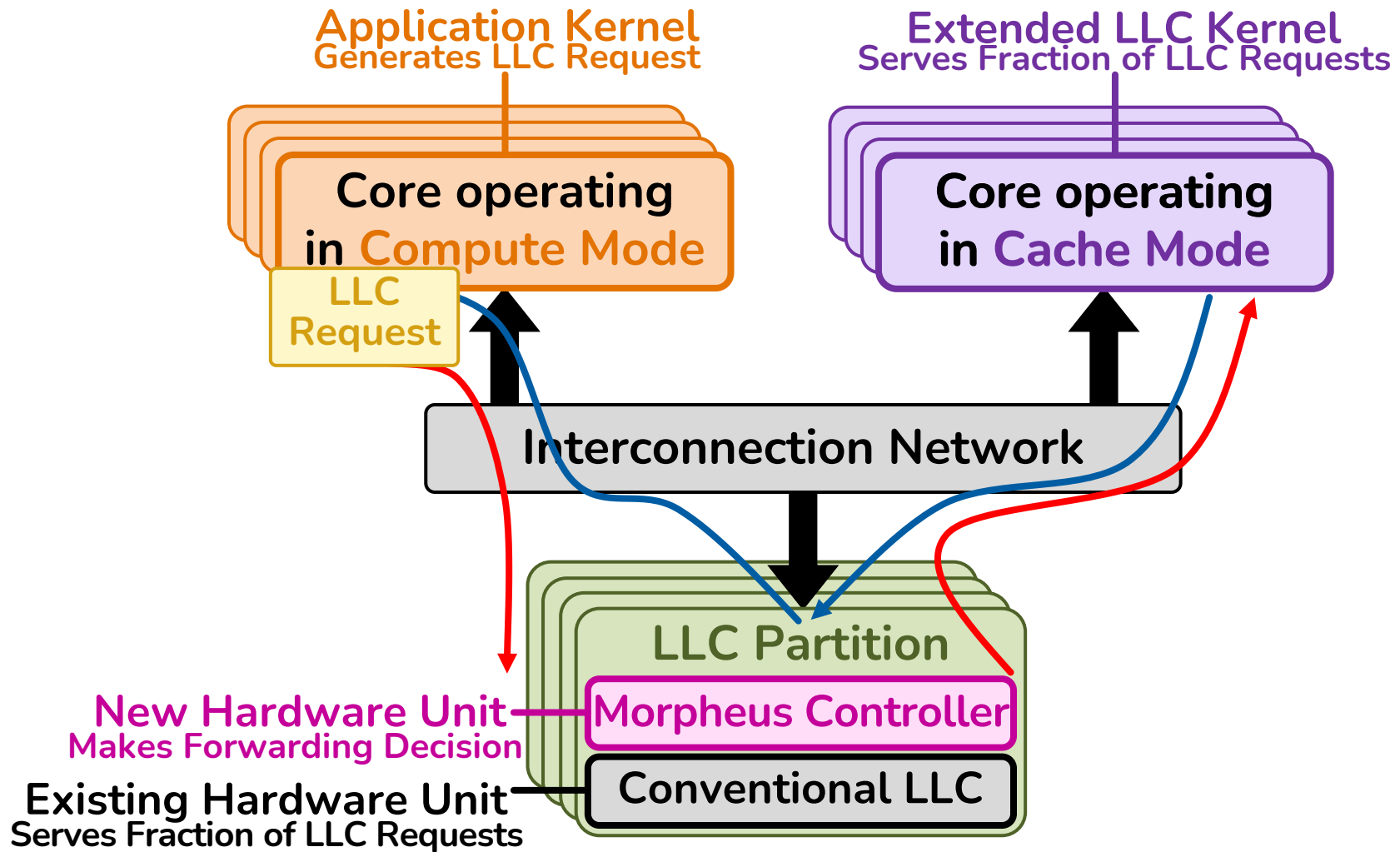
Cache Mode

Lends Private Memory
to Extend the LLC

Extended LLC Kernel

Controller State
Tag Array
Data Array

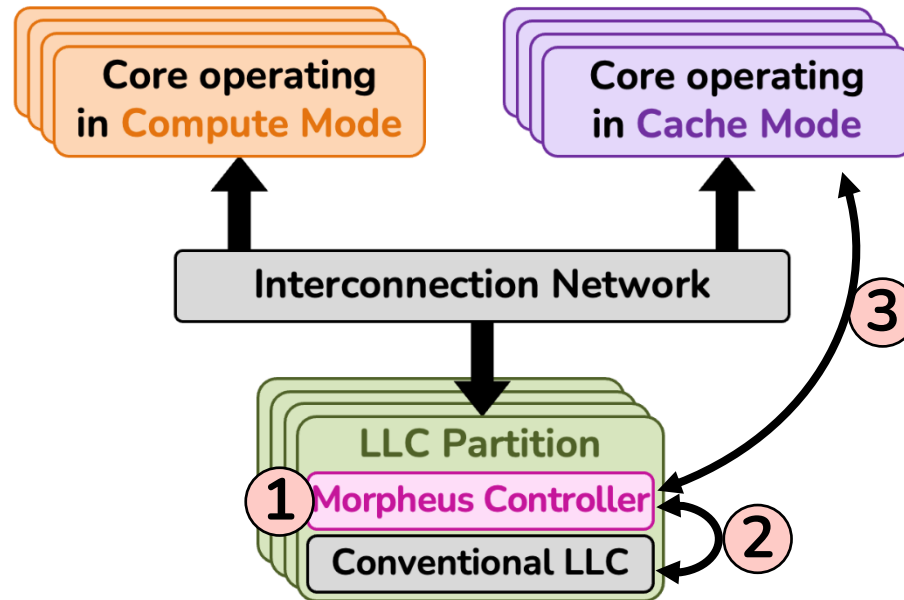
Serving an LLC Request in Morpheus



Outline

- 1 Introduction
- 2 Morpheus: Overview
- 3 Morpheus Controller**
- 4 Extended LLC Kernel
- 5 Hit/Miss Prediction
- 6 Evaluation
- 7 Conclusion

Morpheus Controller



1

Forwarding Decision

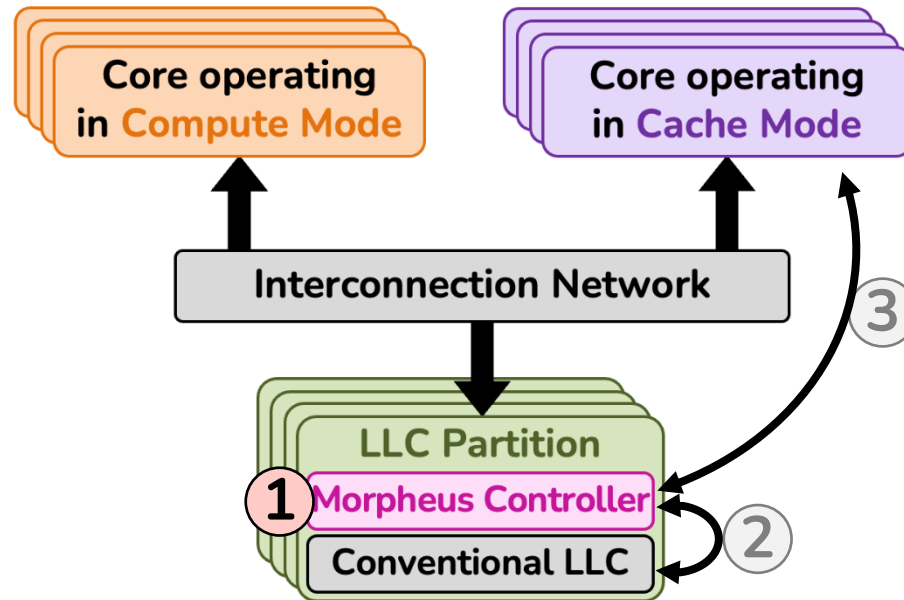
2

Query Conventional LLC

3

Query Extended LLC

Morpheus Controller



1

Forwarding Decision

Address Separation, Combinational Logic

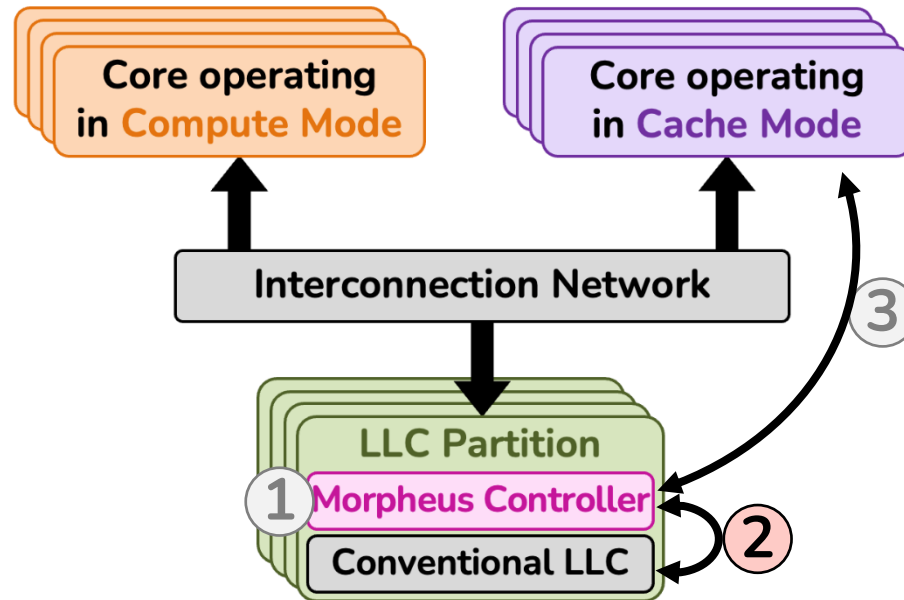
2

Query Conventional LLC

3

Query Extended LLC

Morpheus Controller



1

Forwarding Decision

Address Separation, Combinational Logic

2

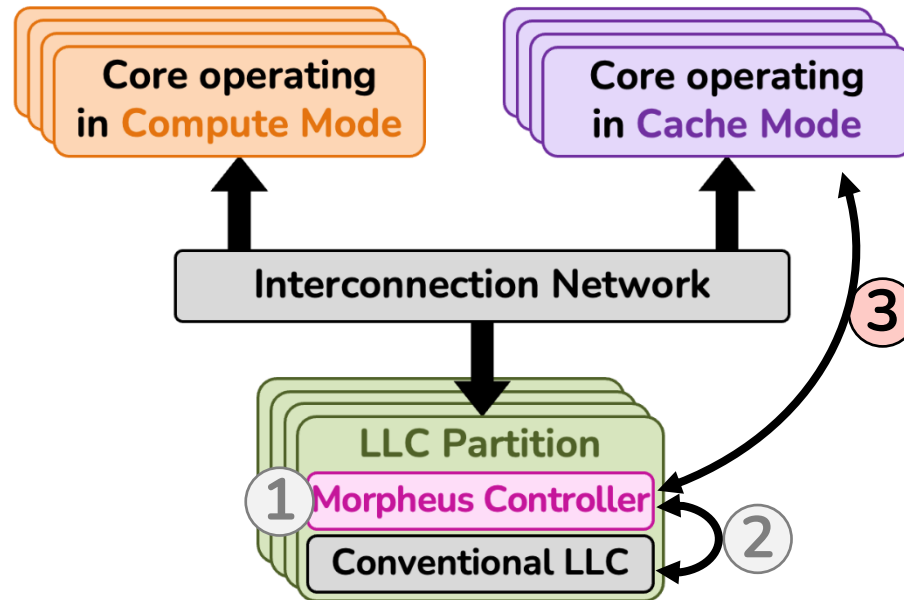
Query Conventional LLC

Physically Co-Located, Simple Forwarding

3

Query Extended LLC

Morpheus Controller



1

Forwarding Decision

Address Separation, Combinational Logic

2

Query Conventional LLC

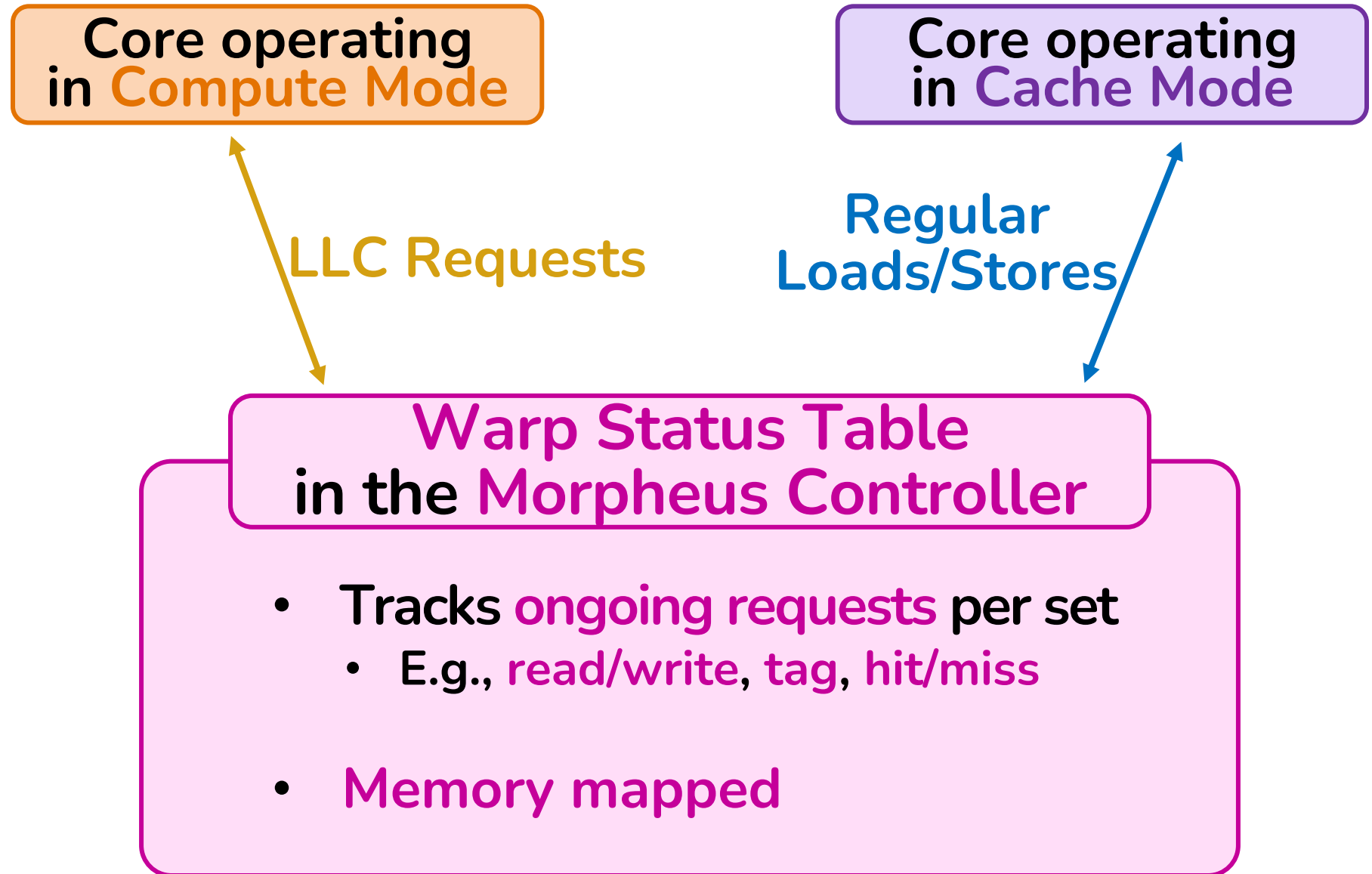
Physically Co-Located, Simple Forwarding

3

Query Extended LLC

Warp Status Table

Morpheus Controller Warp Status Table

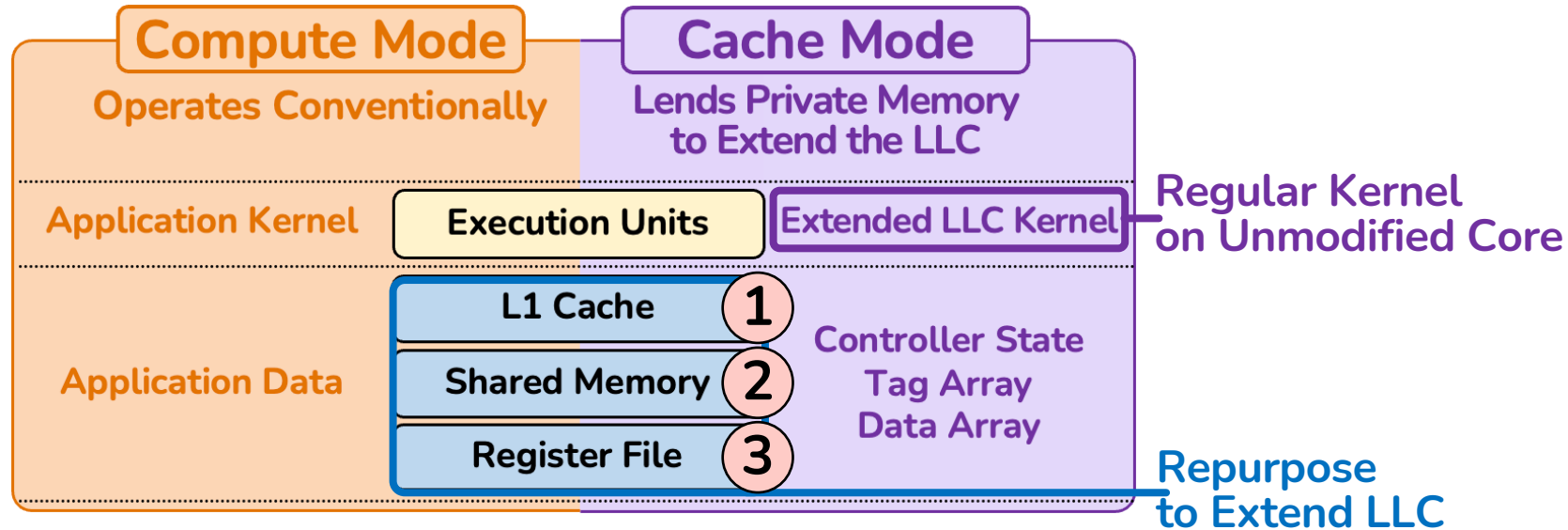


Outline

- 1 Introduction
- 2 Morpheus: Overview
- 3 Morpheus Controller
- 4 Extended LLC Kernel**
- 5 Hit/Miss Prediction
- 6 Evaluation
- 7 Conclusion

Extended LLC Kernel

GPU Core



1

Query L1 Cache

Load/Store Instructions

2

Query Shared Memory

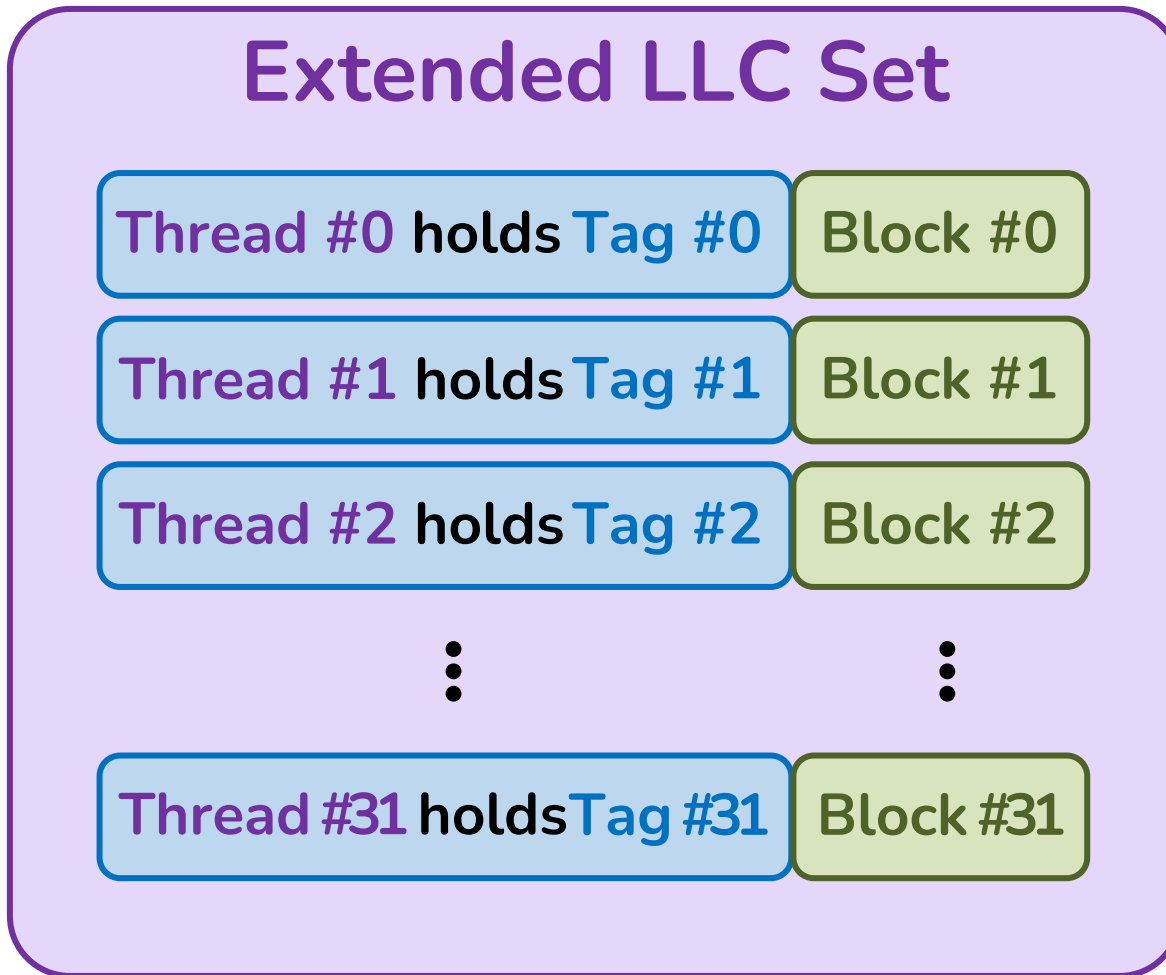
Software Engineering

3

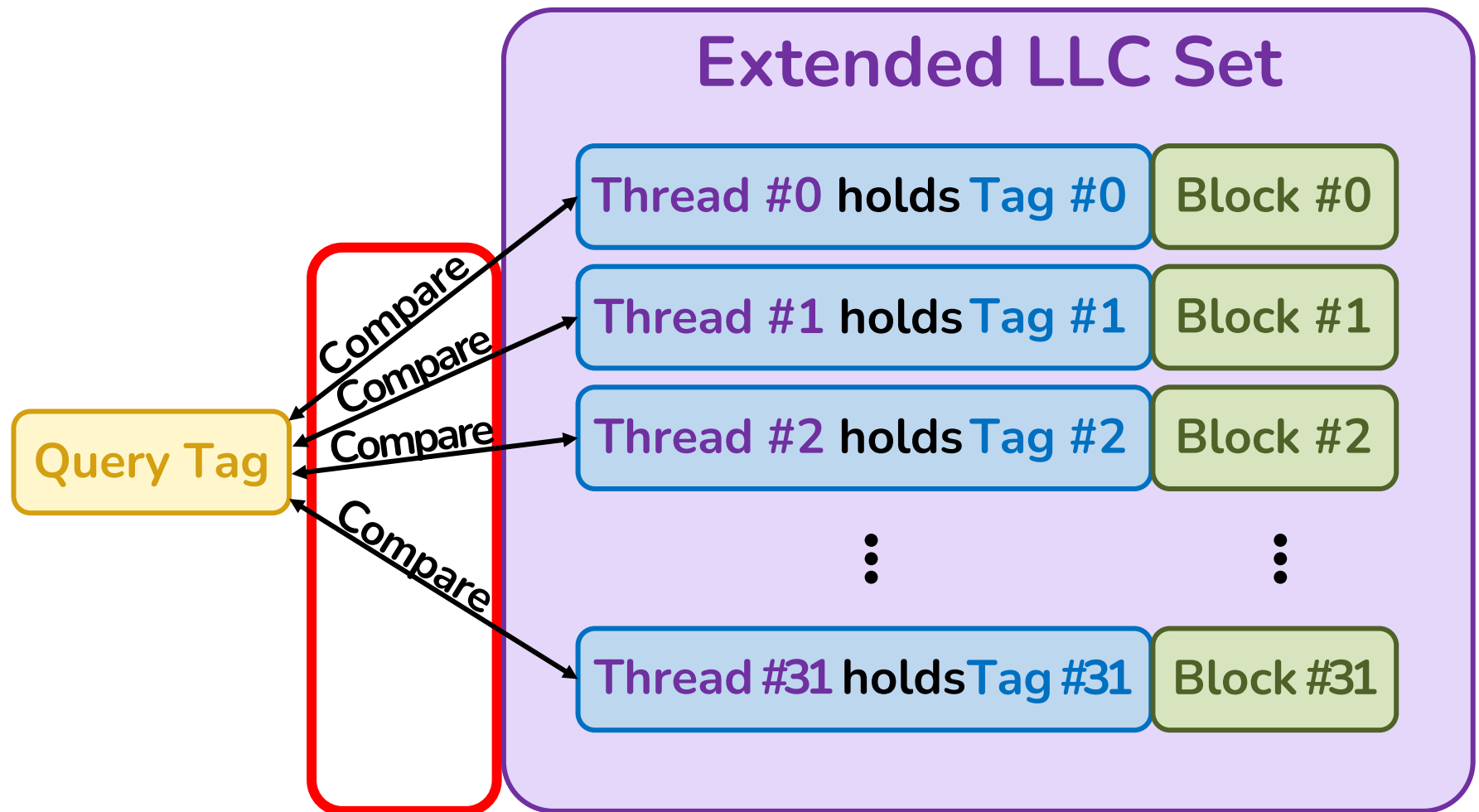
Query Register File

Data Layout across Threads for Parallelism

Extended LLC via Register File

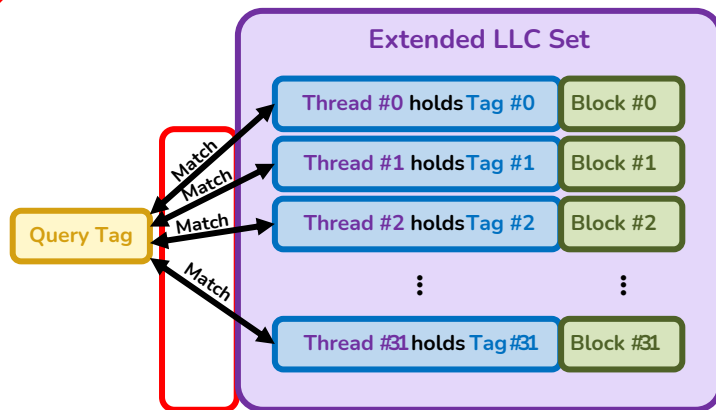


Extended LLC via Register File

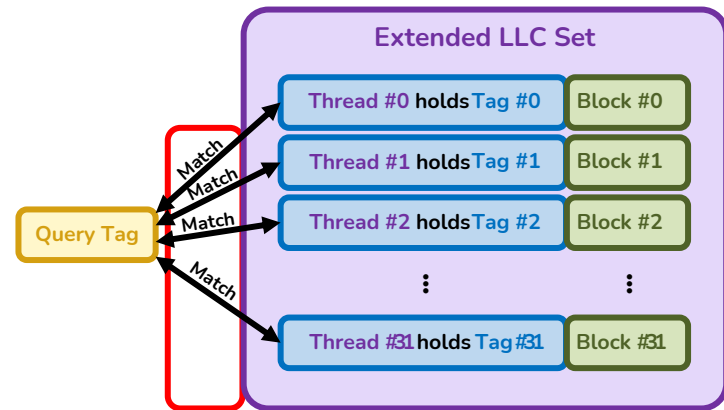


32 Threads
Operate in Parallel

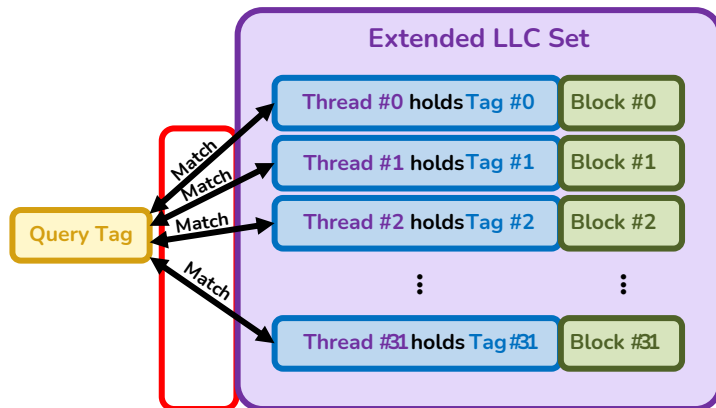
Extended LLC via Register File



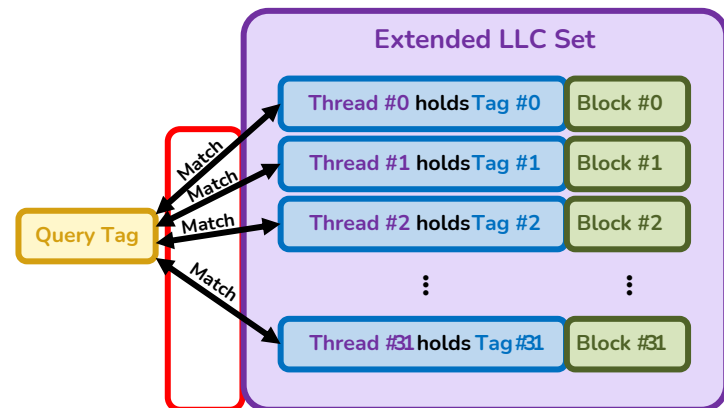
32 Threads
Operate in Parallel



32 Threads
Operate in Parallel



32 Threads
Operate in Parallel



32 Threads
Operate in Parallel

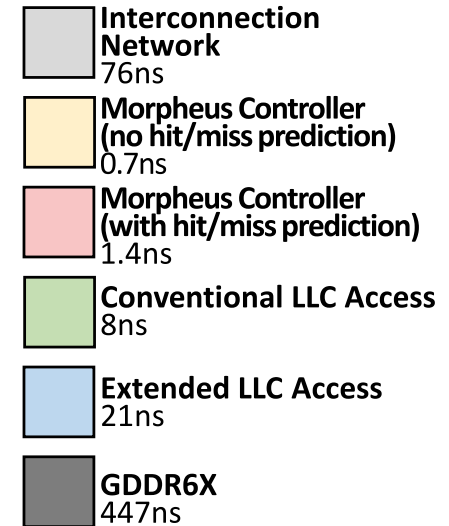
Many Warps (e.g., 48)
Operate in Parallel

Outline

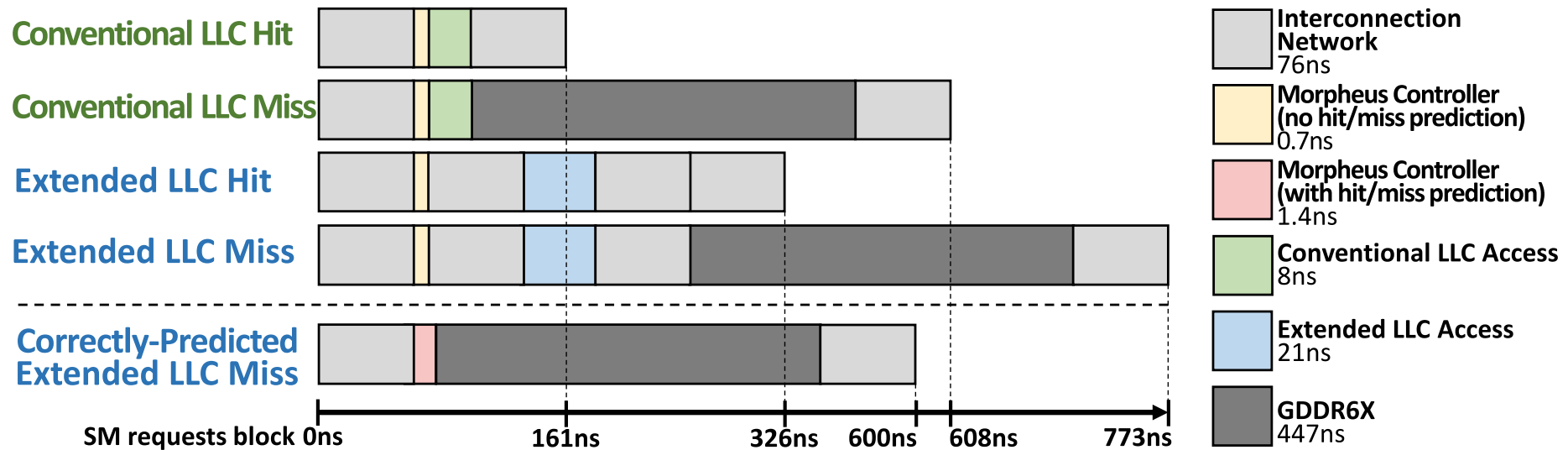
- 1 Introduction
- 2 Morpheus: Overview
- 3 Morpheus Controller
- 4 Extended LLC Kernel
- 5 Hit/Miss Prediction**
- 6 Evaluation
- 7 Conclusion

LLC Timelines

Conventional LLC Hit



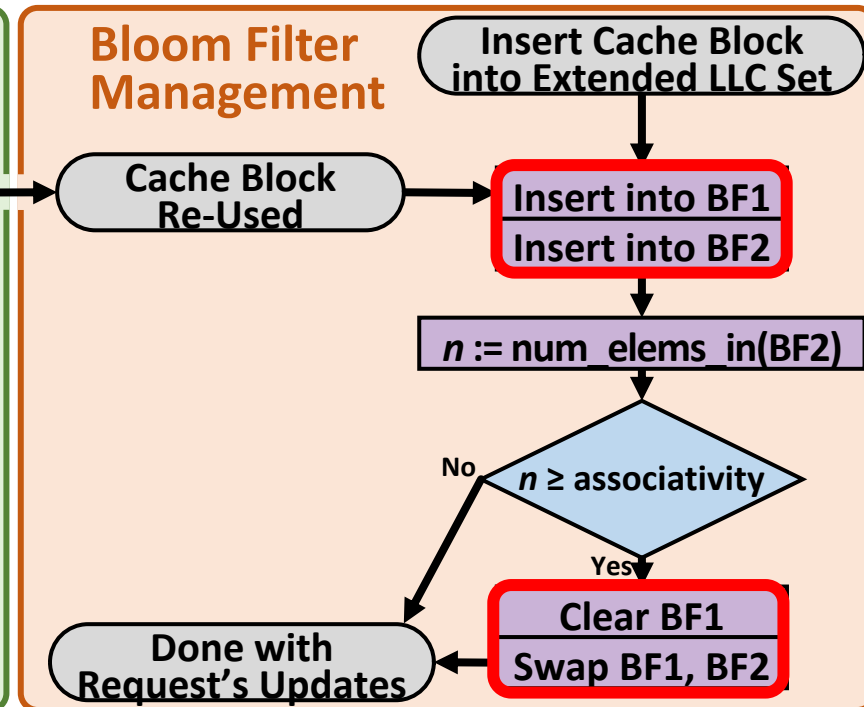
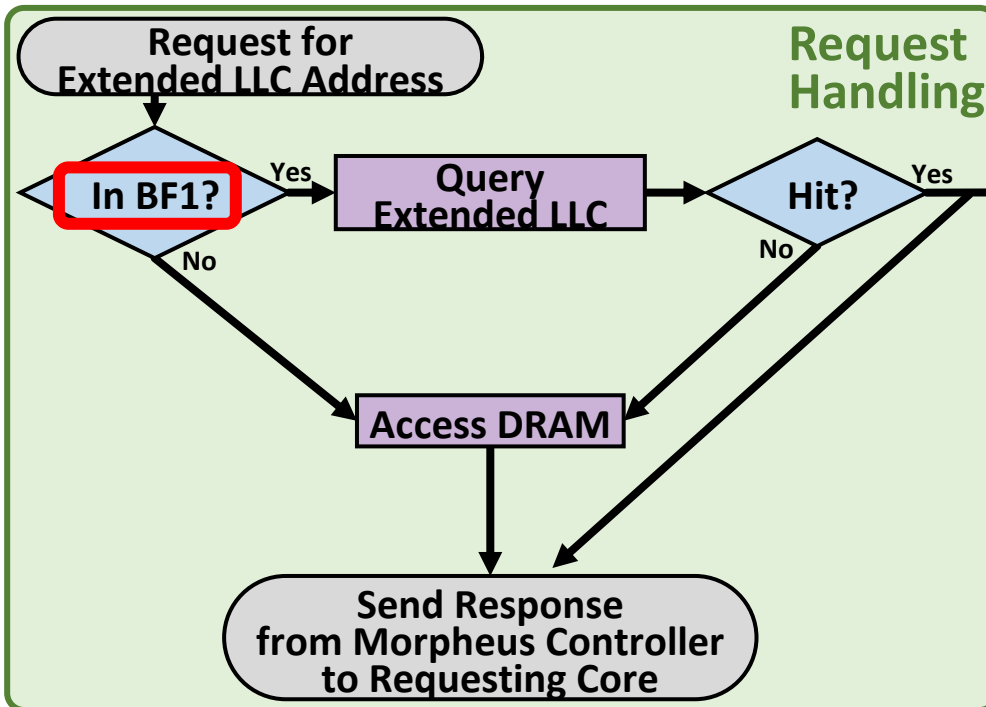
LLC Timelines



Misses in the **extended LLC** are **particularly expensive**

Correctly **predicting** them can **reduce** the **miss latency** significantly

Hit/Miss Predictor



Is There A More Sophisticated Hit/Miss Predictor?





Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera¹ Konstantinos Kanellopoulos¹ Shankar Balachandran² David Novo³
Ataberk Olgun¹ Mohammad Sadrosadati¹ Onur Mutlu¹

¹ETH Zürich ²Intel Processor Architecture Research Lab ³LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests

off-chip main memory (i.e., an off-chip load) often stalls the processor core by blocking the instruction retirement from the re-order buffer (ROB), thus limiting the core's performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art

<https://arxiv.org/pdf/2209.00188.pdf>

Bera+, “Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction”, MICRO ‘22

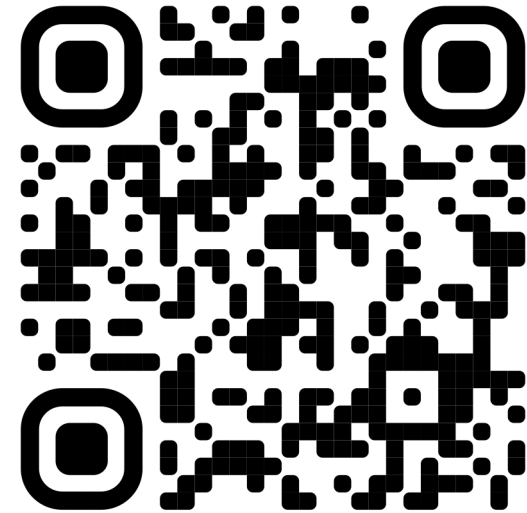
More in the Morpheus Paper: Mechanisms

- **Detailed mechanisms**

- Morpheus Controller
- Extended LLC kernel

- **Optimization techniques**

- Indirect-MOV
- Cache compression applied to Morpheus



Morpheus on arXiv

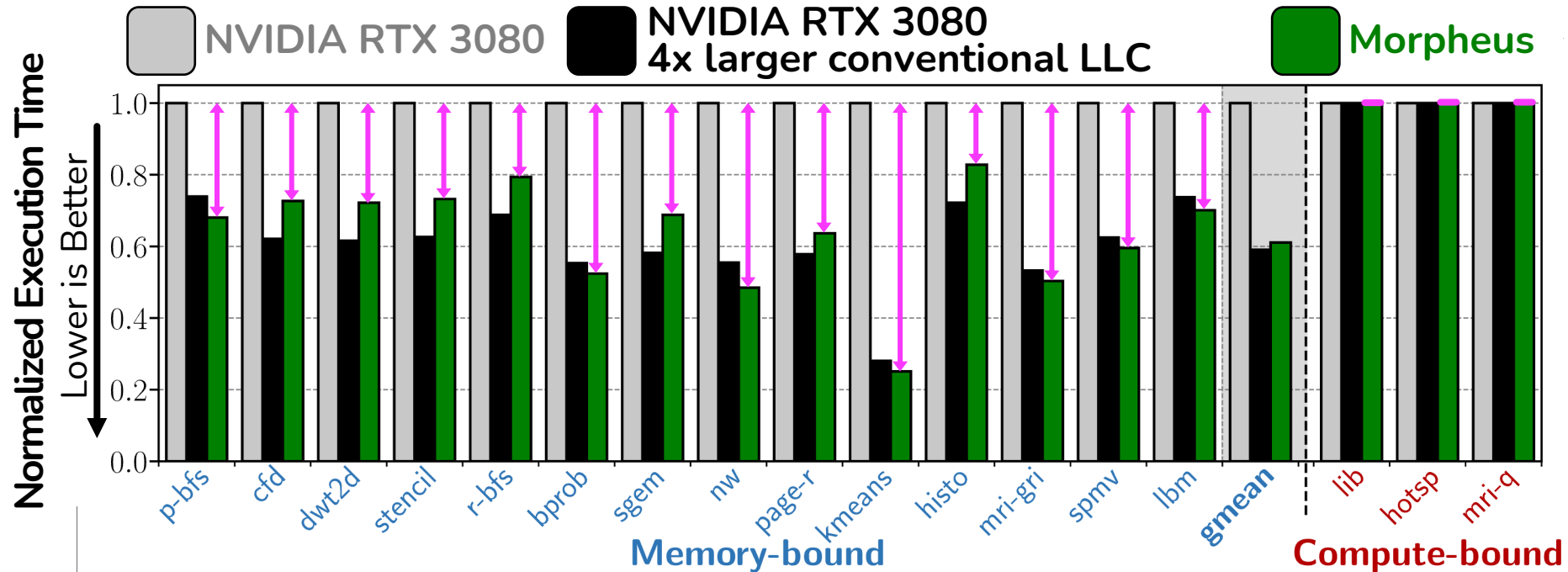
Outline

- 1 Introduction
- 2 Morpheus: Overview
- 3 Morpheus Controller
- 4 Extended LLC Kernel
- 5 Hit/Miss Prediction
- 6 Evaluation**
- 7 Conclusion

Methodology

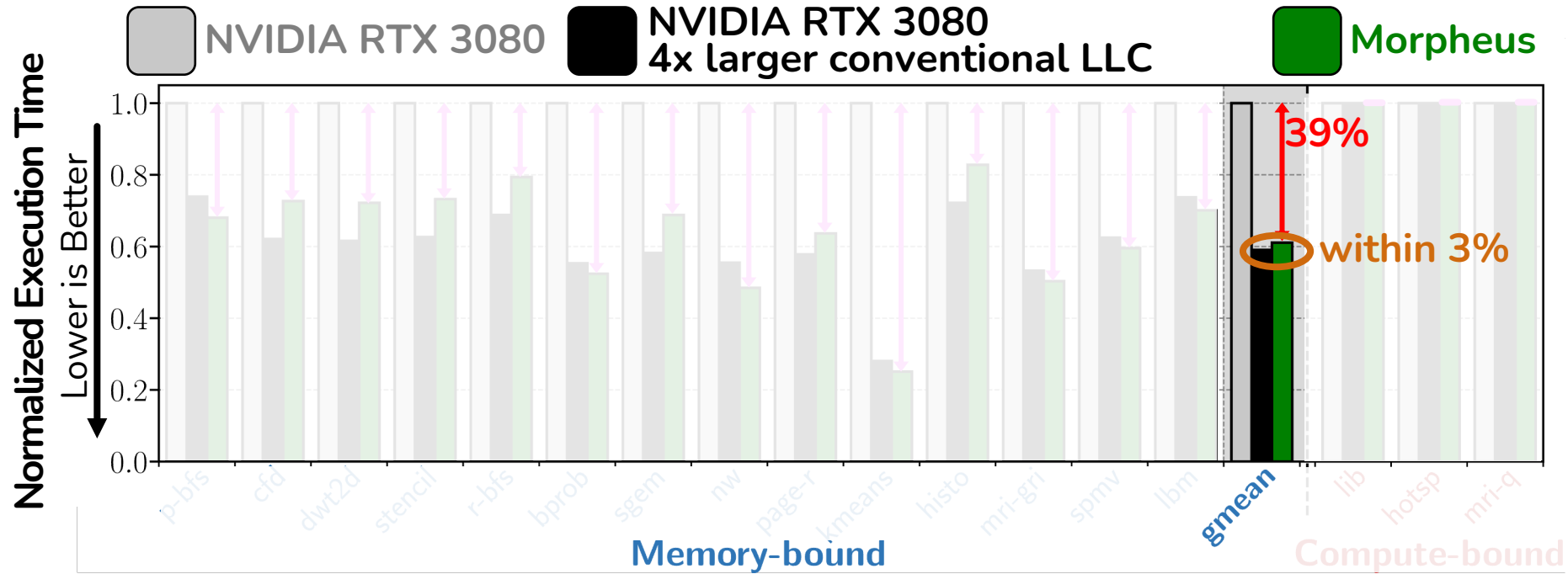
- **17 representative applications**
 - 14 memory-bound
 - 3 compute-bound
- **Experimentally characterize the *extended LLC***
 - On a real NVIDIA RTX 3080, as the *extended LLC* kernel requires no special hardware
 - Capacity, latency, bandwidth, energy/byte of the *extended LLC*
- **Simulate a Morpheus-enabled GPU using AccelSim** [Khairy+ ISCA'20]
 - Performance, energy efficiency, bandwidth utilization

Morpheus Performance



Morpheus performs **better** than or **equal** to the baseline for **all** applications

Morpheus Performance

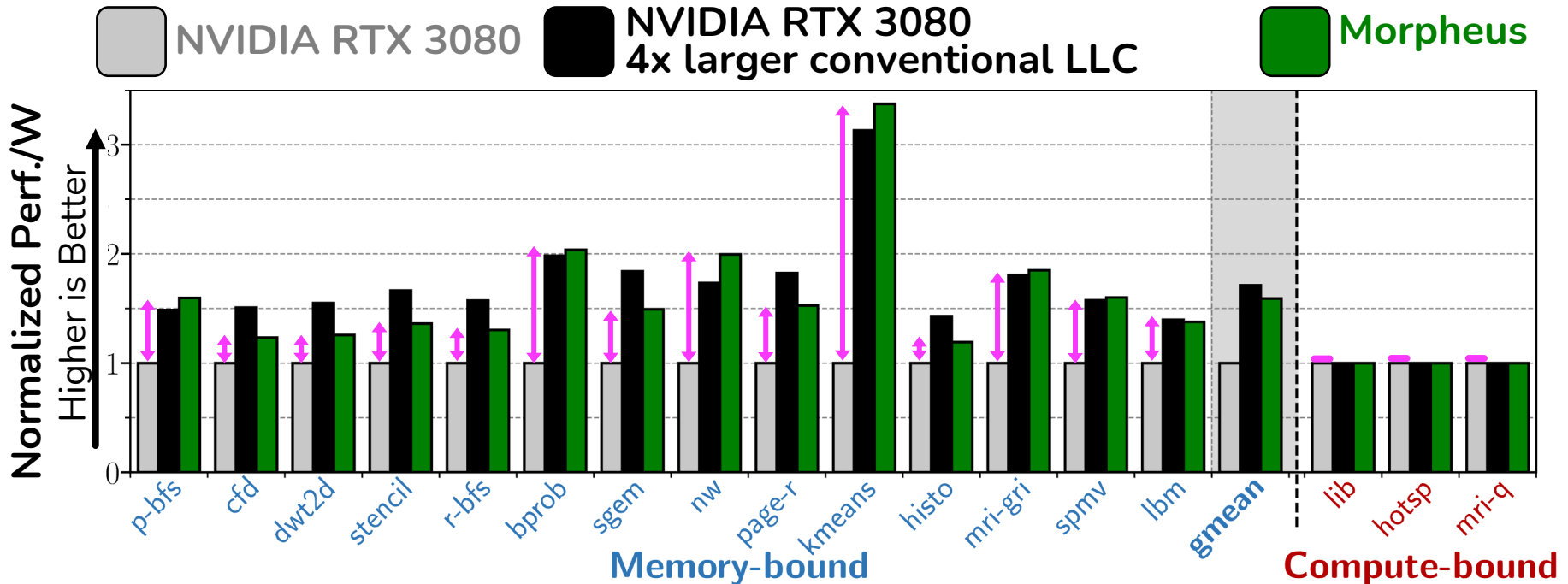


Morpheus performs **better** than or **equal** to the baseline for **all** applications

Morpheus **accelerates memory-bound** applications by **39%** on average

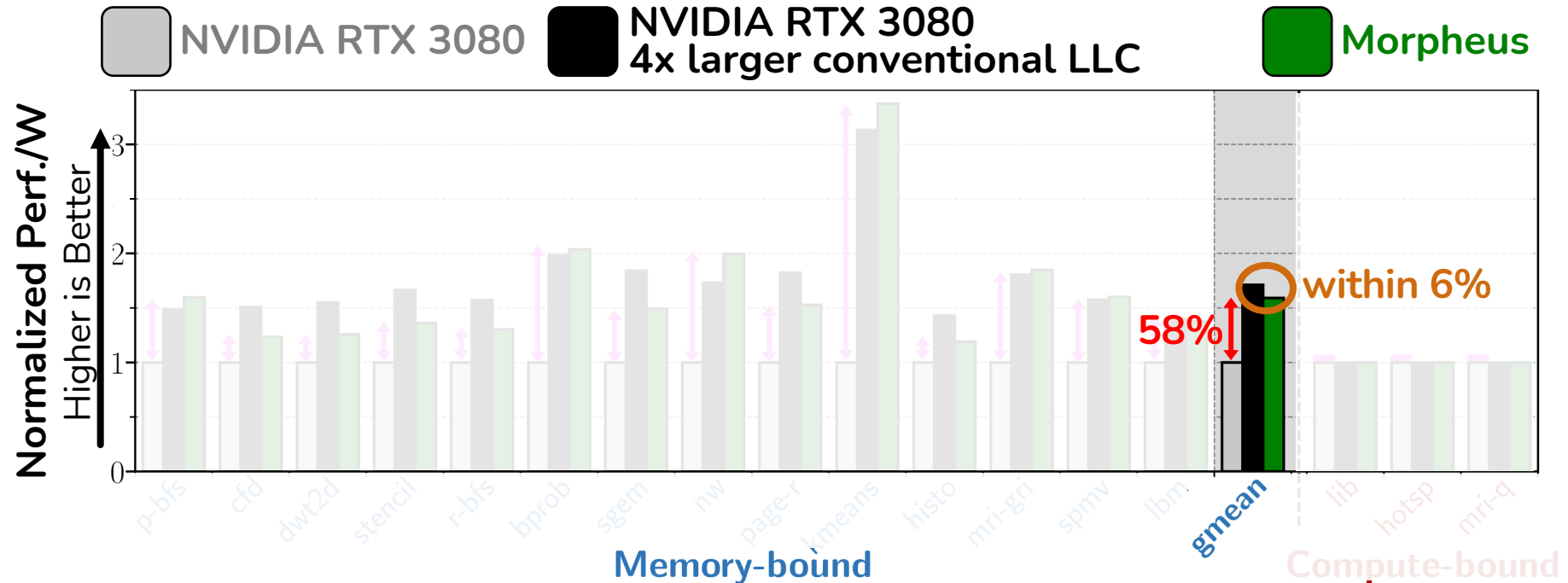
Morpheus **performs within 3%** of a **4x larger conventional LLC**

Morpheus Energy Efficiency



Morpheus uses **less** or the **same** energy as the baseline for **all** applications

Morpheus Energy Efficiency



Morpheus uses **less** or the **same** energy as the baseline for **all** applications

Morpheus **improves** energy efficiency by **58%** on average for **memory-bound** applications

Morpheus' energy efficiency is **within 6%** of a **4x larger** conventional LLC

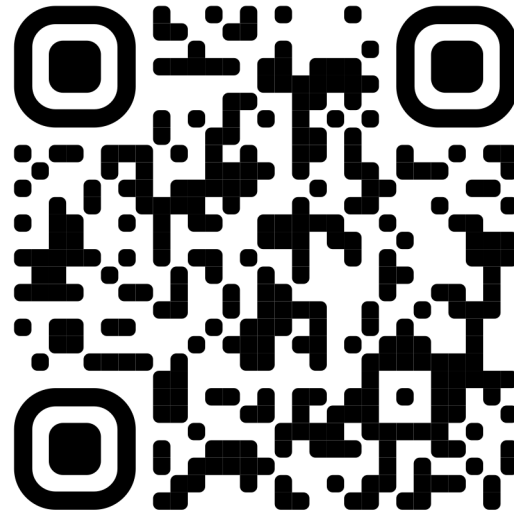
More in the Paper: Evaluation

Morpheus: Extending the Last Level Cache Capacity in GPU Systems Using Idle GPU Core Resources

*Sina Darabi[†] *Mohammad Sadrosadati[§] Joël Lindegger[§] Negar Akbarzadeh[†] Mohammad Hosseini[‡]
Jisung Park^{§∇} Juan Gómez-Luna[§] Hamid Sarbazi-Azad^{†‡} Onur Mutlu[§]

[†]*Sharif University of Technology* [§]*ETH Zürich*

[‡]*Institute for Research in Fundamental Sciences (IPM)* [∇]*POSTECH*



More in the Paper: Evaluation

- Characterization of extended LLC on a real GPU
- Performance and energy comparisons with more baselines
- Total LLC throughput with Morpheus
- On-chip and off-chip memory bandwidth utilization
- Storage and power overhead analysis

Outline

- 1 Introduction
- 2 Morpheus: Overview
- 3 Morpheus Controller
- 4 Extended LLC Kernel
- 5 Hit/Miss Prediction
- 6 Evaluation
- 7 Conclusion**

Summary

Motivation Memory-bound GPU applications leave some **cores under-utilized**

Goal Leverage the **under-utilized cores** to boost the **performance** and **energy efficiency** of **memory-bound applications**

Morpheus First technique that leverages some **GPU cores' private memories** to **extend the total GPU last-level cache capacity**

Key Results Morpheus outperforms state-of-the-art GPU architectures for memory-bound applications

- **39% performance** improvement (relative to RTX 3080)
- **58% energy efficiency** improvement (relative to RTX 3080)
- Morpheus performs **within 3%** of a **4x larger conventional LLC**
- Morpheus provides a **4x larger LLC** with the same LLC hardware

Conclusion Morpheus **effectively repurposes** the **private memory** of under-utilized cores to **extend the GPU LLC capacity**

Morpheus

Extending the Last Level Cache Capacity in GPU Systems
with Idle GPU Core Resources

Sina Darabi, Mohammad Sadrosadati, **Joël Lindegger**,
Negar Akbarzadeh, Mohammad Hosseini, Jisung Park,
Juan Gómez-Luna, Hamid Sarbazi-Azad, Onur Mutlu

ETH zürich



SAFARI
SAFARI Research Group

December 23rd 2022

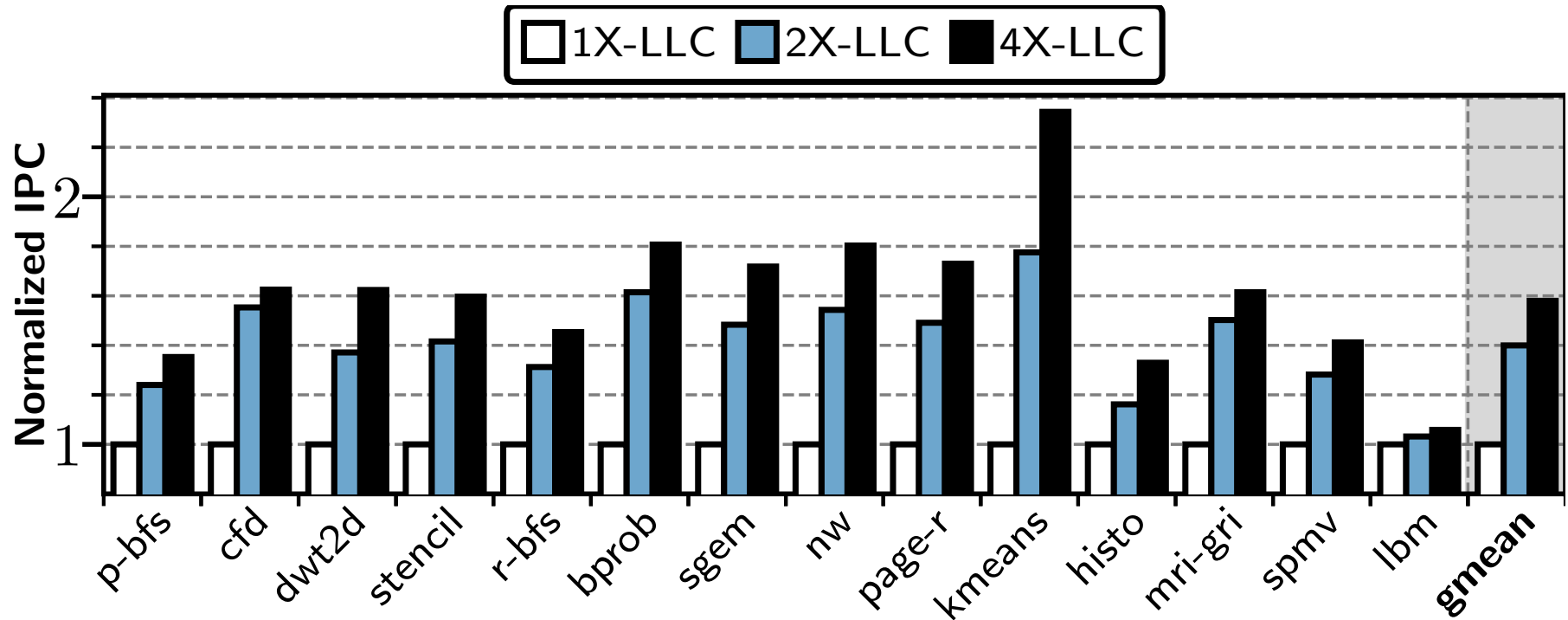
بیژوهشگاه دانش‌های بنیادی (مرکز تحقیقات فیزیک نظری و ریاضیات)
IPM
INSTITUTE FOR RESEARCH IN FUNDAMENTAL SCIENCES



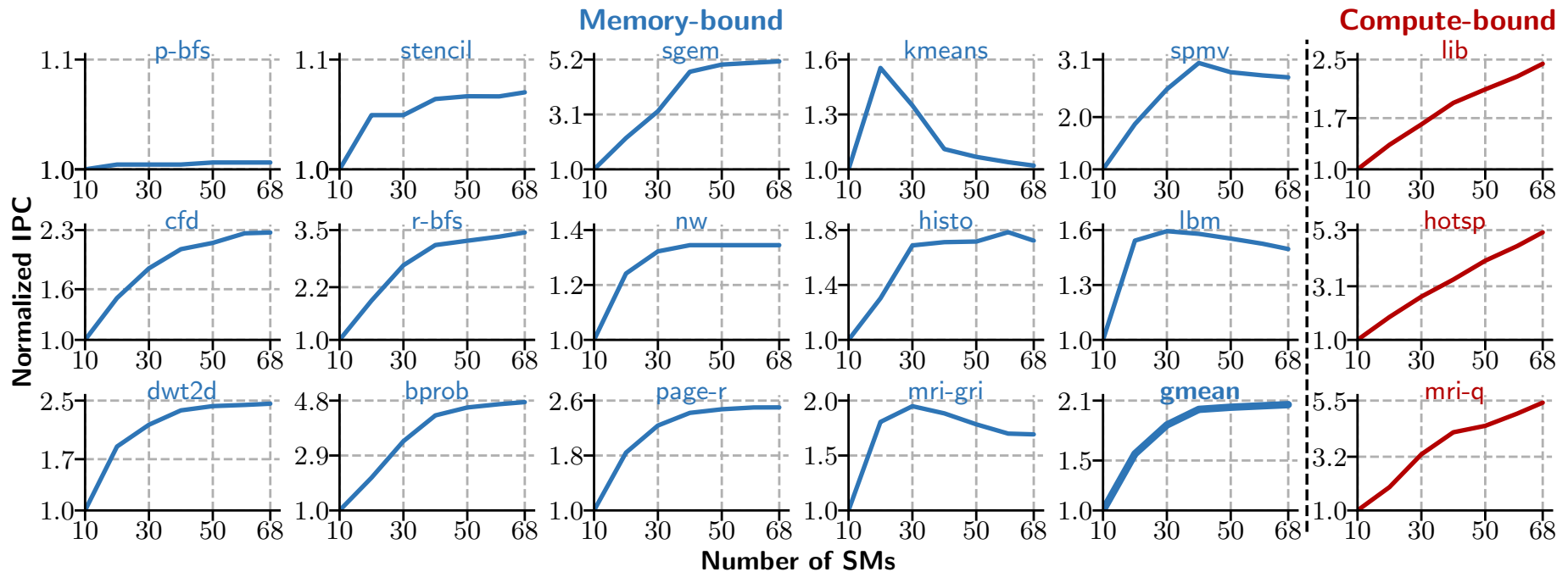
Backup Slides

- Performance Benefits of a Larger LLC
- Memory-bound vs. Compute-bound Applications
- Morpheus' Logical Data Flow
- LLC Timelines
- Hit/Miss Predictor
- Extended LLC Query Logic Unit
- Registerfile Layout
- Cache Compression
- Indirect-MOV Algorithm
- Indirect-MOV Instruction
- Extended LLC Characterization
- Simulation Parameters
- Number of Compute Mode Cores
- Morpheus LLC Capacity
- Morpheus Energy and Power
- Hit/Miss Prediction Benefits
- Evaluated Applications
- Extended LLC Tag Lookup Algorithm

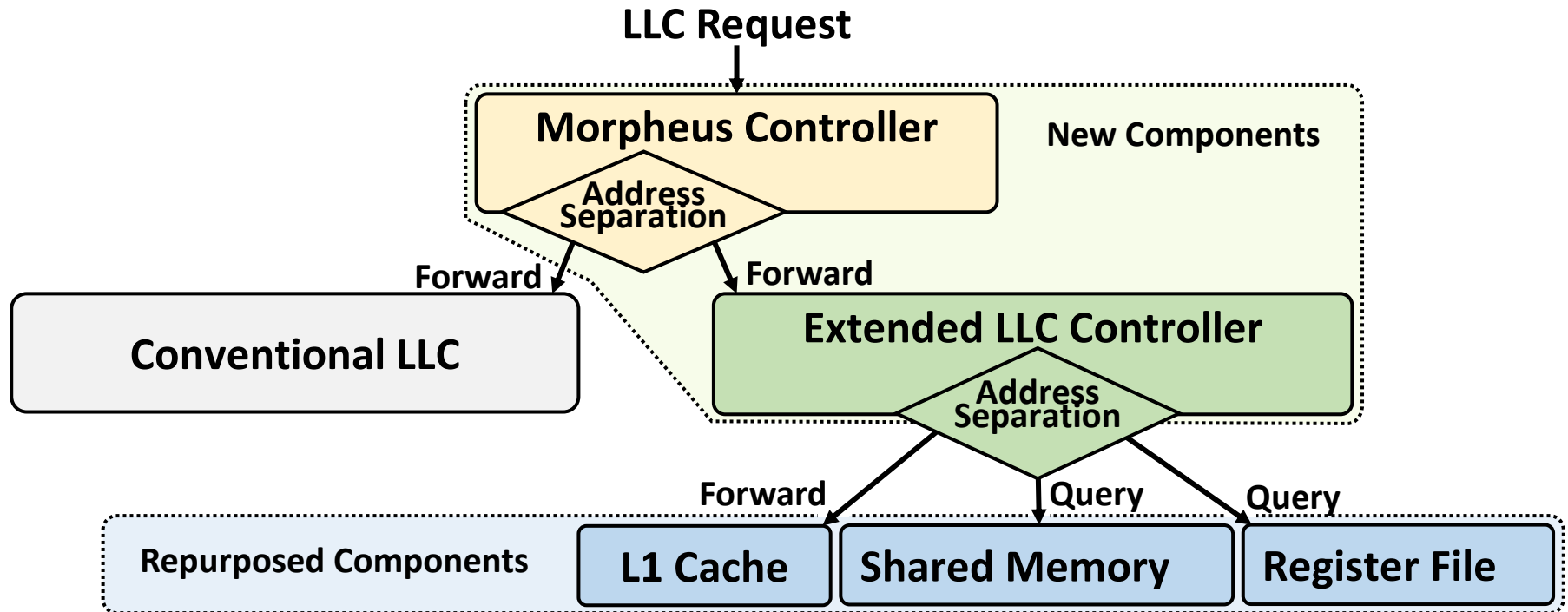
Performance Benefits of a Larger LLC



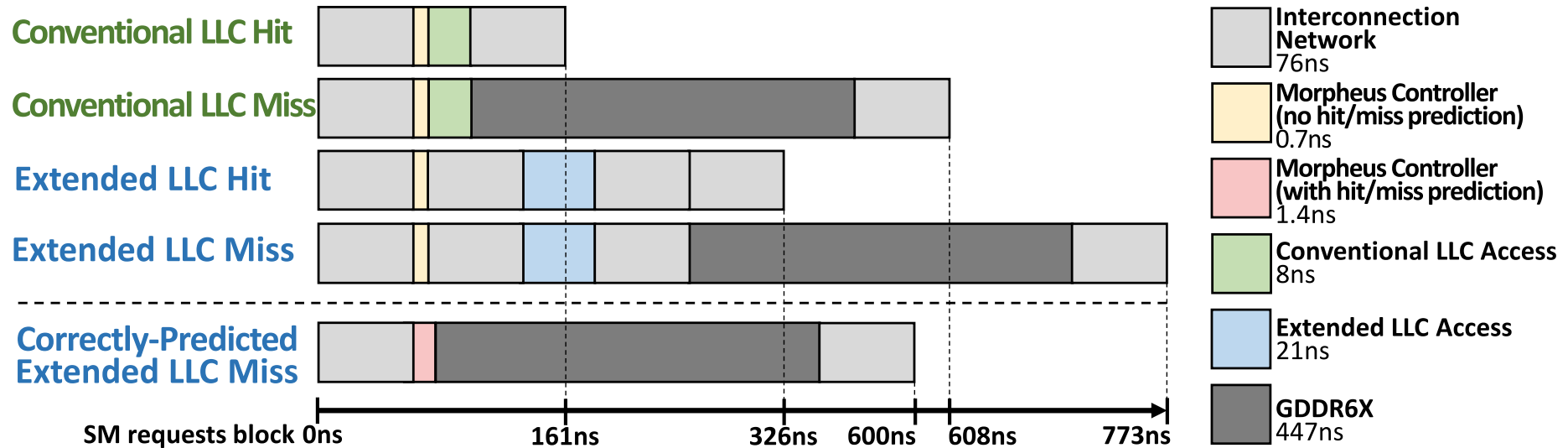
Memory-bound vs. Compute-bound Applications



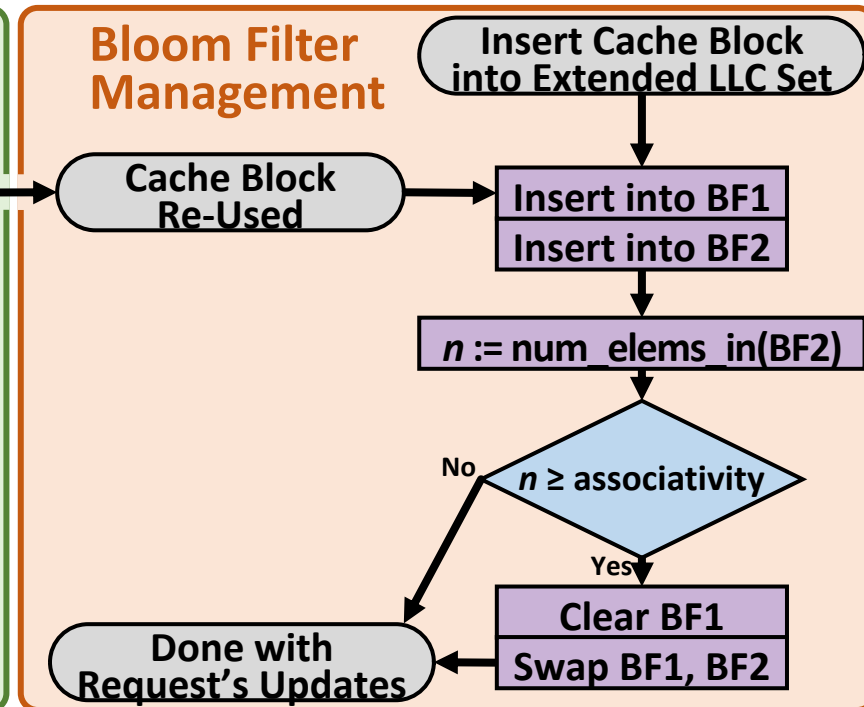
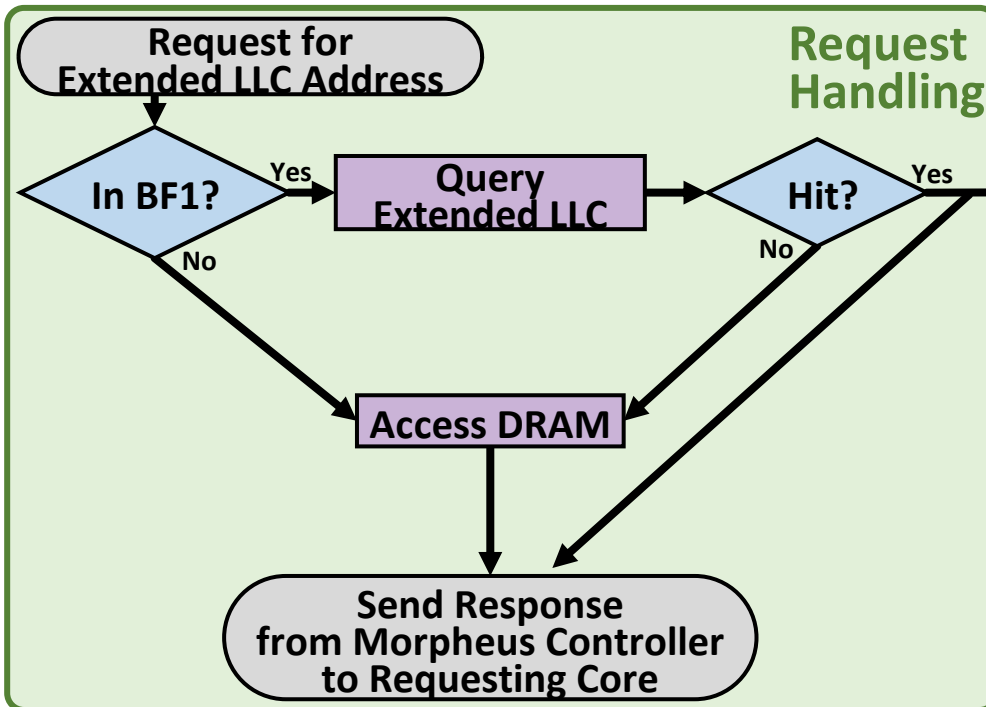
Morpheus' Logical Data Flow



LLC Timelines



Hit/Miss Predictor



Extended LLC Query Logic Unit

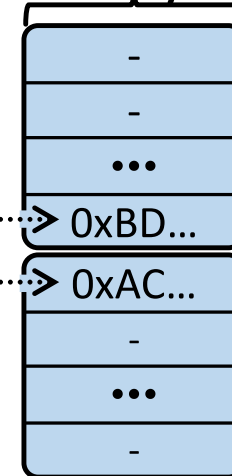
Request from Core
in Compute Mode

Request Queue

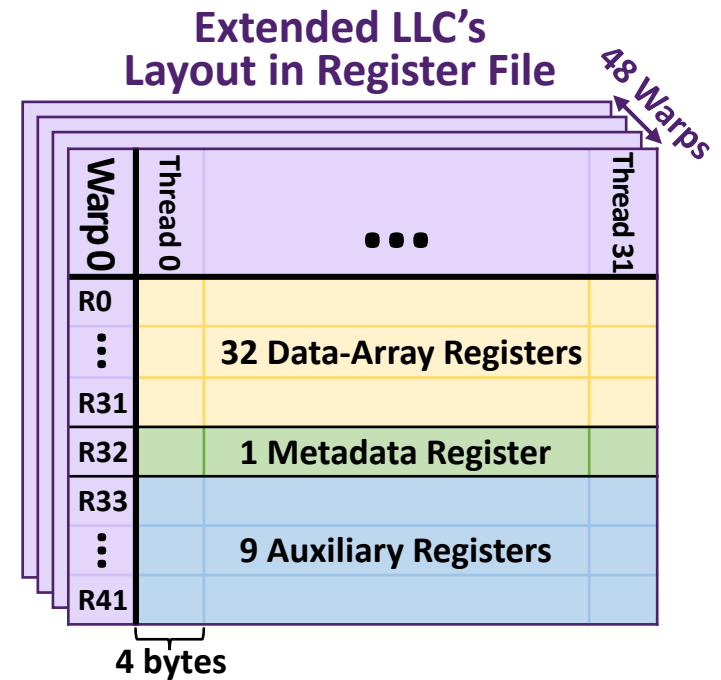
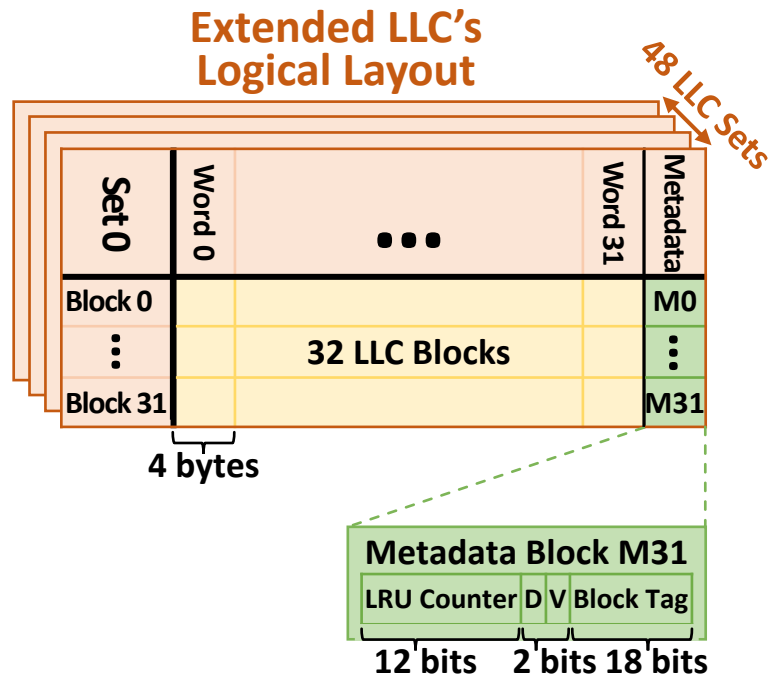
Warp Status Table

Set ID	Tag	Requesting SM	Busy	Op	Result	Data Pointer
0	0xB3...	42	1	Write	-	15
1	0xD7...	3	0	Read	Hit	0
...
255	0x5F...	12	0	Read	Miss	-

128 Bytes

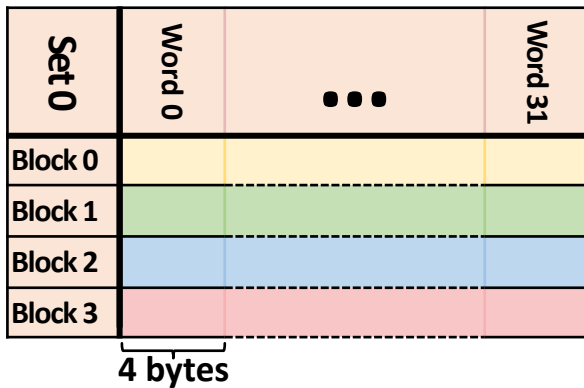


Registerfile Layout

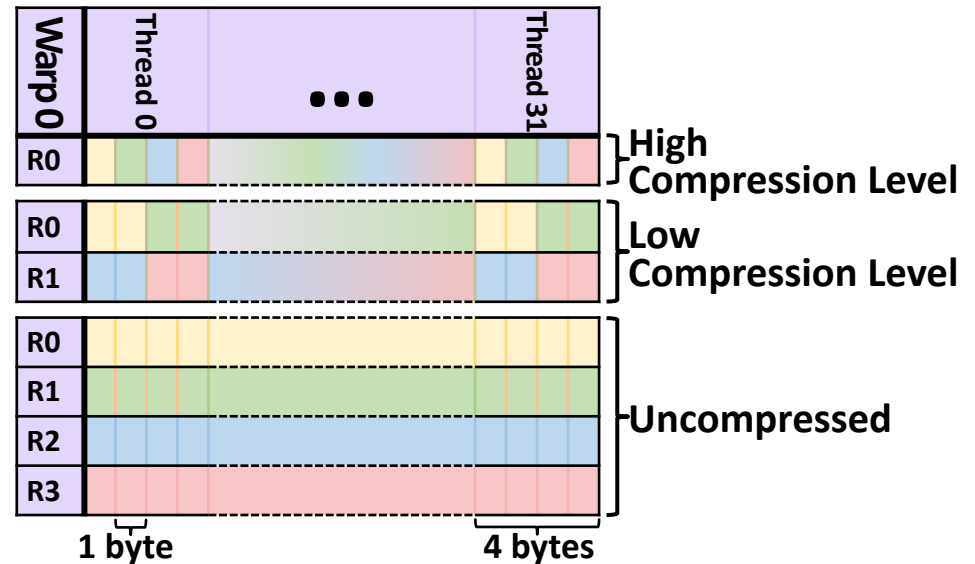


Cache Compression

Extended LLC's Logical Layout



Extended LLC's Layout across Registers with Compression



Indirect-MOV Algorithm

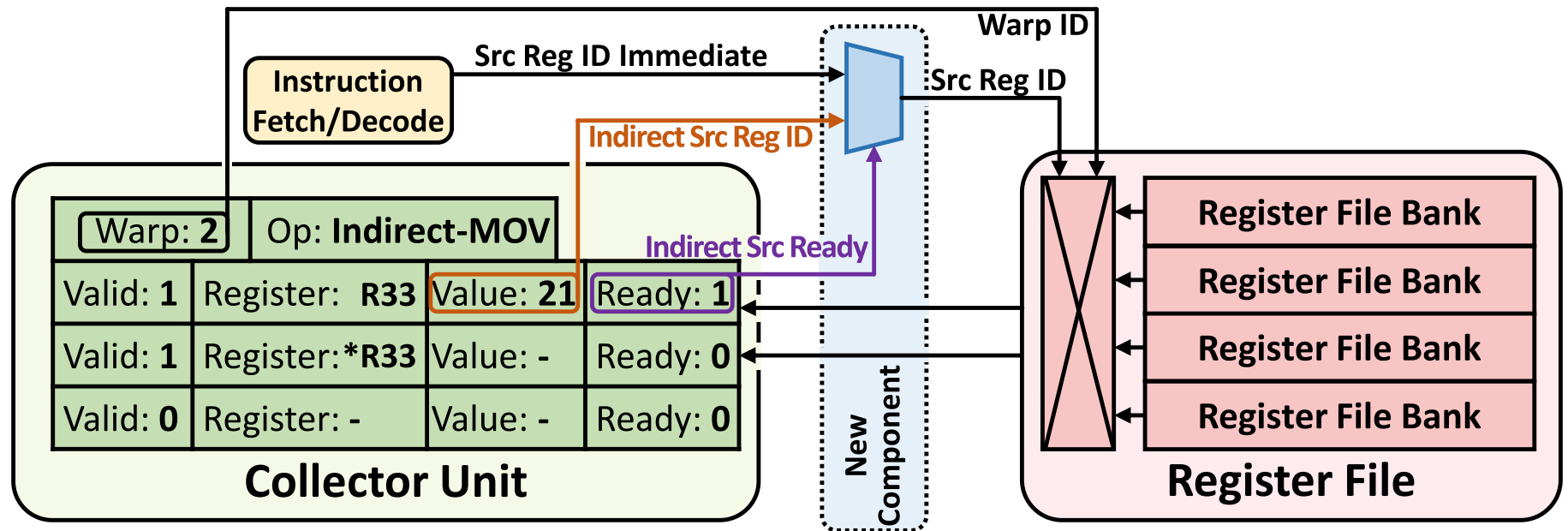
Algorithm 2 Indirect-MOV Algorithm

Input: BLOCK_INDEX:int (R_{aux_3})

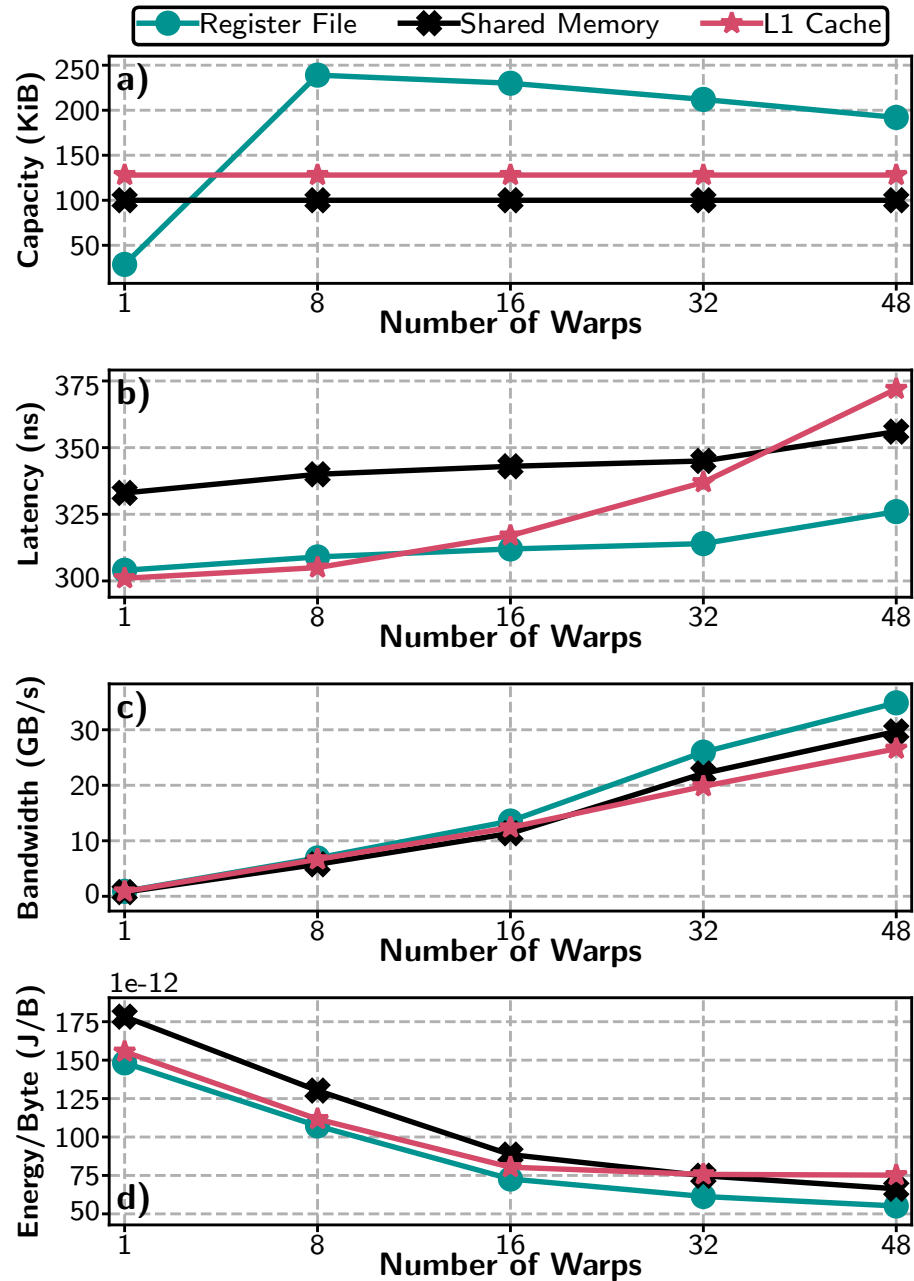
Output: Requested Extended LLC Block (R_{aux_0})

```
1: procedure INDIRECT-MOV //The goal is to implement register indirect access, reading
   from a register whose index is determined by accessing the value in another register.
   This procedure is critical for accessing data-array registers in the extended LLC
   kernel.
2:    $T_{list}$  : .Branch Targets  $L_0, L_1, L_2, \dots, L_{31}$ ; //Define 32 branch targets,
   each is allocated to access a specific register index.
3:   @p brx.idx  $R_{aux_3}, T_{list}$ ; //Branch to label  $L_i$  specified by the target LLC block
   index  $i=R_{aux_3}$ 
4:    $L_0$  :
5:     MOV  $R_0, R_{aux_0}$  //Access data-array register  $R_0$  if target LLC block index is 0
6:     return
7:    $L_1$  :
8:     MOV  $R_1, R_{aux_0}$  //Access data-array register  $R_1$  if target LLC block index is 1
9:     return
10:  ...
11:   $L_{31}$  :
12:    MOV  $R_{31}, R_{aux_0}$  //Access data-array register  $R_{31}$  if target LLC
   block index is 31
13:    return
14: end procedure
```

Indirect-MOV Instruction



Extended LLC Characterization



Simulation Parameters

- NVIDIA RTX 3080

Parameter	Value
Number of SMs	68
Scheduler	Two-Level [18, 38]
GPU Memory Interface	320-bit GDDR6X [39]
GPU Memory Capacity	10 GiB
Conventional LLC Capacity	5 MiB
L1/Shared-Memory Capacity	128 KiB per SM
Register File Capacity	256 KB per SM

Number of Compute Mode Cores

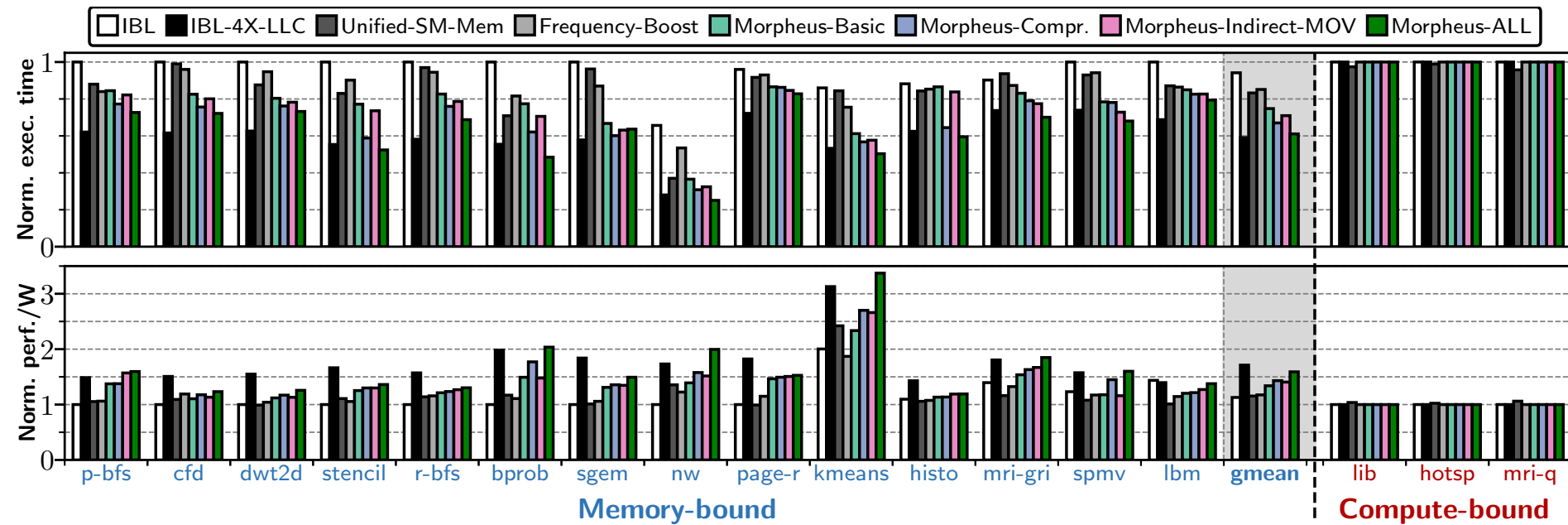
Table 3: Number of GPU cores executing application threads for different evaluated systems (#available GPU cores is 68).

Application	p-bfs	cfld	dwt2d	stencil	r-bfs	bprob	sgem	nw	page-r	kmeans	histo	mri-gri	spmv	lbm	lib	hotsp	mri-q
IBL	68	68	68	68	68	68	68	68	68	24	53	34	42	34	68	68	68
Morpheus-Basic	32	42	42	50	34	39	48	18	42	37	47	36	44	32	68	68	68
Morpheus-ALL	40	55	54	56	37	41	54	26	46	47	52	43	47	36	68	68	68

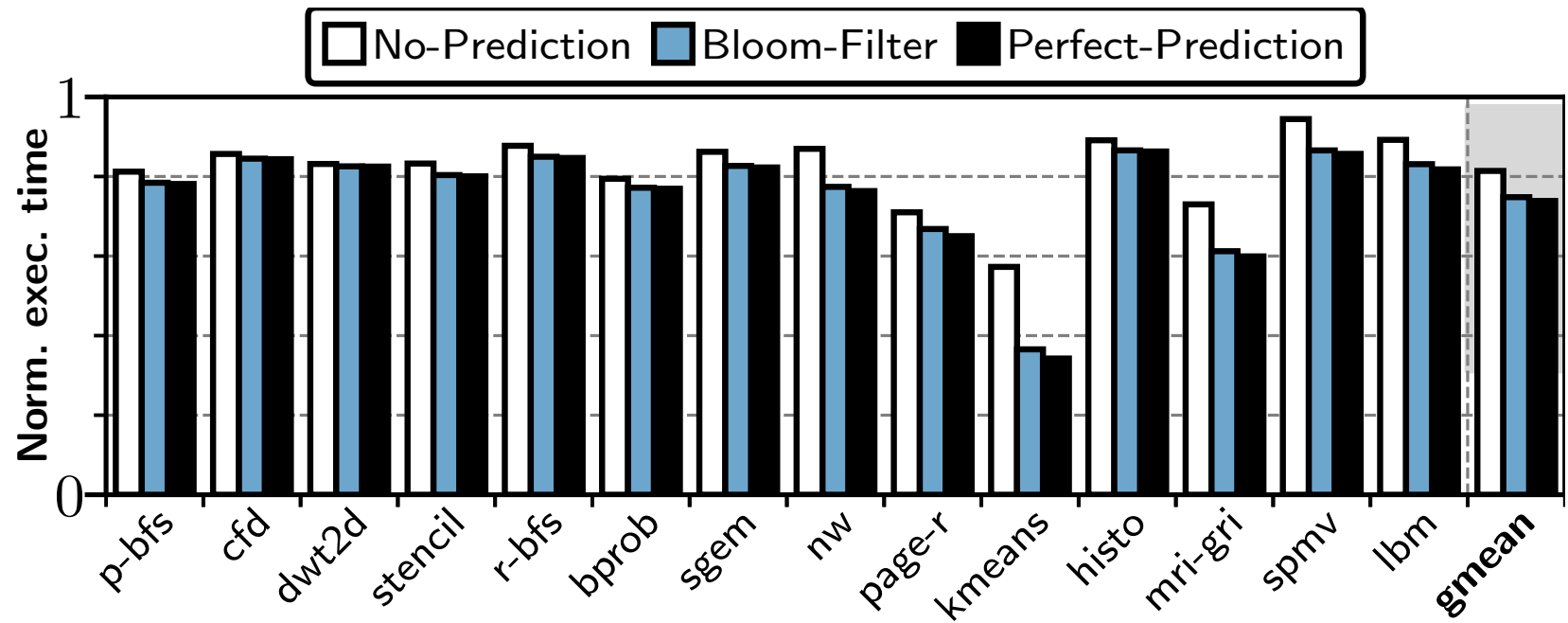
Morpheus LLC Capacity (NVIDIA RTX 3080)

- **Conventional LLC: 5 MiB**
- **Extended LLC Per Core in Cache Mode: 328 KiB**
 - 128 KiB from L1
 - 256 KiB from Register File
 - » 200 KiB useable to extended LLC data
- **Number of Cache Mode Cores: Application Dependent**
 - E.g. 50 for Needleman-Wunsch (nw) => 16 MiB
 - Theoretically up to 67 => 21.4 MiB

Morpheus Energy and Power



Hit/Miss Prediction Benefits



Evaluated Applications

Application	Name	Type
Breadth-First Search [6]	p-bfs	Memory-bound
Computational fluid dynamics [5]	cfd	Memory-bound
Discrete Wavelet Transform (2D) [5]	dwt2d	Memory-bound
Stencil [6]	stencil	Memory-bound
Breadth-First Search [5]	r-bfs	Memory-bound
Back Propagation [5]	bprob	Memory-bound
sgemm [6]	sgem	Memory-bound
Needleman-Wunsch [5]	nw	Memory-bound
Page Rank [40]	page-r	Memory-bound
K-means [5]	kmeans	Memory-bound
Histogram [6]	histo	Memory-bound
Magnetic Resonance Imaging-Gridding [6]	mri-gri	Memory-bound
Sparse-Matrix Dense-Vector Multiplication [6]	spmv	Memory-bound
Lattice-Boltzmann [6]	lbm	Memory-bound
LIBOR Monte Carlo [41]	lib	Compute-bound
HotSpot [5]	hotsp	Compute-bound
Magnetic Resonance Imaging - Q [6]	mri-q	Compute-bound

Extended LLC Tag Lookup

Algorithm 1 Extended LLC Tag Lookup – Register File

Input: Extended LLC Request's Tag (R_{aux_0})

Output: HIT:bool, and if HIT=True, BLOCK_INDEX:int (R_{aux_3})

```
1: procedure TAG LOOKUP //executed by an extended LLC kernel warp of 32 threads
2:    $R_{aux_1} \leftarrow Valid(R_M)$  //ensure the block is valid
3:    $R_{aux_1} \leftarrow R_{aux_1} \ \&\& \ (R_{aux_0} == Tag(R_M))$  //match request tag to metadata
4:    $R_{aux_2} \leftarrow \_\_ballot\_sync(0xffffffff, R_{aux_1})$  //share  $R_{aux_1}$  between all
   threads as a 32-bit vector
5:   if ( $R_{aux_2}$ ) then //one of the bits is non-zero because there was a hit
6:      $R_{aux_3} \leftarrow \_\_ffs(R_{aux_2}) - 1$  //get the 0-based index of the non-zero bit
7:     HIT  $\leftarrow$  True
8:     BLOCK_INDEX  $\leftarrow R_{aux_3}$ 
9:     if ( $thread\_idx == R_{aux_3}$ ) then //reset the LRU counter of the hit block
10:       $LRU\_Counter(R_M) \leftarrow 0xfff$ 
11:    else //decrement the LRU counters of all other blocks
12:       $LRU\_Counter(R_M) \leftarrow LRU\_Counter(R_M) - 1$ 
13:    end if
14:  else
15:    HIT  $\leftarrow$  False
16:  end if
17: end procedure
```
