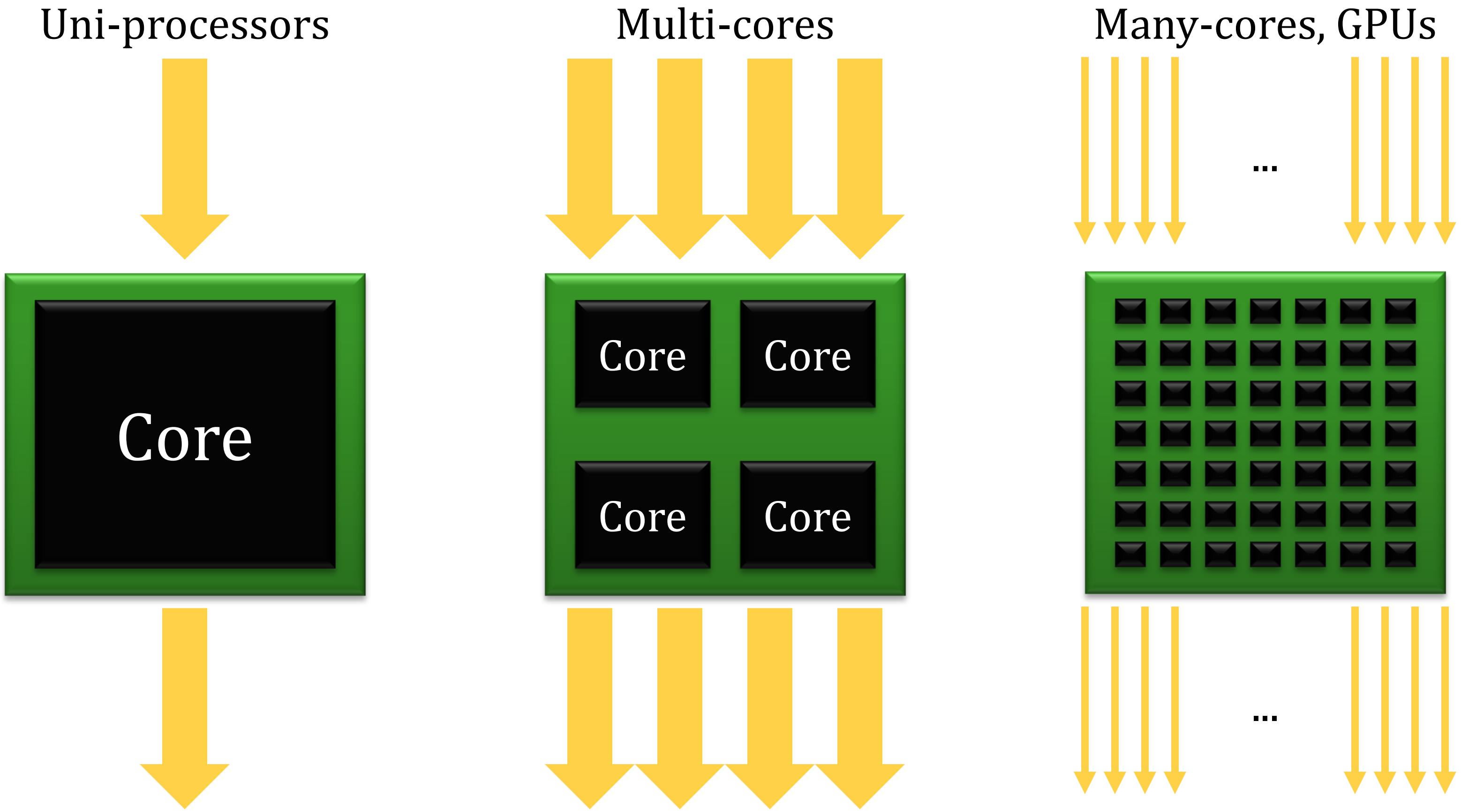
Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs

Onur Kayıran, Adwait Jog, Mahmut Kandemir, Chita R. Das

GPU Computing



- GPUs are known for providing high thread-level parallelism (TLP).

“Too much of anything is bad,

but too much good whiskey is barely enough”

- Mark Twain

Executive Summary

- Current state-of-the-art thread-block schedulers make use of the maximum available TLP
- More threads → more memory requests
- Contention in memory sub-system

Proposal:

A thread-block scheduling algorithm

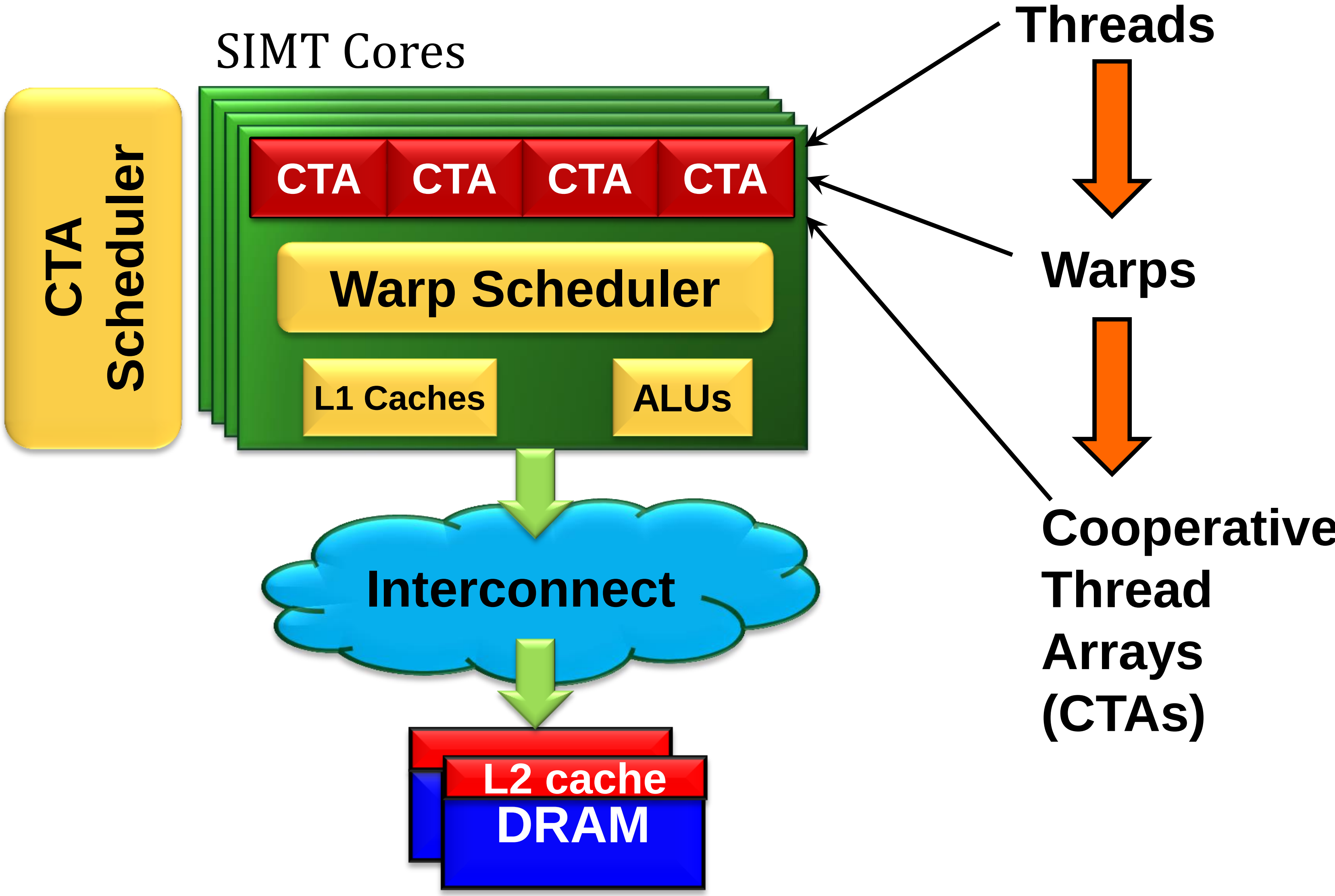
Optimizes TLP and reduces memory sub-system contention

- Improves average application performance by 28%

Outline

- Proposal
- Background
- Motivation
- DYNCTA
- Evaluation
- Conclusions

GPU Architecture



SIMT Cores

CTA Scheduler

CTA CTA CTA CTA

Warp Scheduler

L1 Caches

ALUs

Interconnect

L2 cache

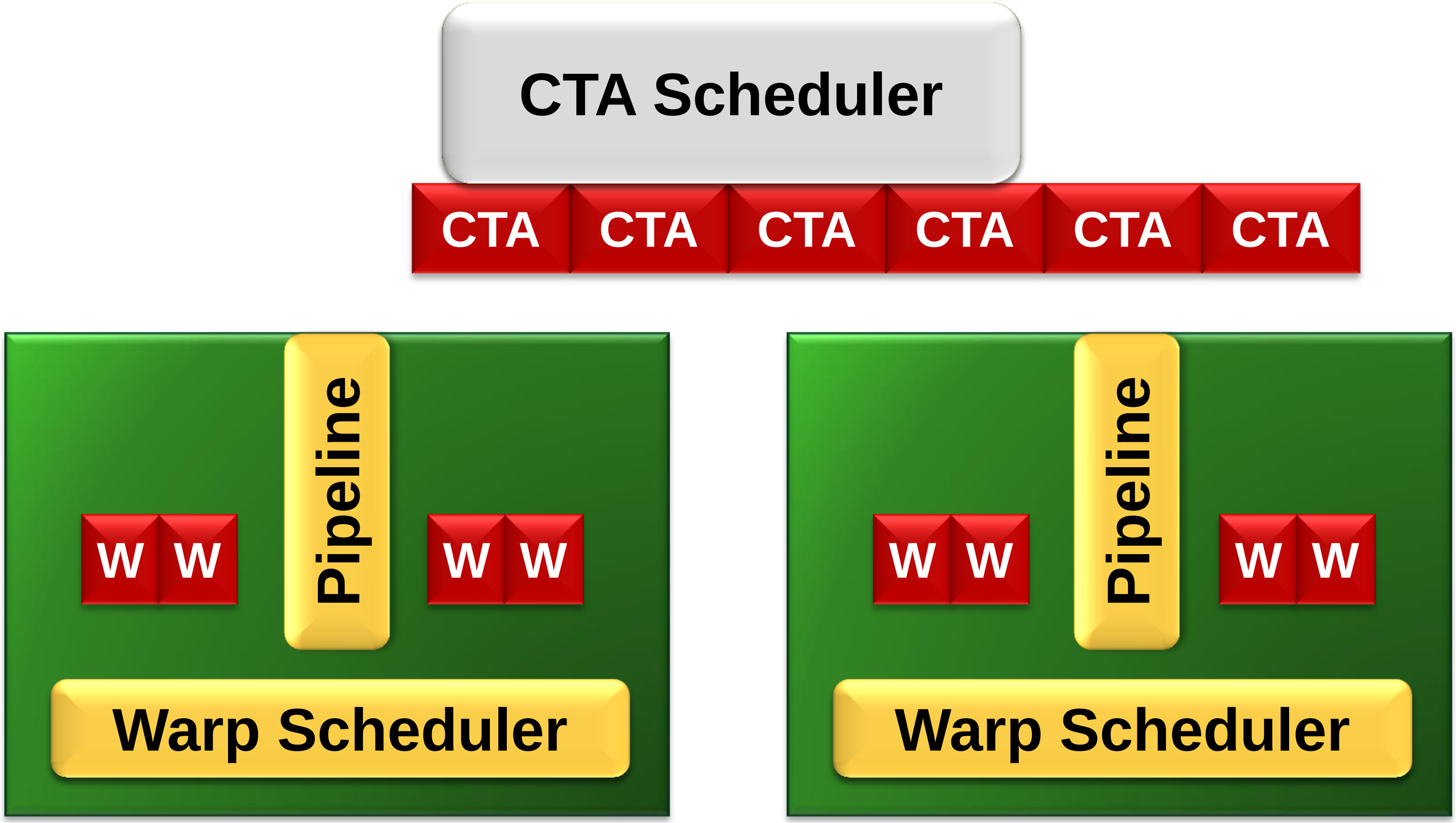
DRAM

Threads

Warps

Cooperative Thread Arrays (CTAs)

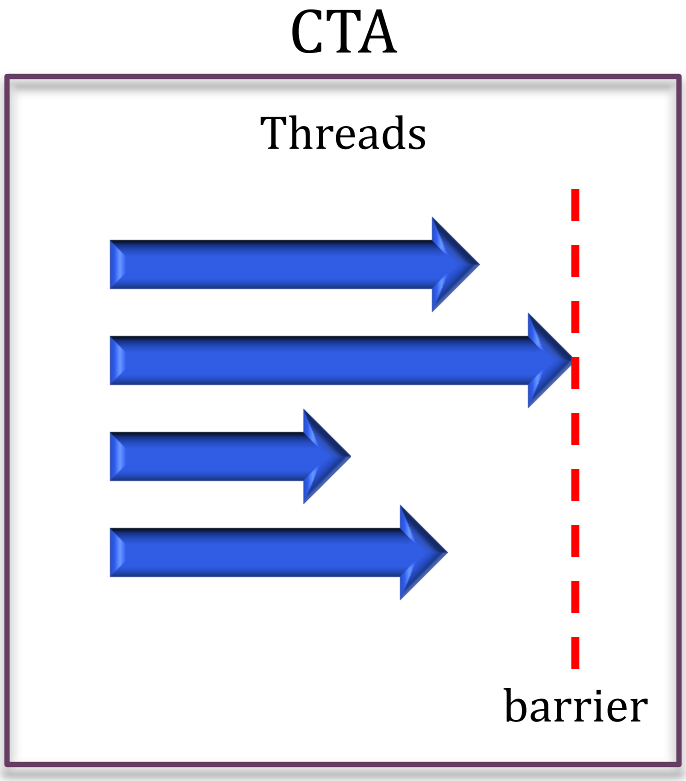
GPU Scheduling



Properties of CTAs

- Threads within a CTA synchronize using barriers.
- There is no synchronization across threads belonging to different CTAs.

- CTAs can be distributed to cores in any order.
- Once assigned to a core, a CTA cannot be preempted.



Properties of CTAs

- The number of CTAs executing on a core is limited by:
  - the number of threads per CTA
  - the amount of shared memory per core
  - the number of registers per core
  - a hard limit (depends on CUDA version for NVIDIA GPUs)
  - the resources required by the application kernel

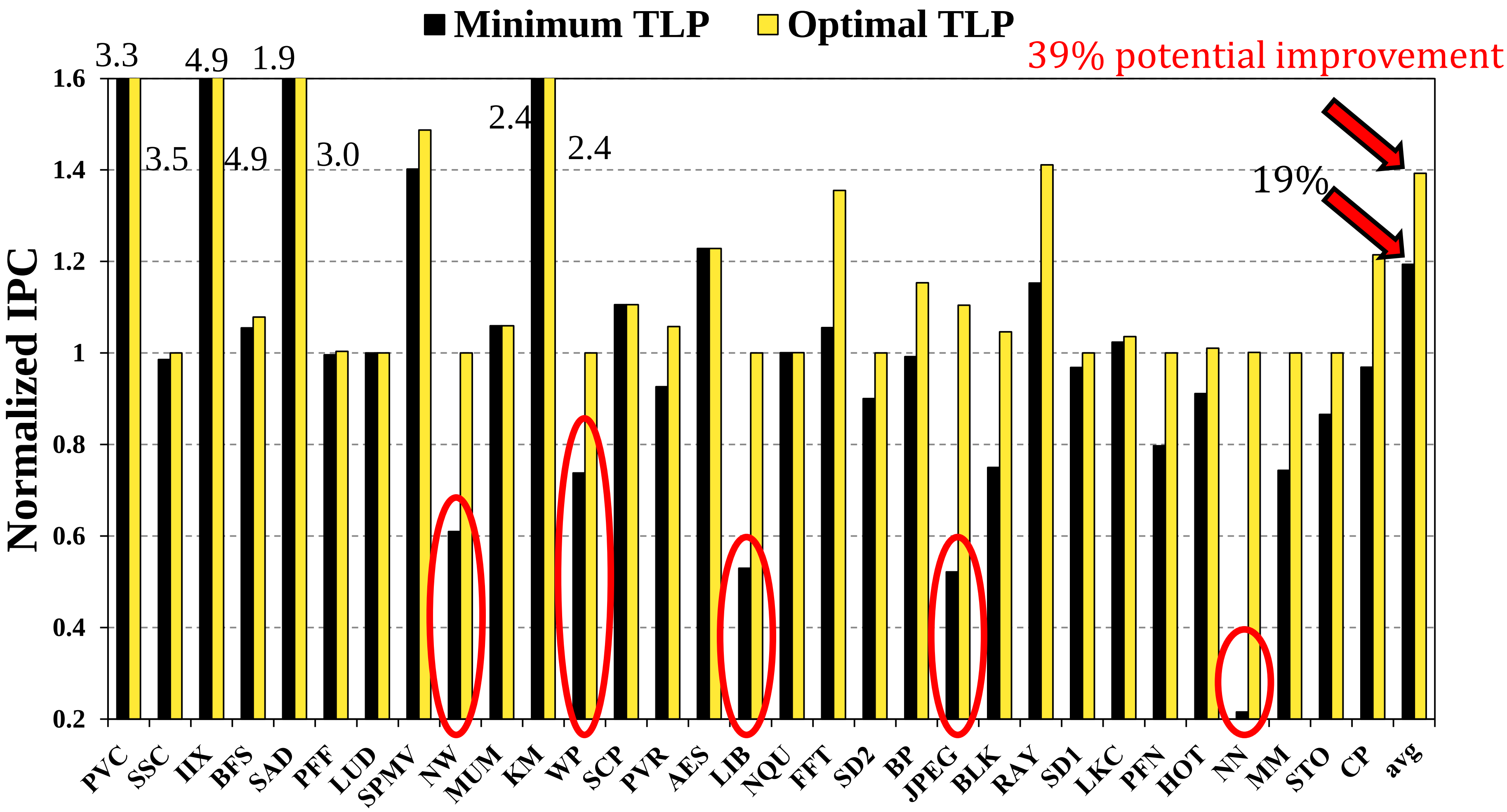
- These factors in turn limit the available TLP on the core.

- By default, if available, a core executes maximum number of CTAs.

Outline

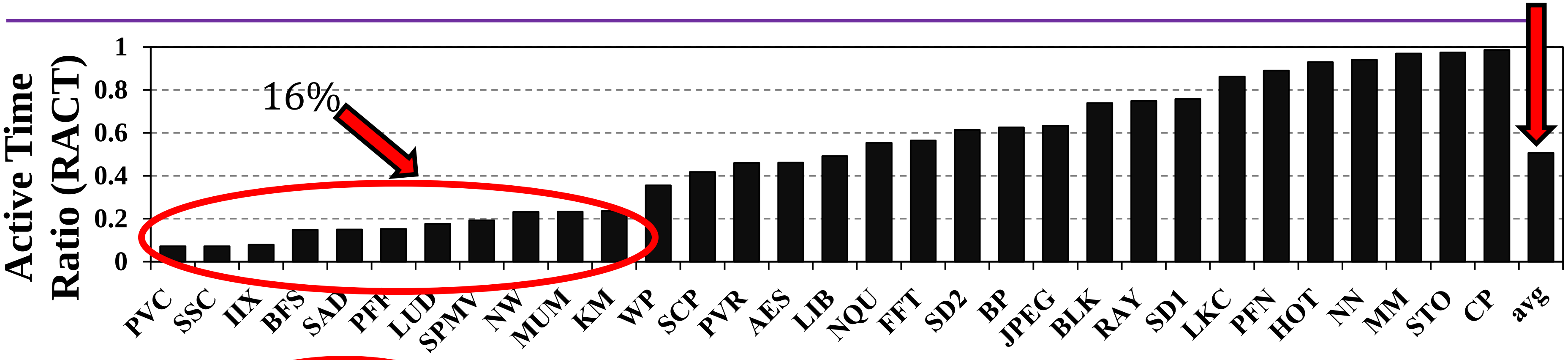
- Proposal
- Background
- Motivation
- DYNCTA
- Evaluation
- Conclusions

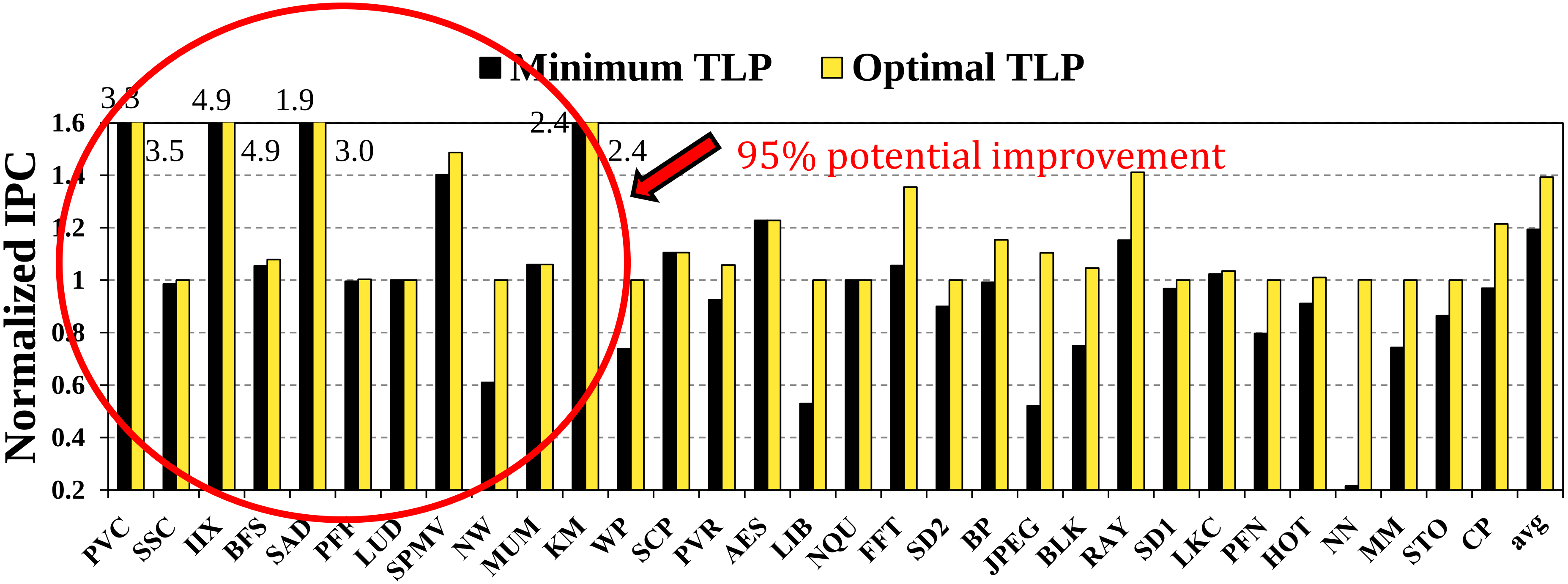
Effect of TLP on GPGPU Performance



Effect of TLP on GPGPU Performance

51%

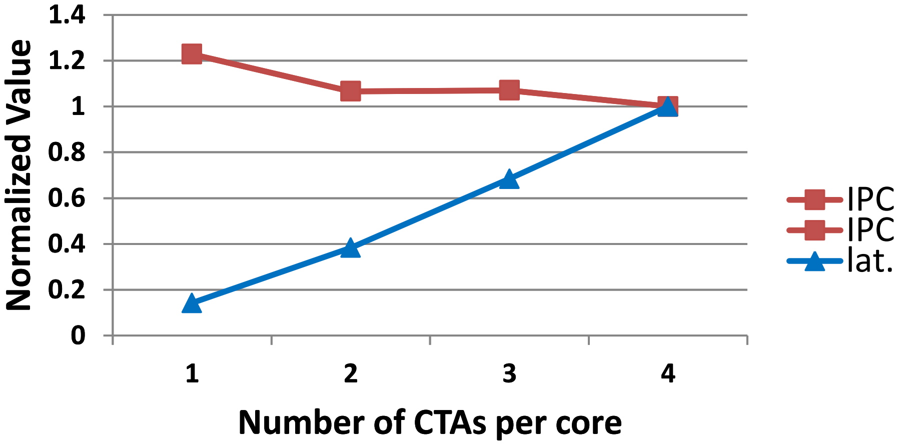




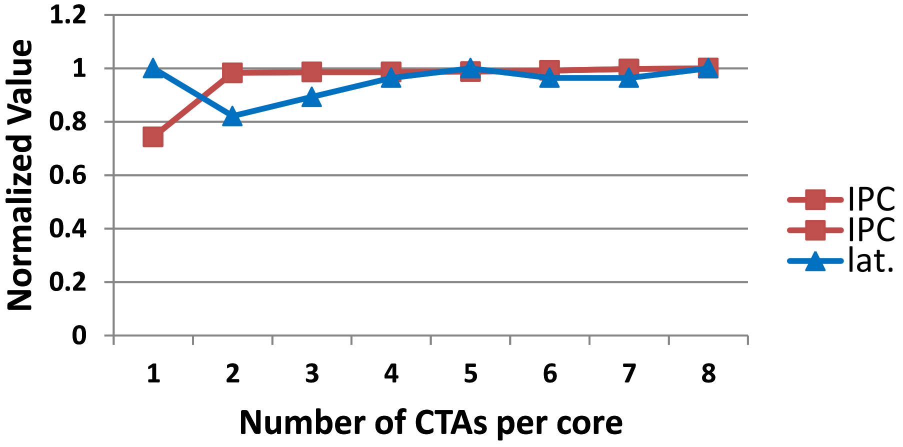
95% potential improvement

Why is not more TLP always optimal?

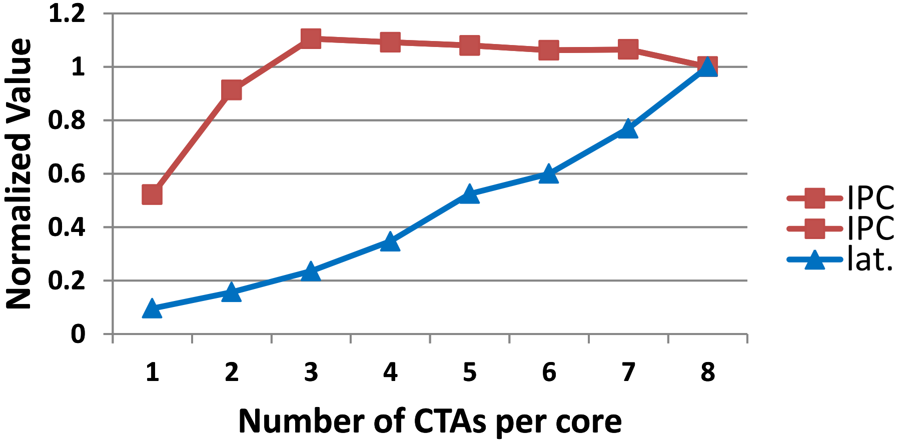
AES



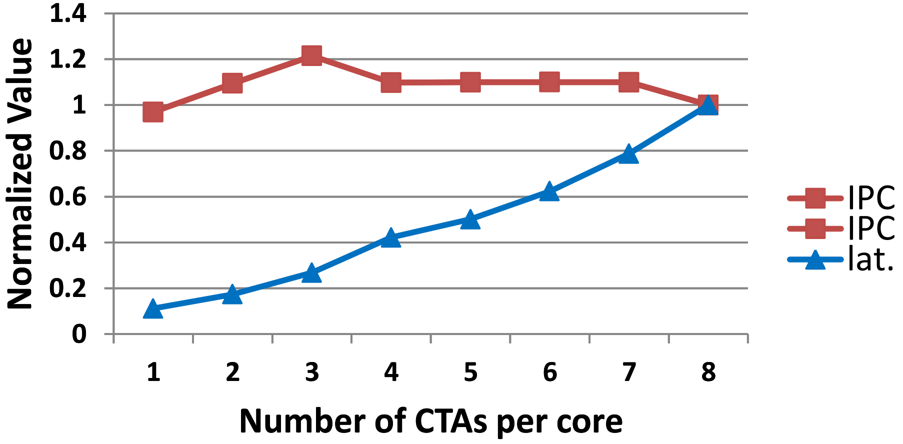
MM



JPEG

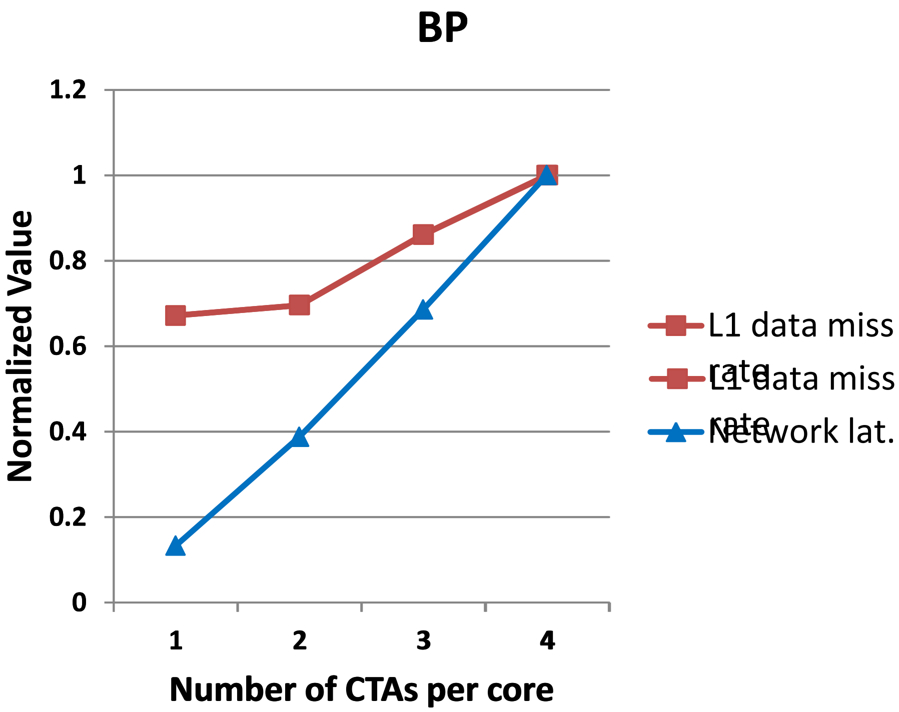


CP



Why is not more TLP always optimal?

- More threads result in larger working data set
  - Causes cache contention
- More L1 misses cause more network injections
  - Network latency increases



Outline

- Proposal
- Background
- Motivation
- DYNCTA
- Evaluation
- Conclusions

DYNCTA Approach

- Execute the optimal number of CTAs for each application
- Requires exhaustive analysis for each application, thus inapplicable

Idea:
Dynamically modulate the number of CTAs on each core using the CTA scheduler

DYNCTA Approach

- Objective 1: keep the cores busy
  - If a core has nothing to execute, give more threads to it

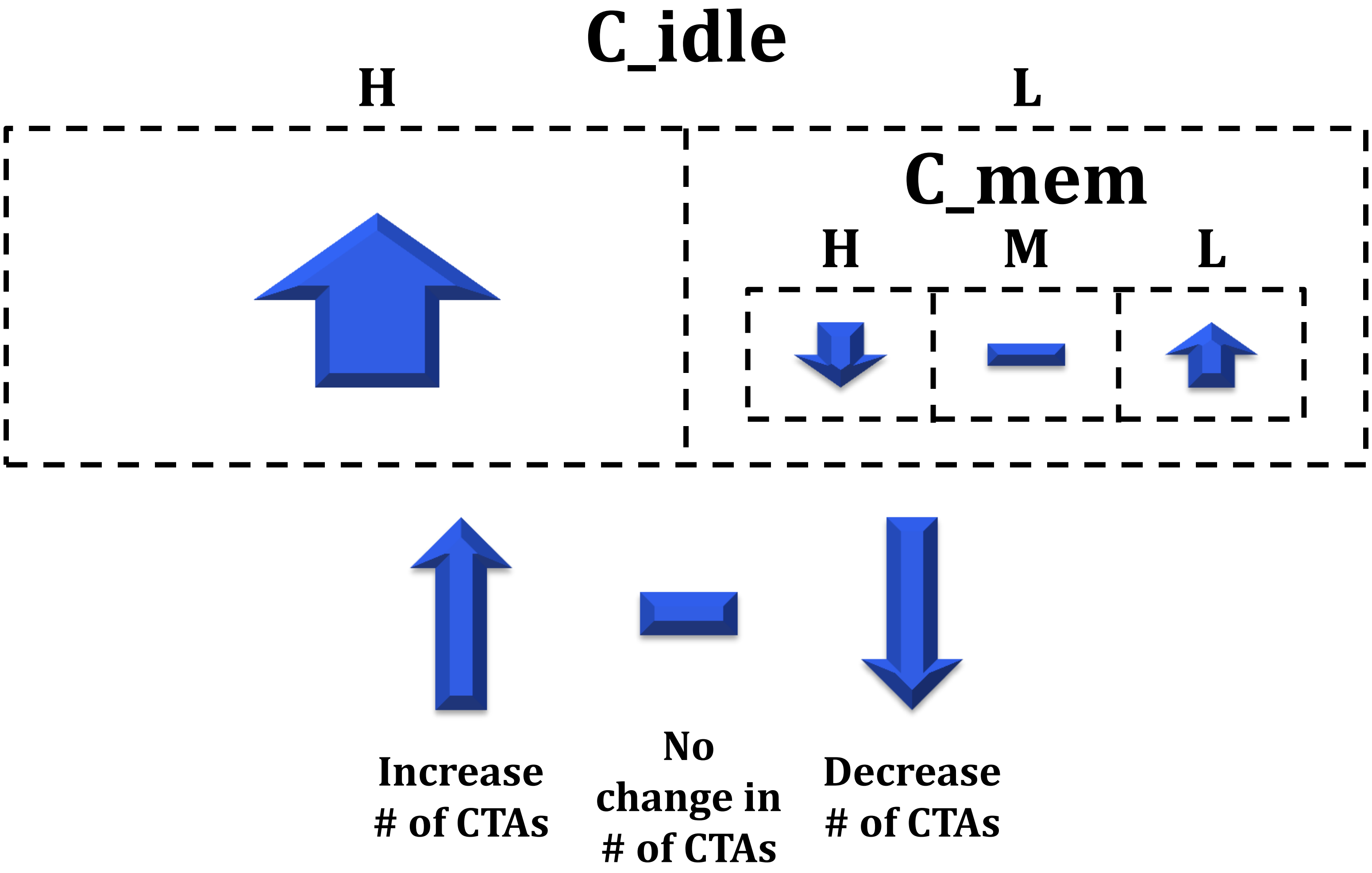
- Objective 2: do not keep the cores TOO BUSY
  - If the memory sub-system is congested due to high number of threads, lower TLP to reduce contention
  - If the memory sub-system is not congested, increase TLP to improve latency tolerance

DYNCTA Approach

- Objective 1: keep the cores busy
  - Monitor C_idle, the number of cycles during which a core does not have anything to execute
  - If it is high, increase the number of CTAs executing on the core

- Objective 2: do not keep the cores TOO BUSY
  - Monitor C_mem, the number of cycles during which a core is waiting for the data to come back from memory
  - If it is low, increase the number of CTAs executing on the core
  - If it is high, decrease the number of CTAs executing on the core

DYNCTA Overview



C_idle

H

L

C_mem

H

M

L

Increase
# of CTAs

No
change in
# of CTAs

Decrease
# of CTAs

CTA Pausing

Once assigned to a core, a CTA cannot be preempted!

Then, how to decrement the number of CTAs ?

CTA

Warps of the most recently assigned CTA are deprioritized in the warp scheduler

Outline

- Proposal
- Background
- Motivation
- DYNCTA
- Evaluation
- Conclusions

Evaluation Methodology

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

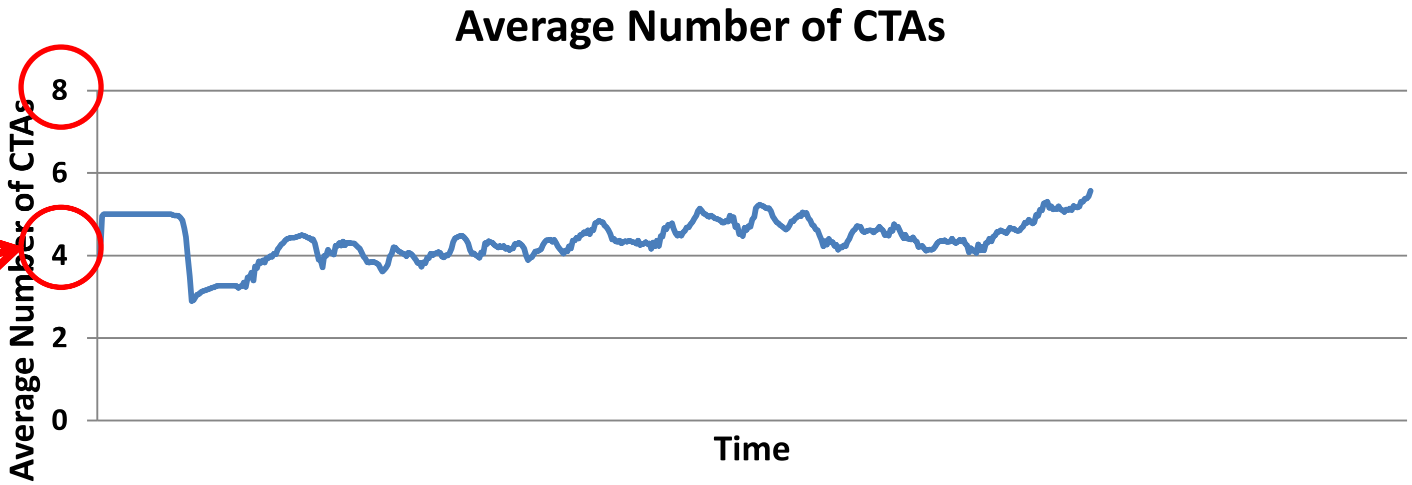
- Baseline Architecture
  - 30 SIMT cores, 8 memory controllers, crossbar connected
  - 1300MHz, SIMT Width = 8, Max. 1024 threads/core
  - 32 KB L1 data cache, 8 KB Texture and Constant Caches
  - GDDR3 800MHz

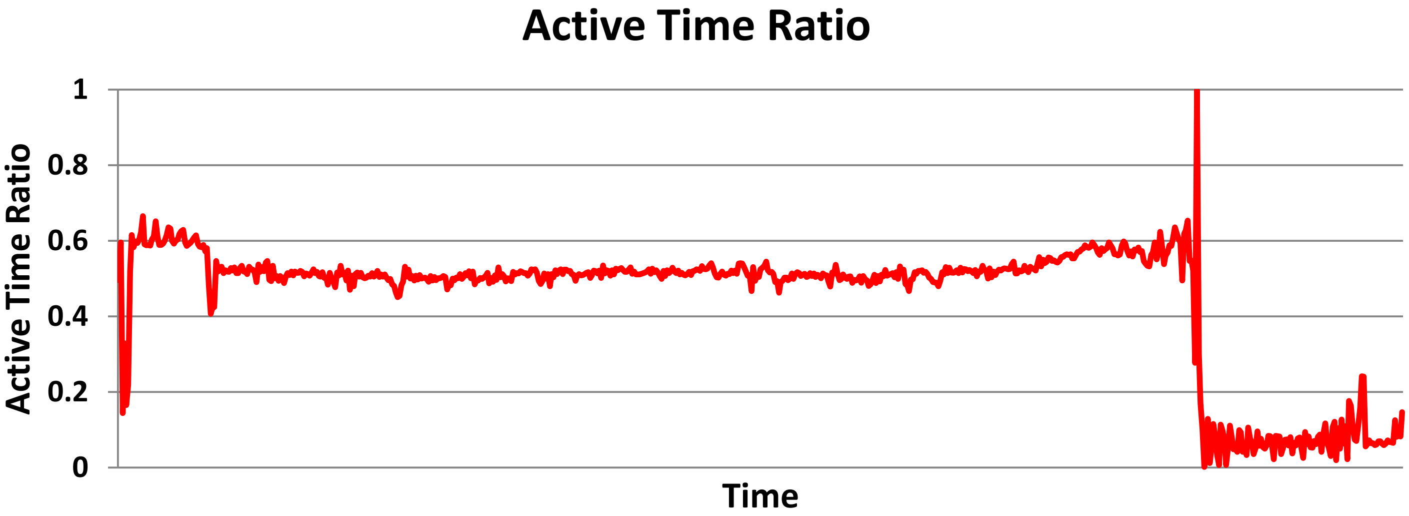
- Applications Considered (in total 31) from:
  - Map Reduce Applications
  - Rodinia – Heterogeneous Applications
  - Parboil – Throughput Computing Focused Applications
  - NVIDIA CUDA SDK – GPGPU Applications

Dynamism

Default number of CTAs

Optimal number of CTAs

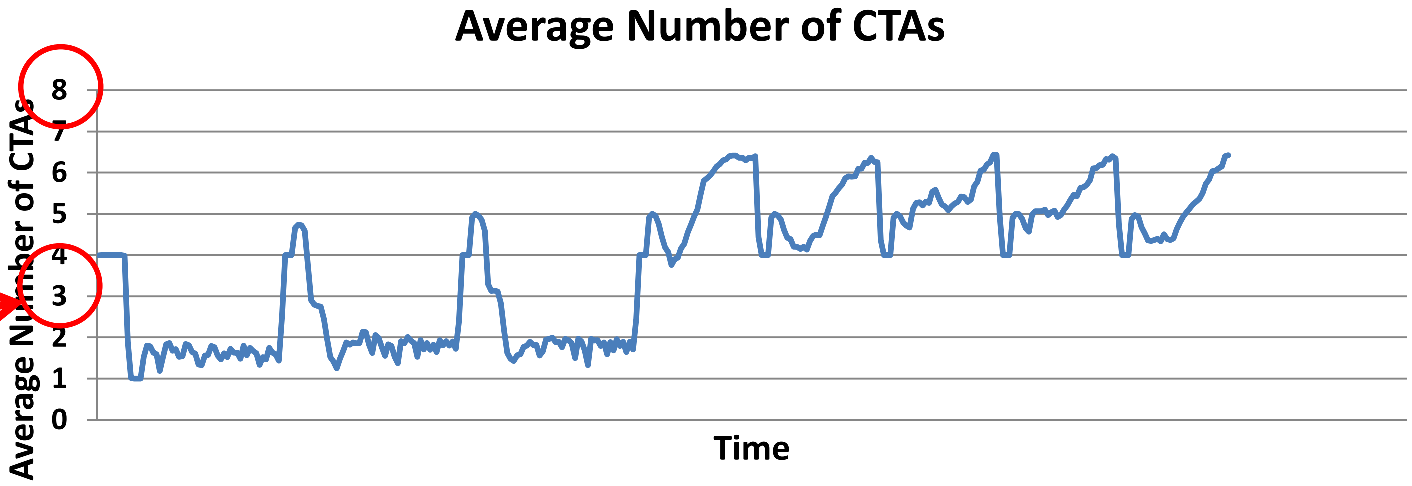


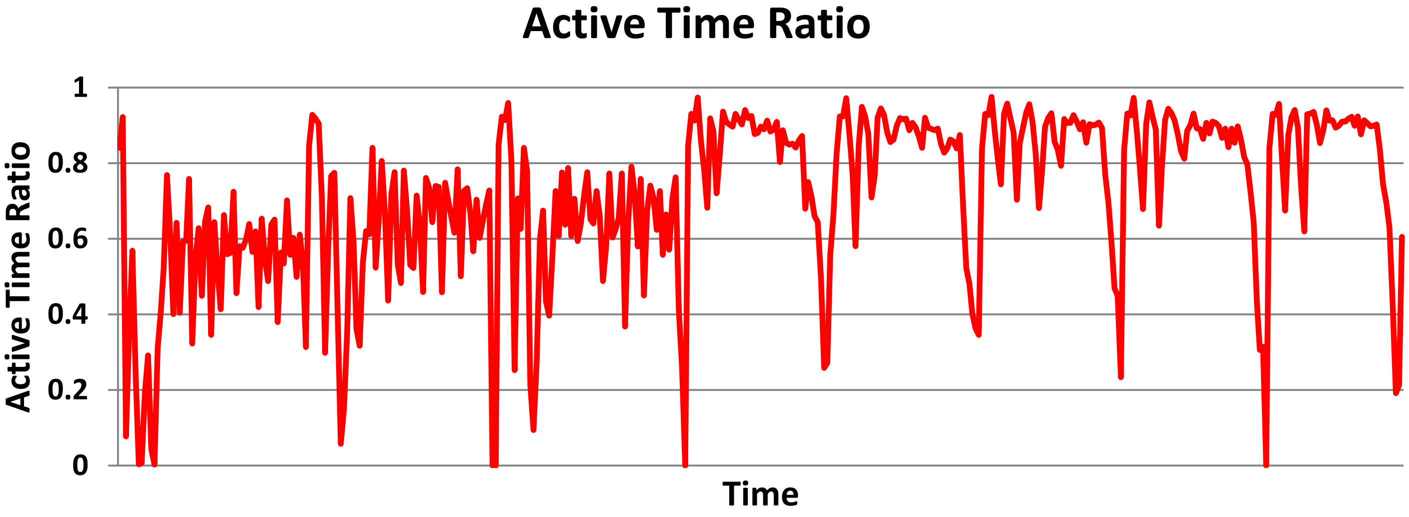


Dynamism

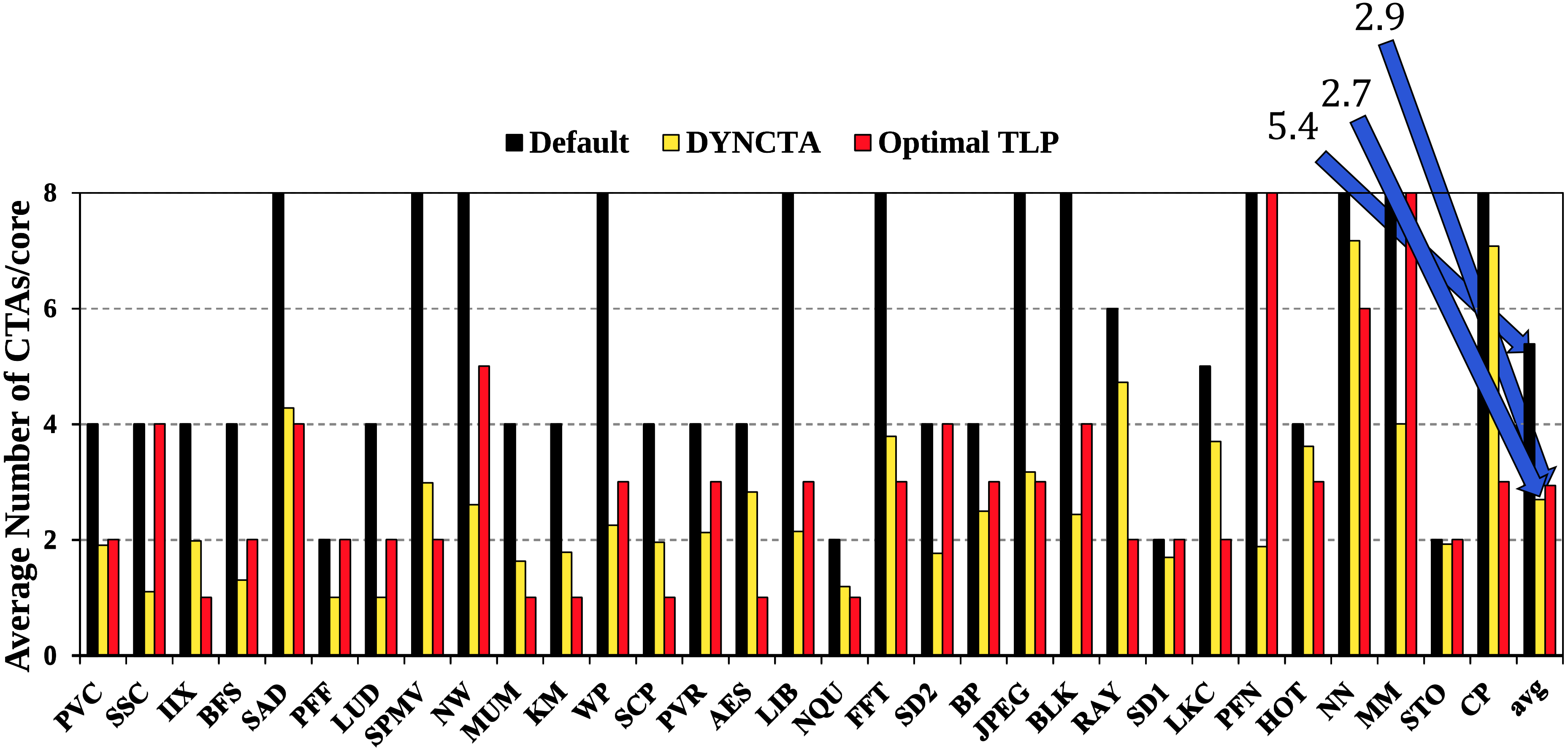
Default number of CTAs

Optimal number of CTAs

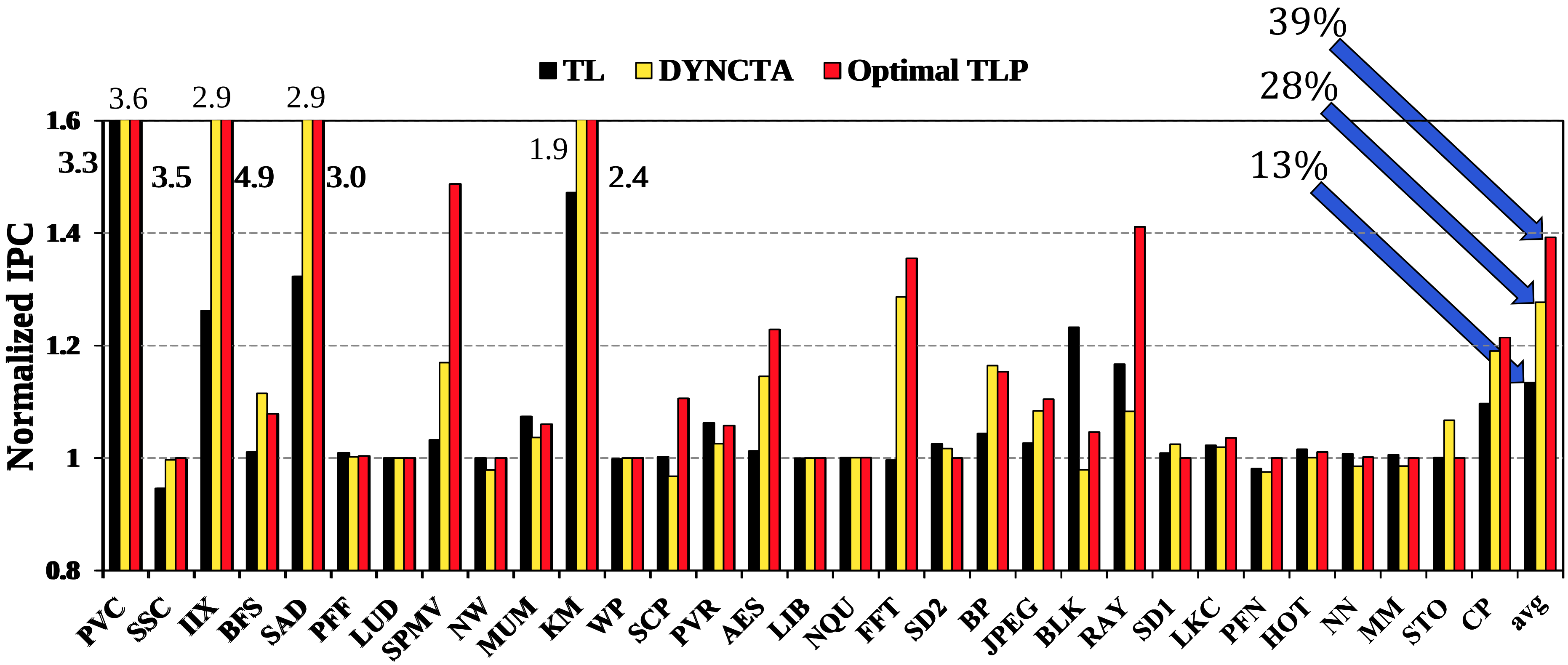




Average Number of CTAs



IPC



39%
28%
13%
■ TL
🟨 DYNCTA
🟥 Optimal TLP
Normalized IPC
1.6
1.4
1.2
1
0.8
3.6
2.9
2.9
3.3
3.5
4.9
3.0
1.9
2.4
PVC
SSC
IIX
BFS
SAD
PFF
LUD
SPMV
NW
MUM
KM
WP
SCP
PVR
AES
LIB
NQU
FFT
SD2
BP
JPEG
BLK
RAY
SD1
LKC
PFN
HOT
NN
MM
STO
CP
avg

Outline

- Proposal
- Background
- Motivation
- DYNCTA
- Evaluation
- Conclusions

Conclusions

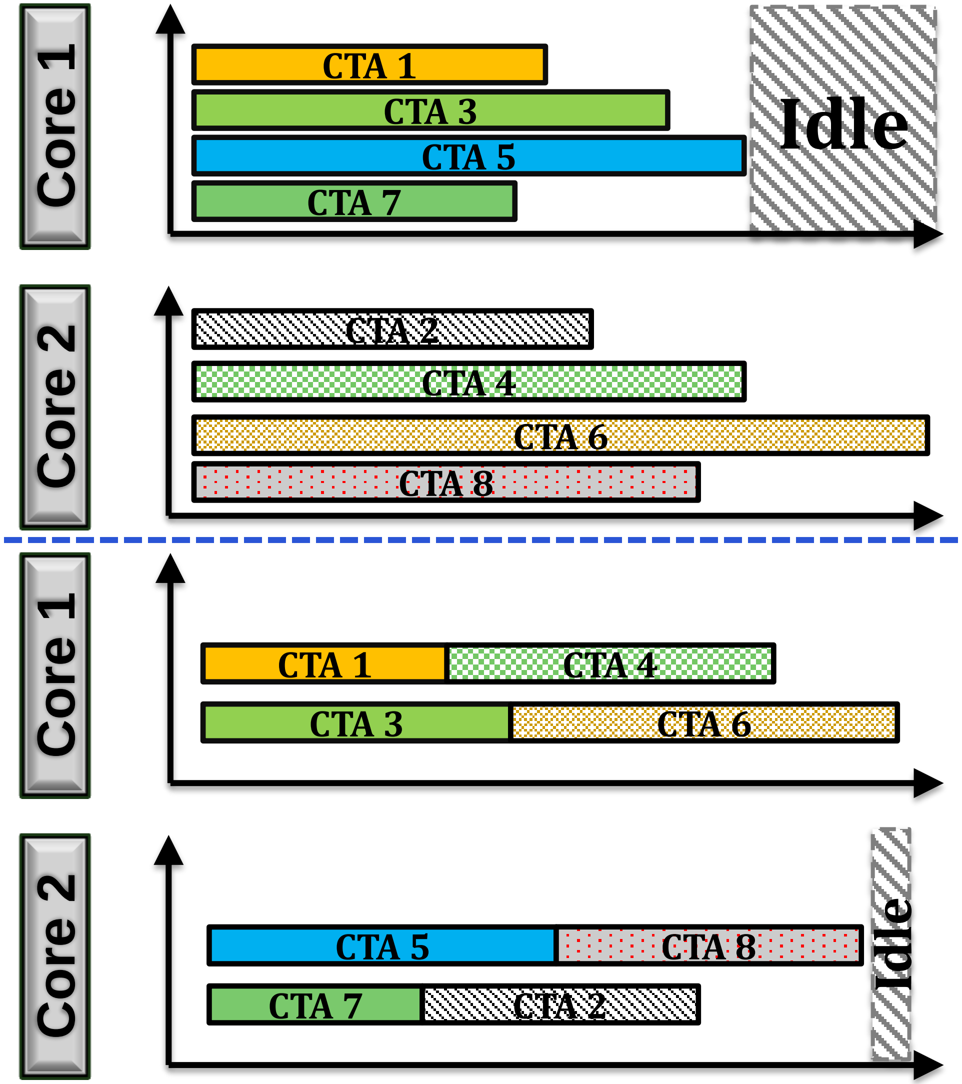
- Maximizing TLP is not always optimal in terms of performance
- We propose a CTA scheduling algorithm, DYNCTA, that optimizes TLP at the cores based on application characteristics
- DYNCTA reduces cache, network and memory contention
- DYNCTA improves average application performance by 28%

THANKS!

QUESTIONS?

BACKUP

Utilization



Initial n

- All cores are initialized with $\lfloor N/2 \rfloor$ CTAs.
- Starting with 1 CTAs and $\lfloor N/2 \rfloor$ CTAs usually converge to the same value.
- Starting with the default number of CTAs might not be as effective

Comparison against optimal CTA count

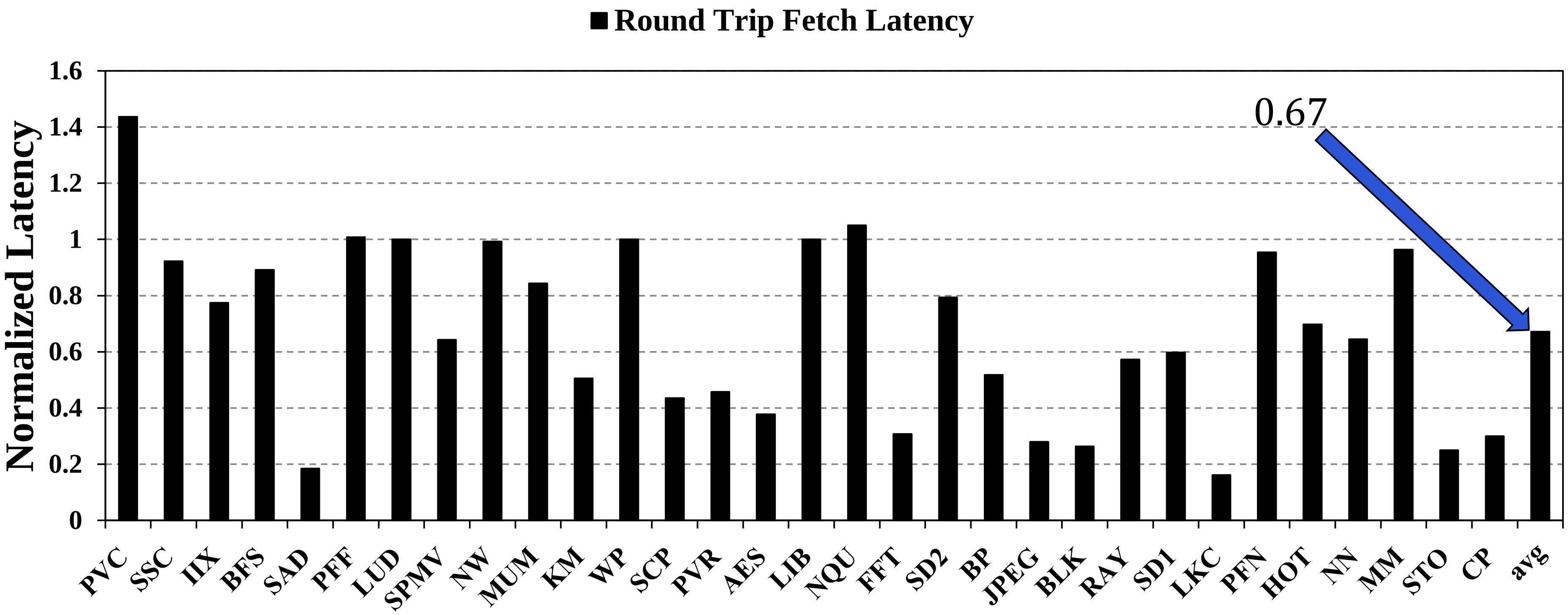
- Optimal number of CTAs might be different for different intervals for applications that exhibit compute- and memory-intensive behaviors at different intervals

- Our algorithm outperforms optimal number of CTAs in some applications

Parameters

Variable	Description	Value
Nact	Active time, where cores can fetch new warps	
Ninact	Inactive time, where cores cannot fetch new warps	
RACT	Active time ratio, Nact/(Nact + Ninact)	
C_idle	The number of core cycles during which the pipeline is not stalled, but there are no threads to execute	
C_mem	The number of core cycles during which all the warps are waiting for their data to come back	
t_idle	Threshold that determines whether C_idle is low or high	16
t_mem_l & t_mem_h	Thresholds that determine if C_mem is low, medium or high	128 & 384
Sampling period	The number of cycles to make a modulation decision	2048

Round Trip Fetch Latency



Other Metrics

- L1 data miss rate: 71% → 64%
- Network latency: ↓ 33%
- Active time ratio: ↑ 14%

Sensitivity

- Large system with 56 and 110 cores: around 20% performance improvement
- MSHR size: 64 – 32 – 16: 0.3% and 0.6% performance loss
- DRAM frequency: 1333 MHz: 1% performance loss
- Sampling period 2048 – 4096: 0.1% performance loss
- Thresholds: 50% - 150% of the default values: losses between 0.7% - 1.6%