Exploiting Core Criticality for Enhanced GPU Performance

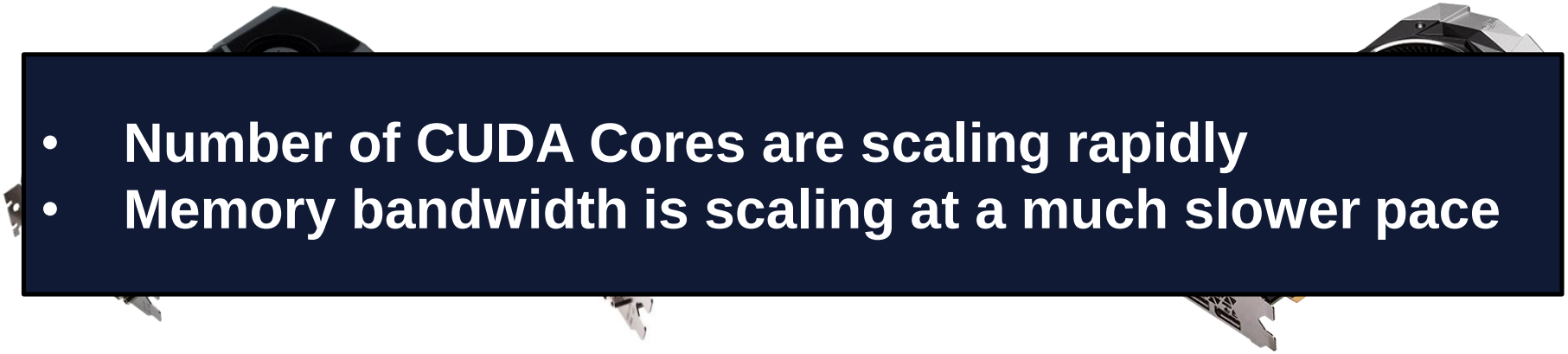
Adwait Jog, Onur Kayıran, Ashutosh Pattnaik,
Mahmut T. Kandemir, Onur Mutlu,
Ravishankar Iyer, Chita R. Das.

SIGMETRICS '16

WILLIAM & MARY
AMD
PennState
ETH zürich
Carnegie Mellon University
intel

Era of Throughput Architectures

GPUs are scaling: Number of CUDA Cores, DRAM bandwidth



- **Number of CUDA Cores are scaling rapidly**
- **Memory bandwidth is scaling at a much slower pace**

2010:	2012:	2016:
GTX 480	GTX 680	GTX 1080
(Fermi)	(Kepler)	(Pascal)
448 cores	1536 cores	2560 cores
(178 GB/sec)	(192 GB/sec)	(320 GB/sec)

Current Trend

- Modern Schedulers (e.g., FR-FCFS)
  - assume that all memory requests are *equally critical* towards performance.
  - *maximize* memory data throughput.
- Inability of FR-FCFS to distinguish memory requests from different GPU cores lead to
  - GPU cores experiencing significant variation in average memory access latencies
  - some GPU cores becoming more "*critical*" than others

Coefficient of Variation (COV) in Average Memory Access Latencies

- To understand further,
  - consider the COV (ratio of Standard Deviation over Arithmetic Mean) in memory

- **We need to take core criticality into account.**
  - **Prioritize requests from GPU cores with less latency tolerance**
- **Contention is present in entire memory hierarchy.**
  - **In this work, we only consider main memory contention**
  - **We propose CLAMS, a criticality aware memory scheduling mechanism**

LUH	RED	SCAN	LPS	RAY	CONS	SCP	BLK	HS	CFD	GAUSS	AVG.

  - some GPU cores experience higher avg. memory latency than others
  - these cores are less latency tolerant (“critical”)
  - latency variations correlate with IPC variation

Outline

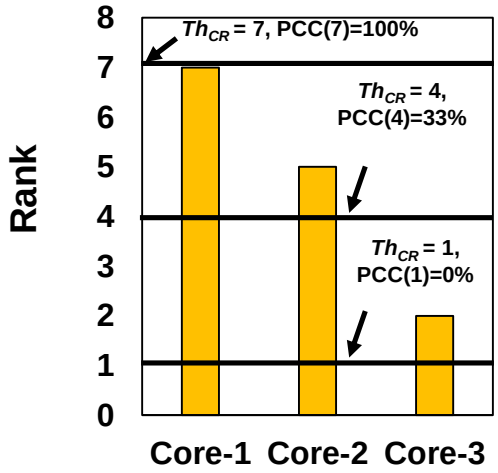
- Introduction and Motivation
- Core Criticality: Metrics and Analysis
- Design of Criticality Aware Memory Scheduler
- Infrastructure Setup and Evaluation
- Conclusions

Core Criticality: Metrics

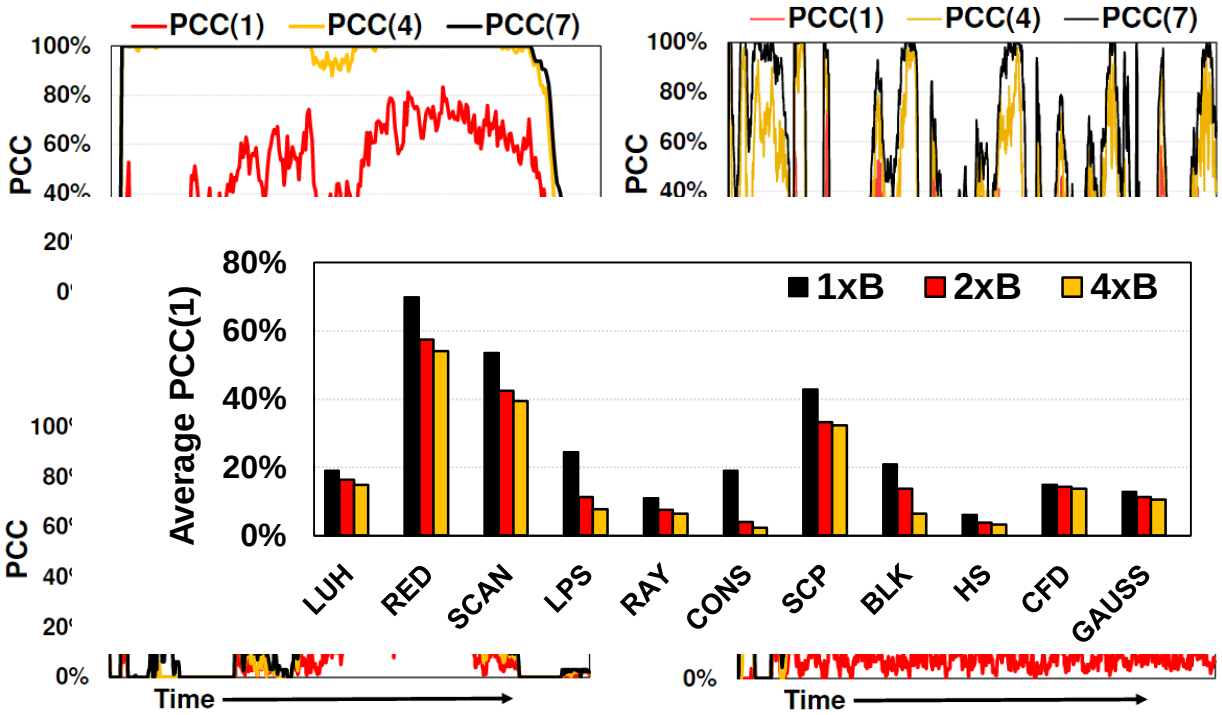
- We need to quantify core criticality.
- Use latency tolerance as a measure of core criticality.
  1. Classify warps into *short-* and *long-latency* warp
     - Short-latency: compute instruction/data in private cache
     - Long-latency: stalled due to pending memory requests
  2. Calculate *short-latency ratio*
     - Ratio of short-latency warps over total issued warps
  3. Assign criticality rank
     - Quantize short-latency ratio
       - 8 discrete steps, with step size of 1/8.
       - *rank-1*: *short-latency ratio* < *1/8,* Most critical
       - *rank-8*: *short-latency ratio* > *7/8,* Least critical

Core Criticality: Metrics

- ***Th_{CR}*** *: Criticality-Rank-Threshold*
  - Core is critical if current *rank* <= ***Th_{CR}***
  - Takes any integer value from 1 to 8.
- ***$PCC(Th_{CR})$****: Percentage of critical cores for given* ***Th_{CR}***
  - ***$PCC(Th_{CR})$*** of 100%, all cores critical
  - ***$PCC(Th_{CR})$*** of 0%, all cores non-critical



Core Criticality: Metrics



(c) RAY (d) CONS

1. PCC is dependent on the chosen criticality-rank threshold
2. PCC varies within an application over time
3. PCC varies across applications
4. PCC reduces significantly as main memory bandwidth increases

Core Criticality: Metrics and Analysis

- If ***PCC(Th_{CR})*** is low for any given Th_{CR}, then
  - GPU cores have similar latency tolerance
  - Memory scheduler should preserve locality

- ***PCC(Th_{CR})*** is calculated periodically and requires
  - Exchange of global information across cores and MCs
  - Hardware overhead of calculation

- Expensive approach!

- We need a metric which can be calculated at the MCs

Core Criticality: Metrics and Analysis

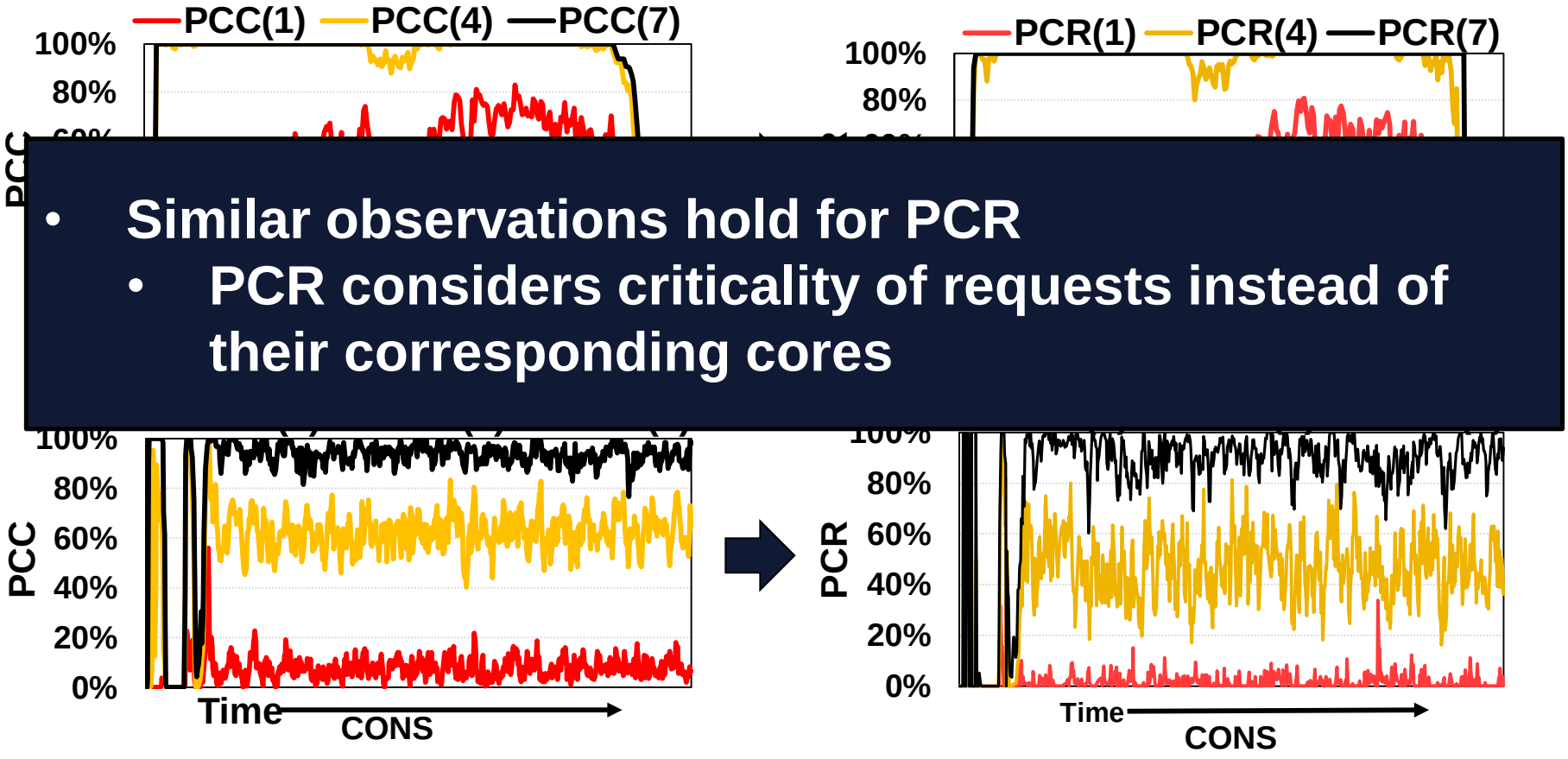
- We use Percentage of Critical Requests (PCR).
  - Tag memory requests with core's current rank
  - Calculate the percentage of critical memory requests present in the MC request buffer

- Eliminates the exchange of global information between GPU cores and MCs

- Similar to PCC, PCR needs to be calculated for a given Th_{CR}.

Core Criticality: Metrics and Analysis

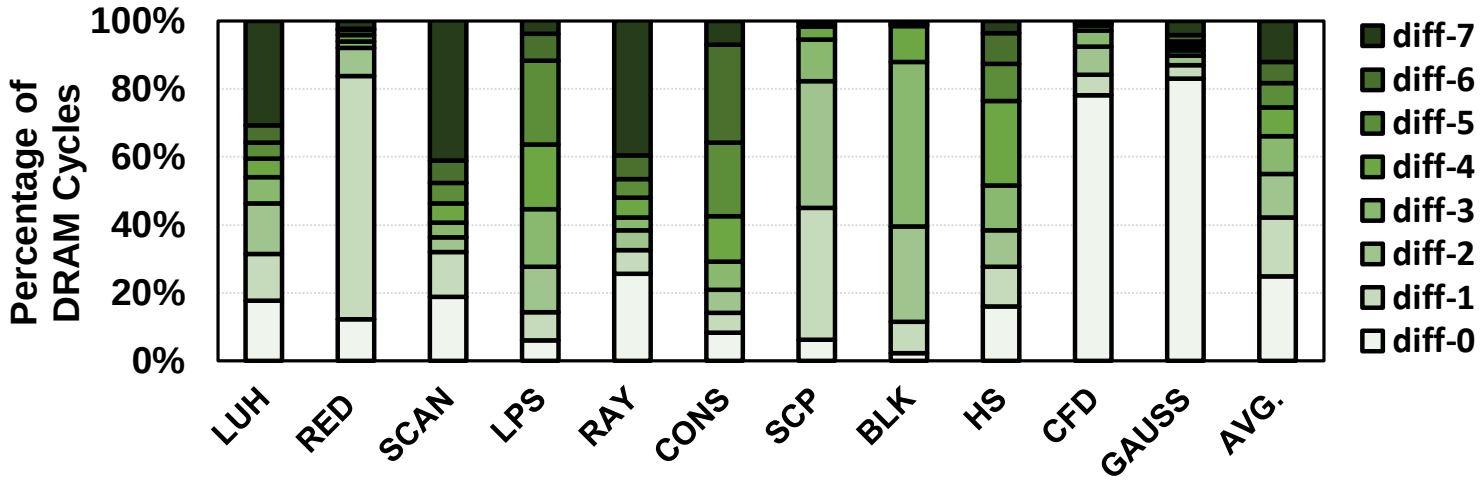
- PCR patterns for an application similar to PCC



- **Similar observations hold for PCR**
  - **PCR considers criticality of requests instead of their corresponding cores**

Core Criticality: Analysis

- Scope of criticality aware memory scheduler



- **Significant rank difference in LUH, RAY, SCAN**
  - **High scope**
- **Rank difference is 0 for most of the time for CFD, GAUSS**
  - **Low scope**

Outline

- Introduction and Motivation
- Core Criticality: Metrics and Analysis
- Design of Criticality Aware Memory Scheduler
- Infrastructure Setup and Evaluation
- Conclusions

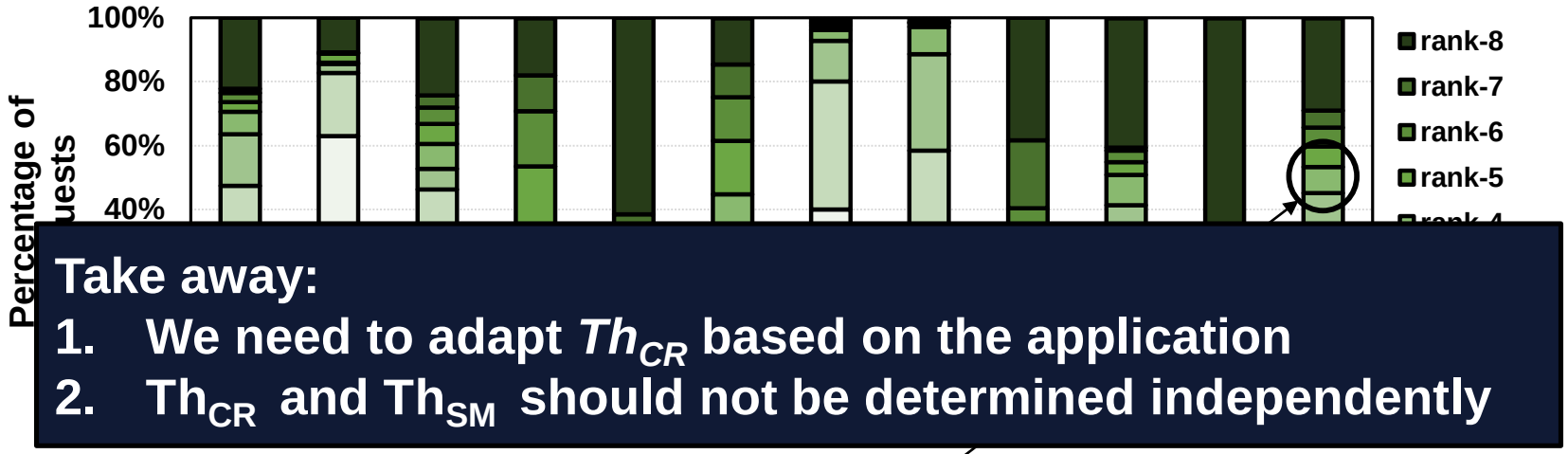
Design of CLAMS

- Two major challenges
  - Co-existence of critical and non-critical requests
    - Finding appropriate value of Th_{CR}
      - Low Th_{CR} -> less critical cores
      - High Th_{CR} -> too many critical cores
  - Balancing DRAM locality and criticality
    - Switching between schedulers optimized for criticality or locality
    - Calculate PCR(Th_{CR}) periodically and compare with Switching-Mode-Threshold(Th_{SM})
      - PCR(Th_{CR}) > Th_{SM}, locality mode
      - PCR(Th_{CR}) <= Th_{SM}, criticality mode
    - Need to find appropriate value of Th_{SM}

Design of CLAMS

- Three different approaches
  - Static-CLAMS
    - Single and fixed set values for Th_{CR} and Th_{SM}
  - Semi-Dyn-CLAMS
    - Dynamically calculates Th_{CR}, based on fixed Th_{SM} and $PCR(k)$ $\forall$ k information at MC
  - Dyn-CLAMS
    - Calculates both, Th_{CR} and Th_{SM}.
- Working modes of CLAMS
  - Decided based on per bank's memory requests
  - Locality mode
  - Criticality mode

Design of Static-CLAMS



Take away:

1. **We need to adapt Th_{CR} based on the application**
2. **Th_{CR} and Th_{SM} should not be determined independently**

- rank-4 provides a mix of both, critical and non-critical requests
  - We choose Th_{CR} = 4
- But, many applications have different distribution such as SCP
  - Assuming Th_{SM} = 20% -> locality mode most of the time
  - Assuming Th_{SM} = 80% -> criticality mode most of the time
    - Potentially degrade DRAM row buffer locality

Design of Semi-Dyn-CLAMS

- Computes Th_{CR}
  - Based on fixed Th_{SM} value, and $PCR(k)$ $\forall$ $k \in \{1...8\}$ information at MC
  - We find a value for Th_{CR} such that PCR(Th_{CR}) is $\leqq Th_{SM}$ and is as close to it as possible
- This will switch scheduler into criticality mode
- In case no such Th_{CR} can be found, switch to locality mode

Design of Semi-Dyn-CLAMS

- Lets look at SCP
- Assume Th_{SM} **= 40%,** and Th_{CR} can take any value in {1,4,7}.

Take away:
1. Semi-Dyn-CLAMS aggressively uses criticality mode
2. Only goes to locality mode, when too many critical requests present in the MC buffer
3. Can lead to significant loss in locality and performance
4. No feedback on Th_{SM} when new value of Th_{CR} is calculated.

- works in criticality mode. ❶
- For second half of execution, for no $Th_{CR,}$ is PCR(Th_{CR}) <= 40%. Scheduler switches to locality mode. ❷
  - Most of the requests are critical and cannot be prioritized

Design of Dyn-CLAMS

- Even though Semi-Dyn-CLAMS facilitates criticality mode
  - Actual mode based on requests to each bank

Key Idea:
1. Gauge the negative effect of loss in row locality on latency tolerance
2. Mitigate the loss by lowering Th_{SM} while maintaining the value of Th_{CR}.

locality.

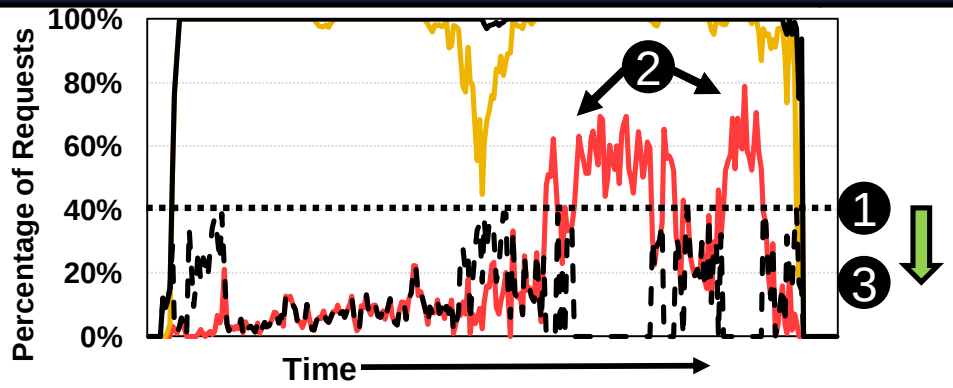
- Dynamically calculates Th_{CR} and updates Th_{SM} based on the new value of Th_{CR}.

Design of Dyn-CLAMS

- Initialize Th_{SM} **= 40%,** and Th_{CR} = 8 ❶
- Calculate Th_{CR} based on Semi-Dyn-CLAMS
  - Th_{CR} updated to 1

Take away:

1. Dynamically updating Th_{CR} allows scheduler to aggressively work in criticality mode
2. By reducing Th_{SM}, the scheduler's starts to improve locality by using locality mode for the banks
   1. chances of a bank's PCR(Th_{CR}) > Th_{SM} increase



Outline

- Introduction and Motivation
- Core Criticality: Metrics and Analysis
- Design of Criticality Aware Memory Scheduler
- Simulation Setup and Evaluation
- Conclusions

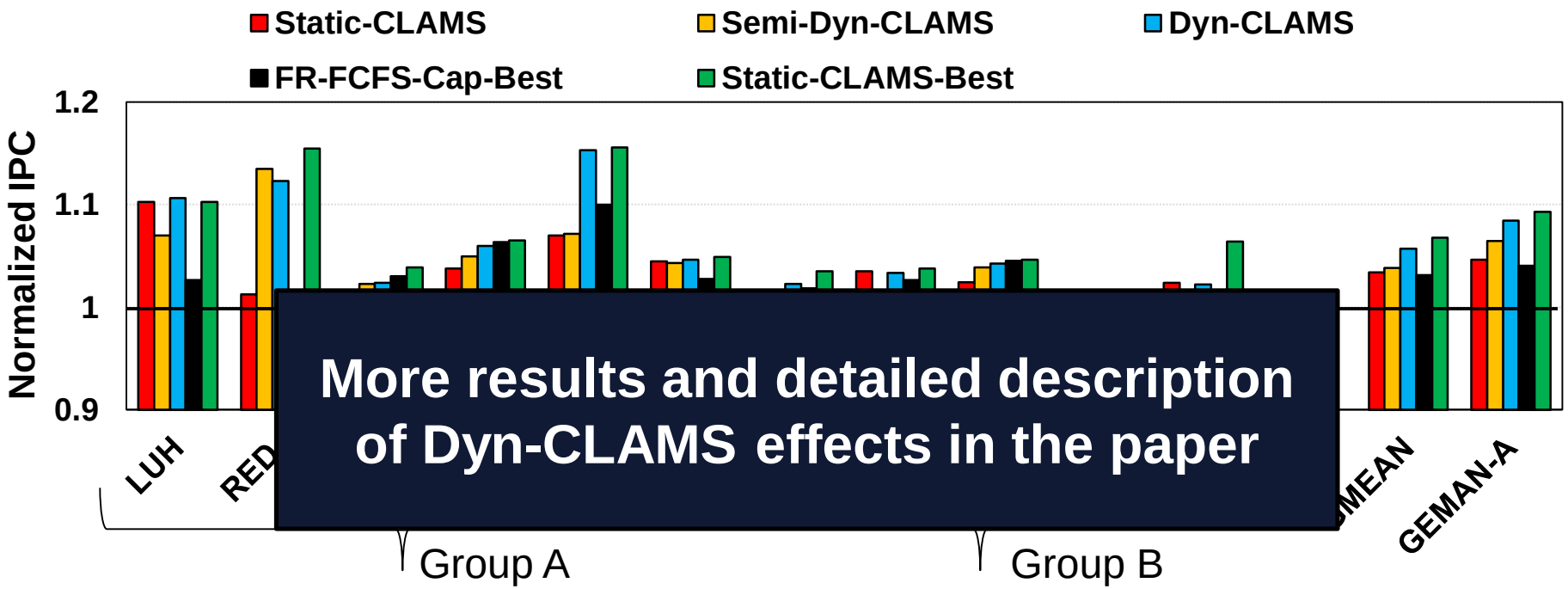
Simulation Setup

- Evaluated on GPGPU-Sim, a cycle accurate GPU simulator

- Baseline configuration similar to scaled-up version of GTX480
  - 32 SMs, 32-SIMT lanes, 32-threads/warp
  - 16KB L1 (4-way, 128B cache block) + 48KB Shared Mem/SM
  - 6 partitions/channels

- Applications classified into 2 groups
  - Group A: Moderate to High scope
  - Group B: Low scope

Performance with CLAMS (Normalized to FR-FCFS)



- Performance improvement for Group A applications
  - Static-CLAMS = 4.6%
  - Static-CLAMS-Best = 9.3%
  - FR-FCFS-Cap-Best = 4%
  - Semi-Dyn-CLAMS = 6.5%
  - Dyn-CLAMS = 8.4%

Take Away Messages

- Variation in average memory access latencies across the GPU cores, makes some GPU cores more "critical" than others.
- Memory schedulers being agnostic to the criticality of GPU cores can lead to sub-optimal performance
- We need to orchestrate the exploitation of core-criticality and DRAM locality
- Dyn-CLAMS provides an average performance improvement of 8.4%.

Exploiting Core Criticality for Enhanced GPU Performance

Adwait Jog, Onur Kayıran, Ashutosh Pattnaik,
Mahmut T. Kandemir, Onur Mutlu,
Ravishankar Iyer, Chita R. Das.

SIGMETRICS '16

WILLIAM & MARY
AMD
PennState
ETH zürich
Carnegie Mellon University
intel