LaPerm: Locality Aware Scheduler for Dynamic Parallelism on GPUs

Jin Wang*, Norm Rubin†, Albert Sidelnik†, Sudhakar Yalamanchili*

*Computer Architecture and System Lab, Georgia Institute of Technology

†NVIDIA Research

Sponsors:

Executive Summary

- Motivation: New memory reference locality relationship in dynamic parallelism
- Problem: State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache
- Proposed: LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions

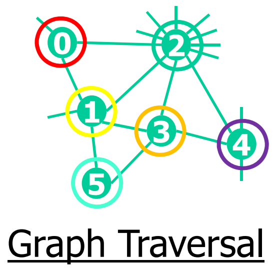
Dynamic Parallelism on GPU

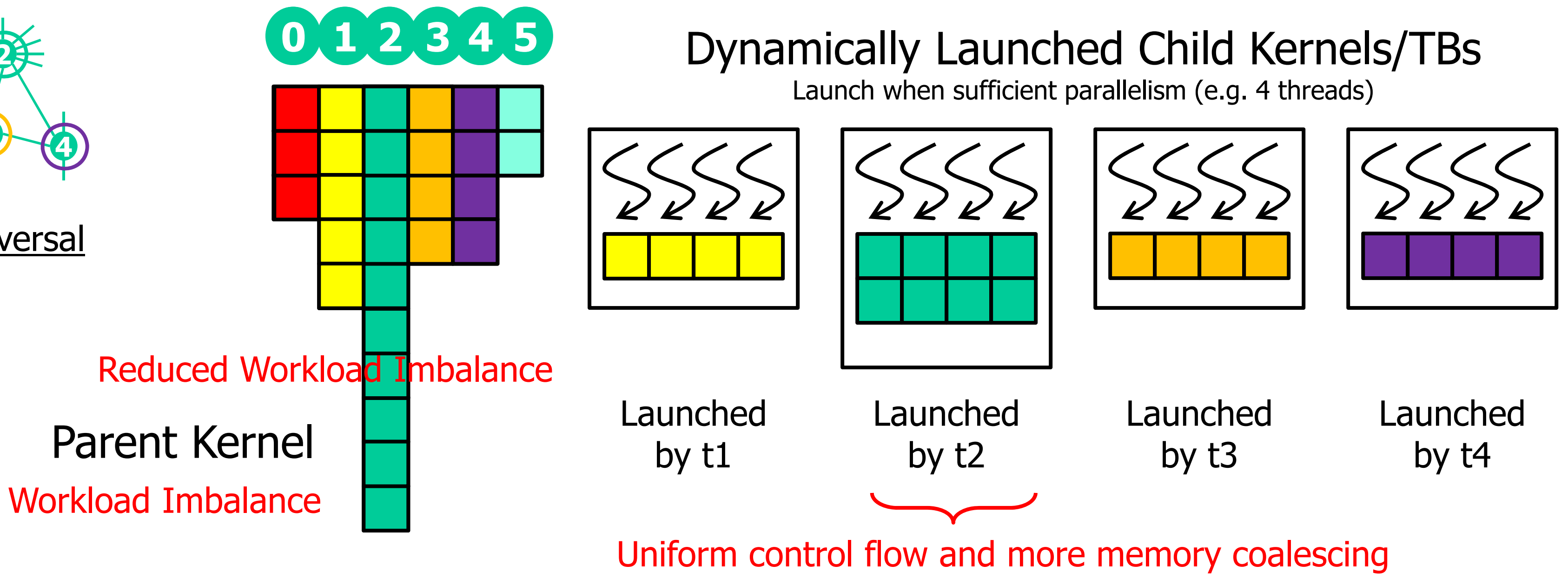
- Launch workload on demand from GPU
  - CUDA Dynamic Parallelism (CDP)
  - OpenCL device-side enqueue
  - Dynamic Thread Block Launch[1] (DTBL)

} Launch new kernels

} Launch new thread blocks

- Benefits
  - Apply to fine-grained parallelism in irregular applications
  - Increase execution efficiency and productivity

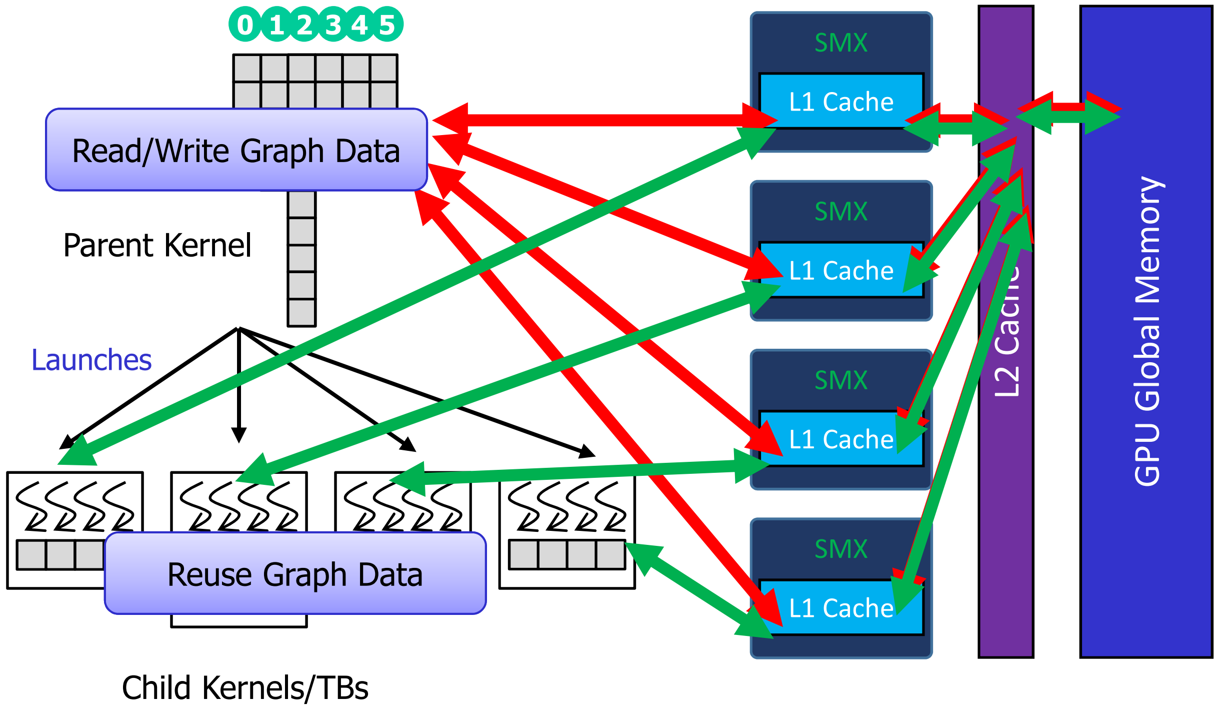



0 1 2 3 4 5
Reduced Workload Imbalance
Parent Kernel
Workload Imbalance
Dynamically Launched Child Kernels/TBs
Launch when sufficient parallelism (e.g. 4 threads)
Launched by t1
Launched by t2
Launched by t3
Launched by t4
Uniform control flow and more memory coalescing

[1] Wang, Rubin, Sidelnik and Yalamanchili, "Dynamic thread block launch: A lightweight execution mechanism to support irregular applications on gpus", ISCA 2015

Memory Locality in Dynamic Parallelism

- New ***data reference locality relationship*** in parent-child launching
- Potential Locality:
  - Parent-child and child-sibling data sharing
  - L1/L2 locality



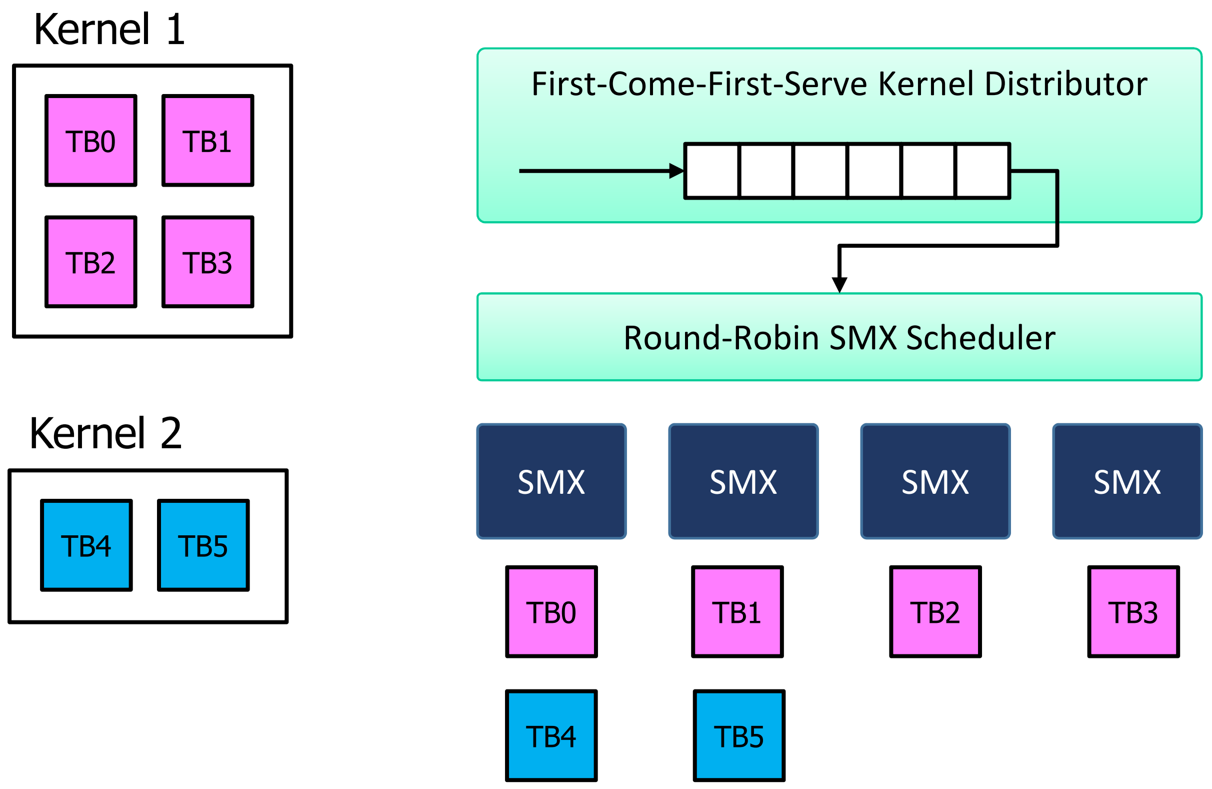
- Average shared footprint ratio for 8 benchmarks:
  - Parent-child: 38.4%
  - Child-sibling: 30.5%

Executive Summary

- Motivation: New memory reference locality relationship in dynamic parallelism
- Problem: State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache
- Proposed: LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions

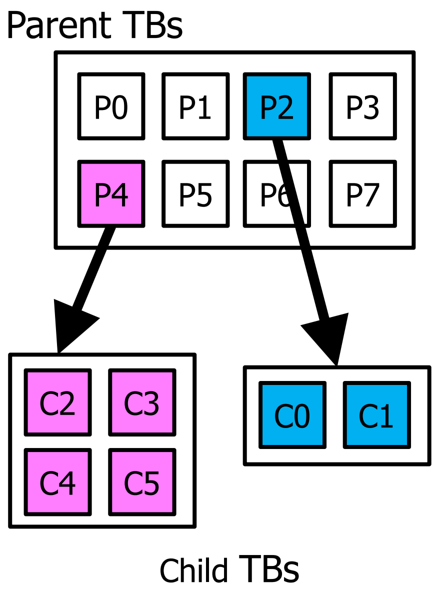
Current GPU Scheduler

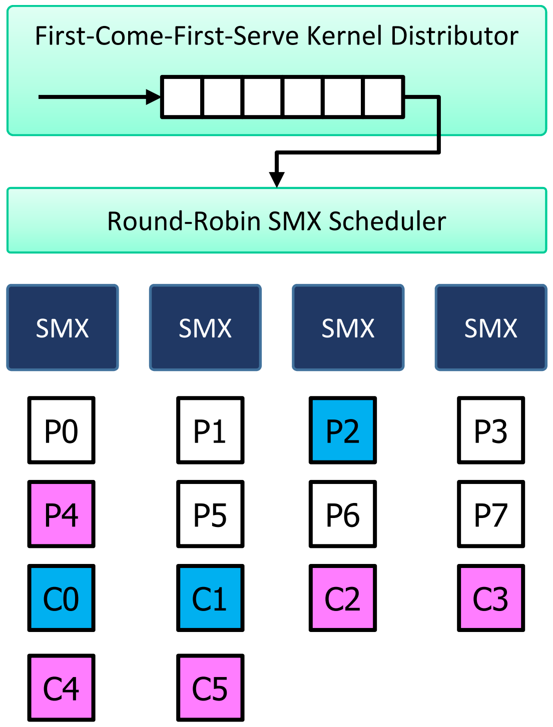
- First-Come-First-Serve kernel scheduler
- Round-Robin TB Scheduler


Kernel 1
TB0
TB1
TB2
TB3
Kernel 2
TB4
TB5
First-Come-First-Serve Kernel Distributor
Round-Robin SMX Scheduler
SMX
SMX
SMX
SMX
TB0
TB1
TB2
TB3
TB4
TB5

Current GPU Scheduler (continue)

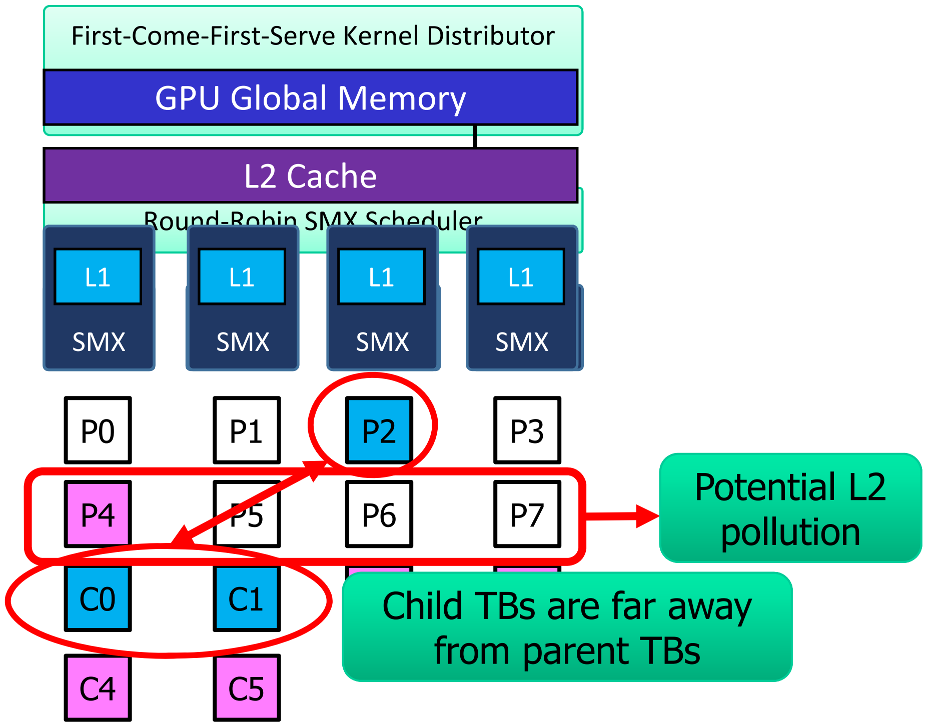
- For non-Dynamic Parallelism scenario
  - Fairness and efficiency
- For Dynamic Parallelism scenario
  - Child TBs are scheduled after parent TBs




First-Come-First-Serve Kernel Distributor
Round-Robin SMX Scheduler
SMX
SMX
SMX
SMX
P0
P1
P2
P3
P4
P5
P6
P7
C0
C1
C2
C3
C4
C5

Current GPU Scheduler: Issues

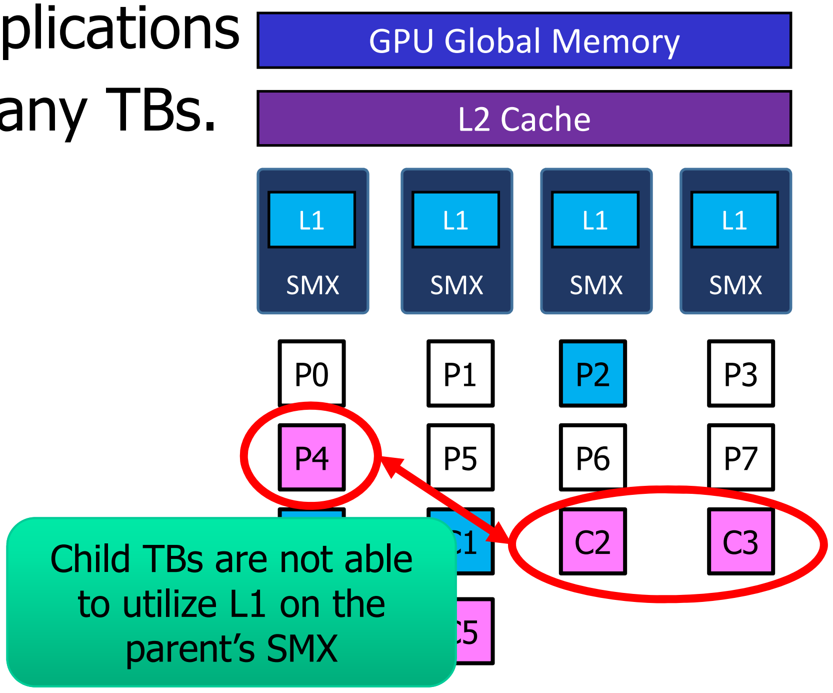
- Issue 1: Child TBs are executed far later than the parent TBs, decreasing L2 locality



First-Come-First-Serve Kernel Distributor
GPU Global Memory
L2 Cache
Round-Robin SMX Scheduler
L1
SMX
L1
SMX
L1
SMX
L1
SMX
P0
P1
P2
P3
P4
P5
P6
P7
C0
C1
C4
C5
Potential L2 pollution
Child TBs are far away from parent TBs

Current GPU Scheduler: Issues (continue)

- Issue 1: Child TBs are executed far later than the parent TBs, decreasing L2 locality
- Issue 2: Child TBs are executed on a different SMX than its parent, decreasing L1 locality
- <u>Fails to exploit parent-child or child-sibling locality</u>
  - Exacerbated in real applications which generally have many TBs.



Executive Summary

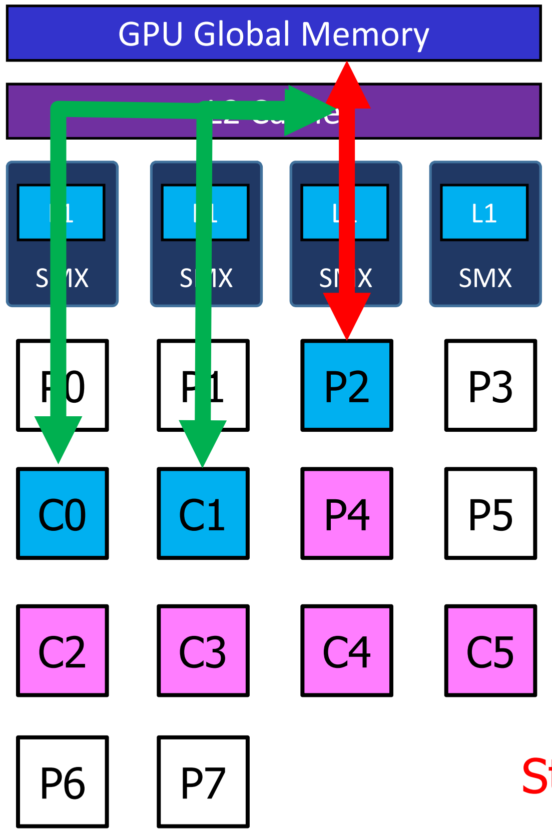
- Motivation: New memory reference locality relationship in dynamic parallelism
- Problem: State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache
- Proposed: LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions

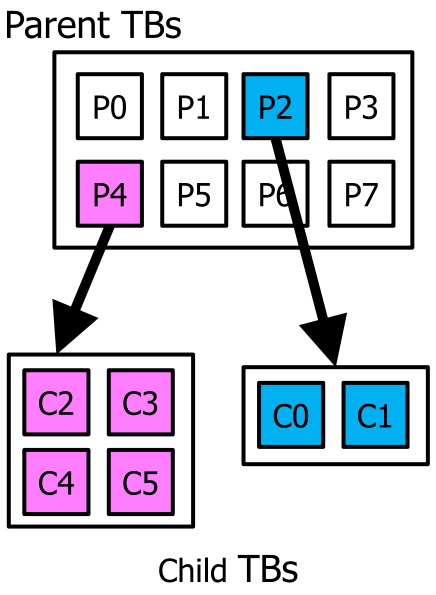
LaPerm: Locality-Aware Scheduler for Dynamic Parallelism

- Leverage locality between parent and child TBs
- Three scheduling decisions
  - Accommodate different forms of locality
- Goal: improve memory efficiency and overall performance

Scheduling Decision 1: TB Prioritizing

- Prioritize child TBs to be executed immediately after direct parent TBs
- Reuse parent data and avoid L2 cache pollution

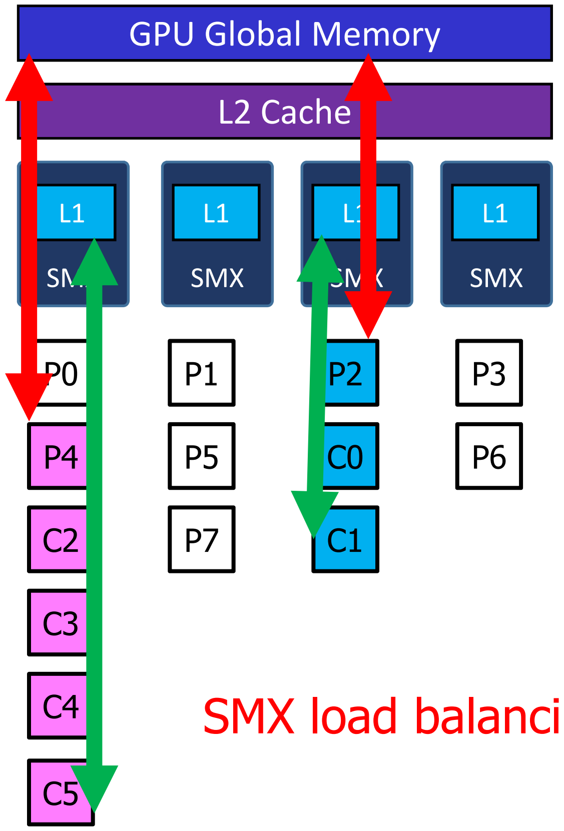




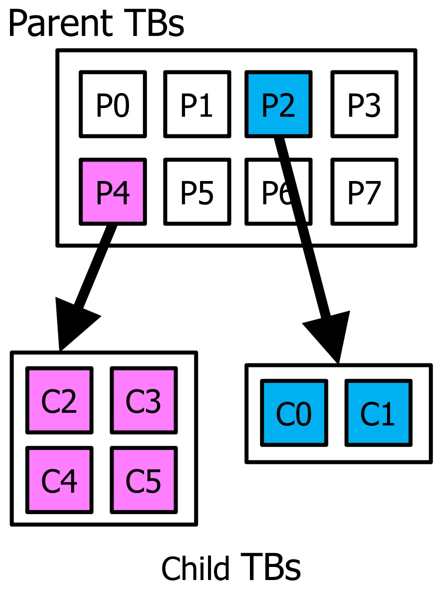
Still no L1 locality!

Scheduling Decision 2: Prioritized SMX Binding

- Bind the prioritized child TBs on the same SMX as the parent TBs
- Utilize L1 cache for data reuse

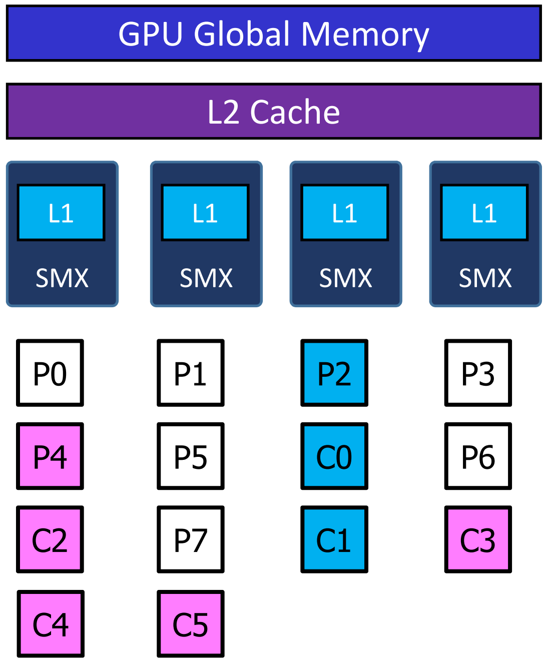


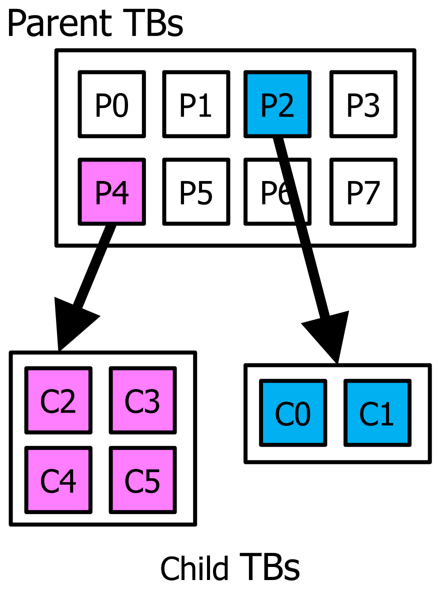
SMX load balancing issue!



Scheduling Decision 3: Adaptive Prioritized SMX Binding

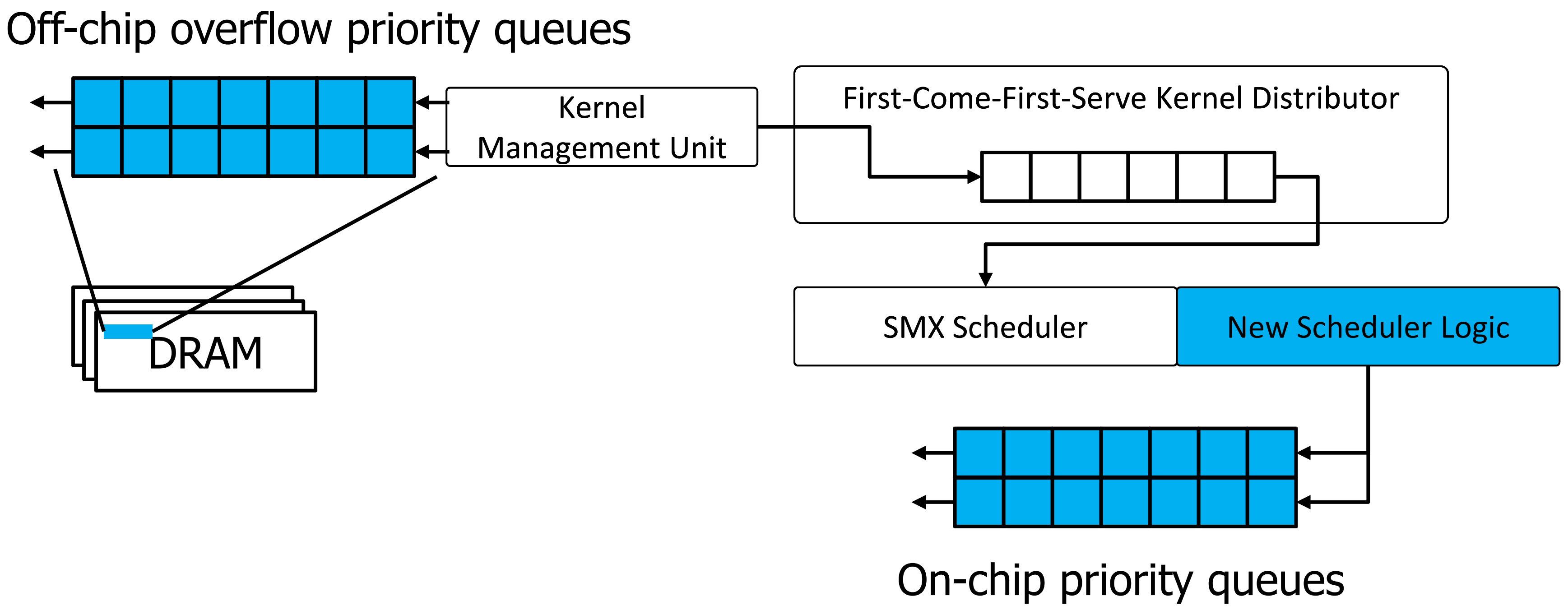
- Adaptively bind child TBs on available SMXs
- Avoid SMX load balancing caused by SMX binding





Architecture Support

- Multi-level priority queues
  - Used to store new TB/Kernel information
  - Ordered using priority value
  - Divided among multiple SMXs



Benchmark and Experimental Environment

- Benchmark implemented with dynamic parallelism
- Simulated on GPGPU-Sim

Benchmark	Inpu	Notation
Adaptive Mesh Refinement	Ener	amr
Barnes Hut Tree	Random data set	bht
Breadth-First Search	Citatio	bfs_citation
	USA r	bfs_usa_road
	Cage1	bfs_cage15
Graph Coloring	Citation network	clr_citation
	Graph 500	clr_graph500
	Cage15 sparse matrix	clr_cage15
Regular Expression Match	Darpa network	regx_darpa
	Rando	regx_string
Product Recommendation	Movie	pre
Relational Join	Uniform distribution synthetic	join_uniform
	Gaussian distribution synthetic	join_gaussian
Single-Source Shortest Path	Citatio	sssp_citation
	Fight network	sssp_flight
	Cage15 sparse matrix	sssp_cage15

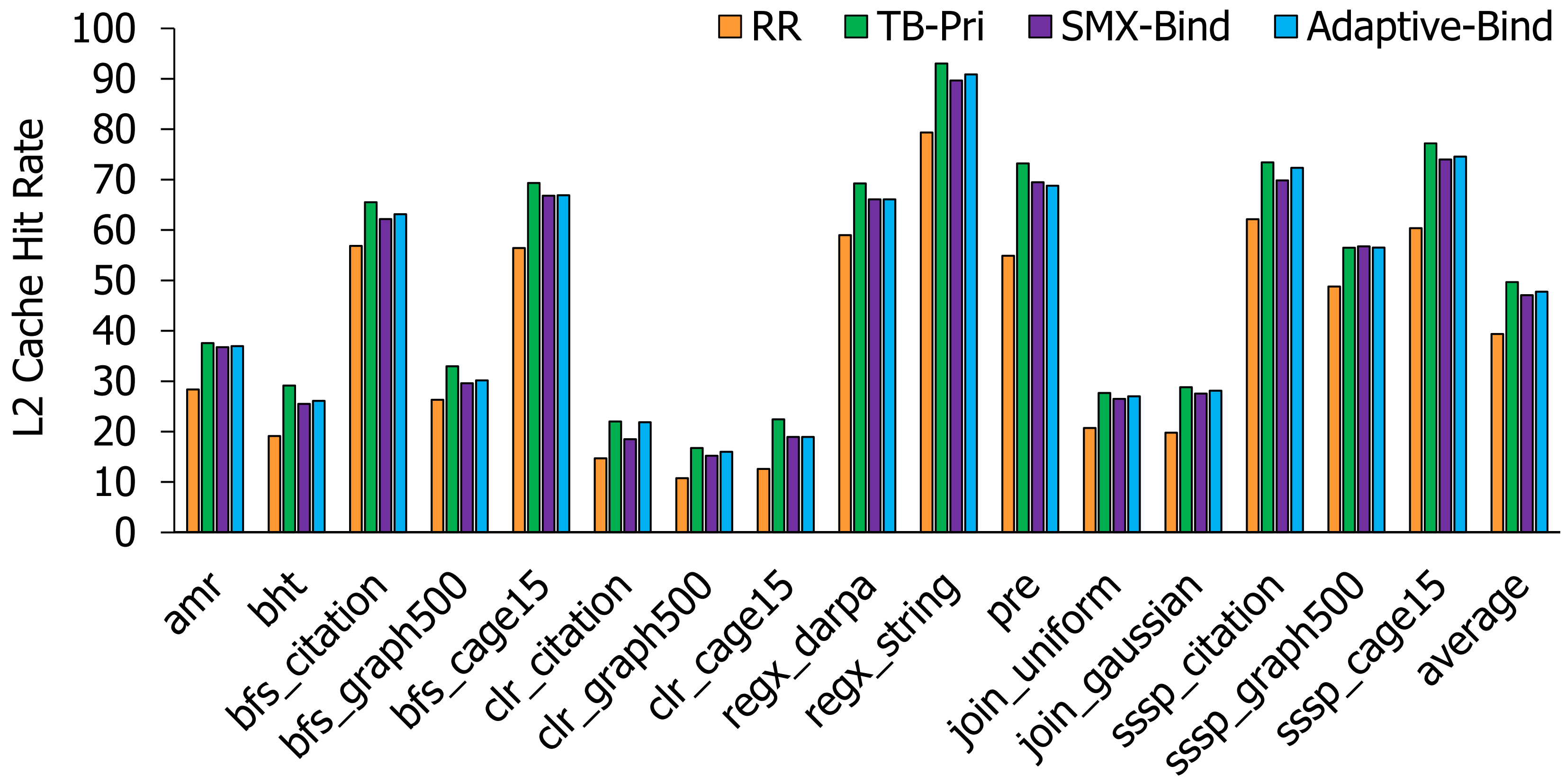
Physics Simulation

Tree and Graph Applications

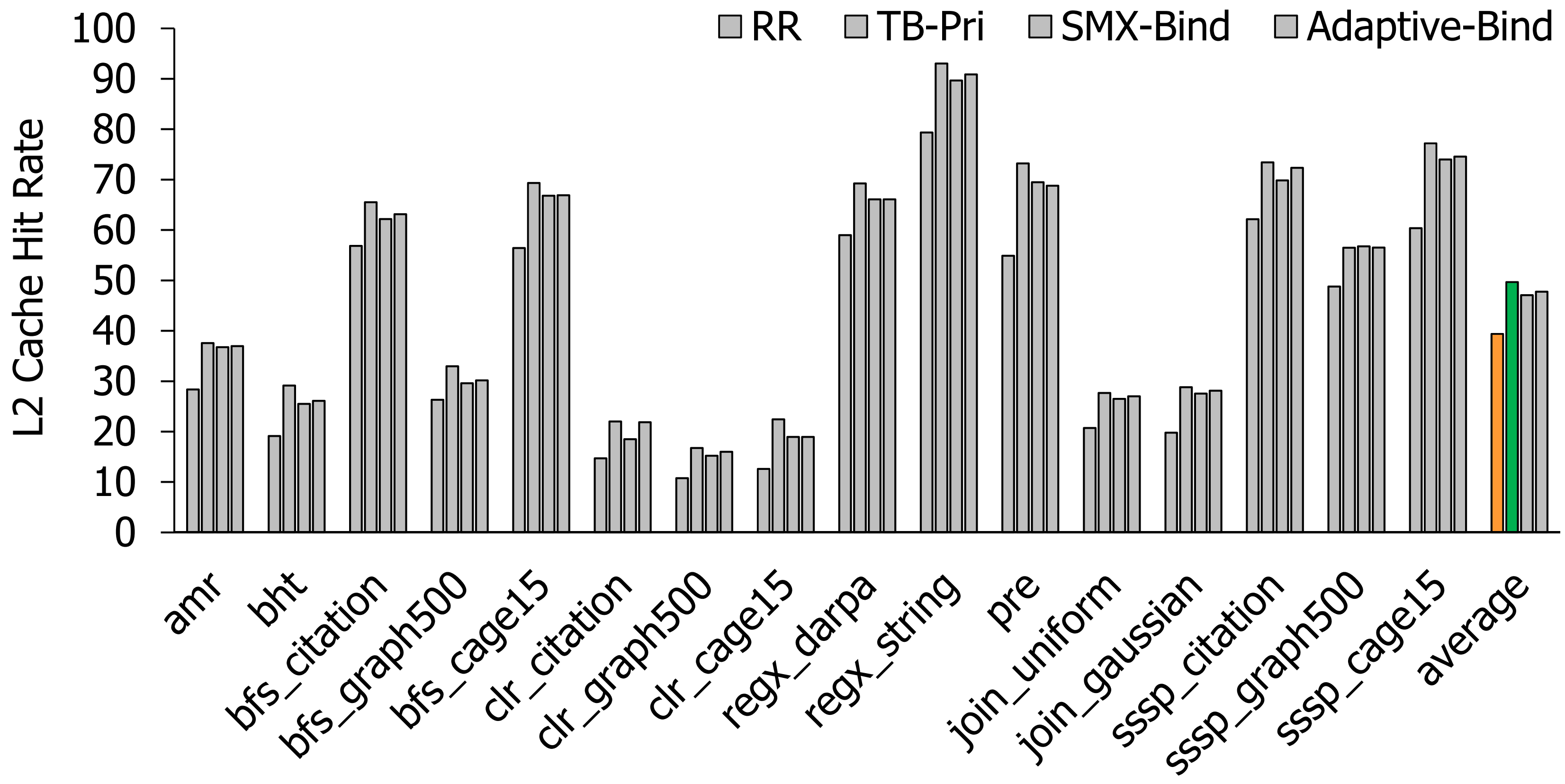
Machine Learning

Relational Database

Performance of LaPerm: L2 Cache

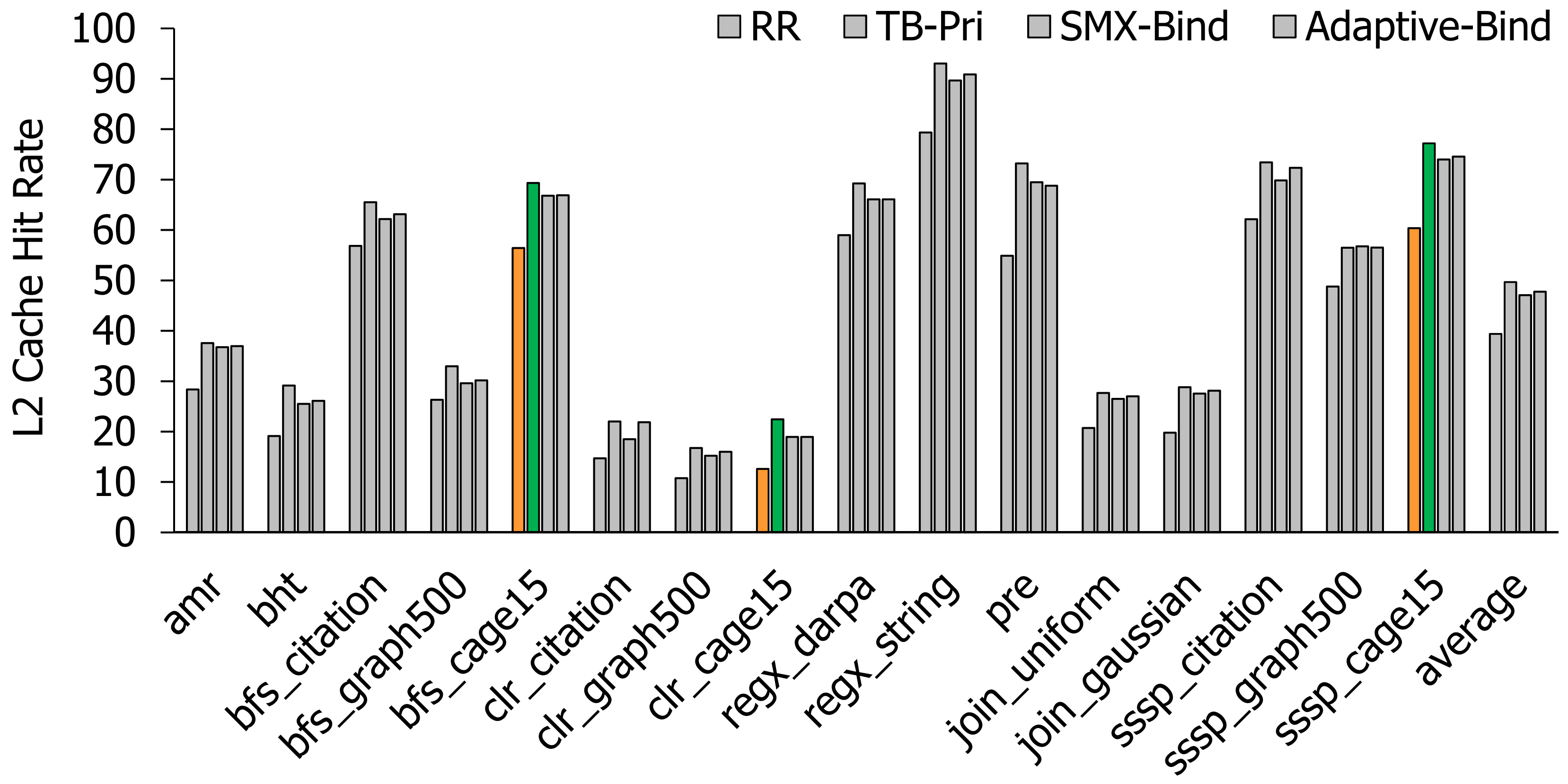


Performance of LaPerm: L2 Cache



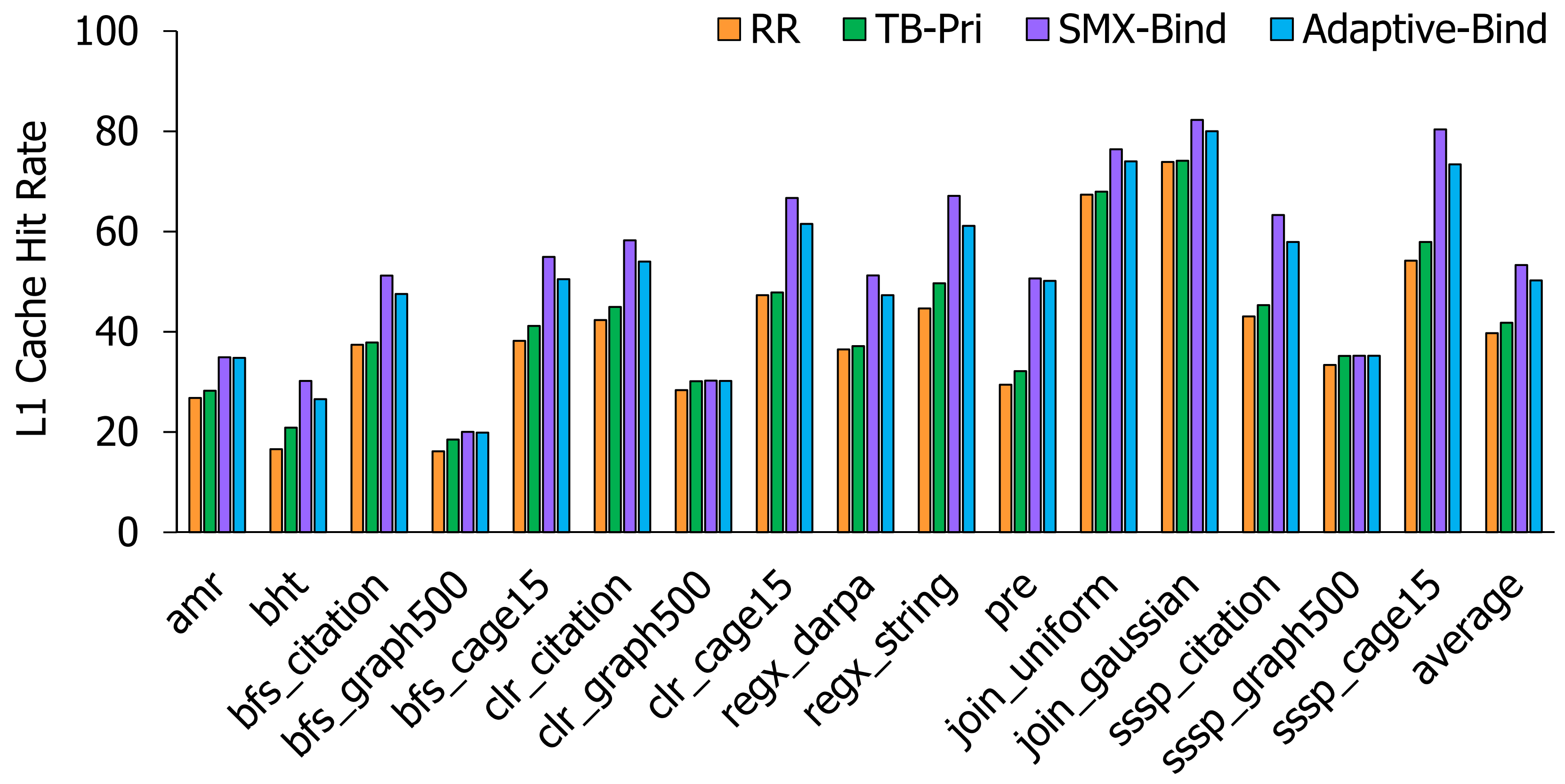
- L2 Cache hit rate benefits mainly from prioritizing child TBs

Performance of LaPerm: L2 Cache

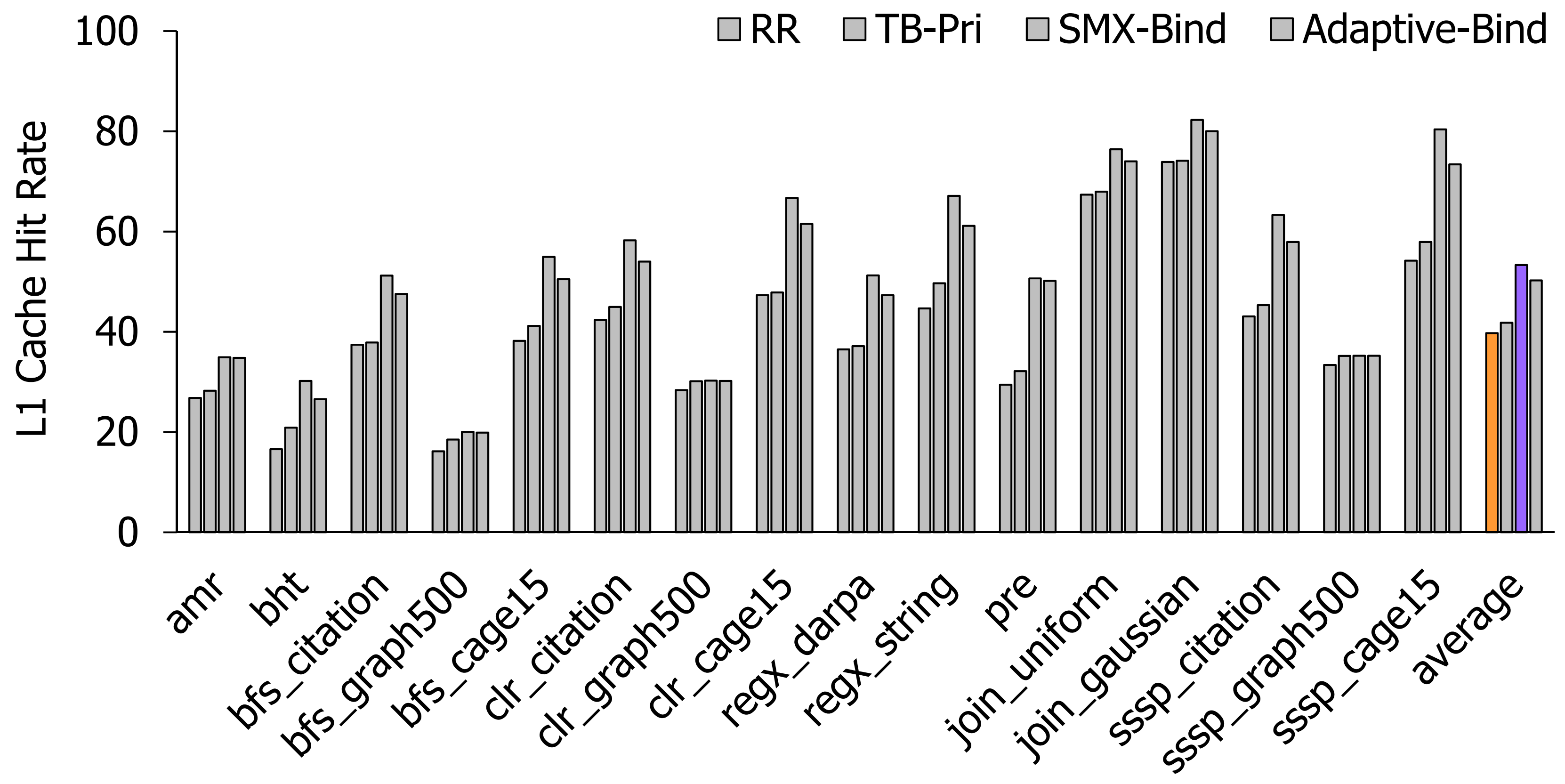


- L2 Cache hit rate benefits mainly from prioritizing child TBs
- Graph applications with *cage15* input have more parent-child data reuse

Performance of LaPerm: L1 Cache

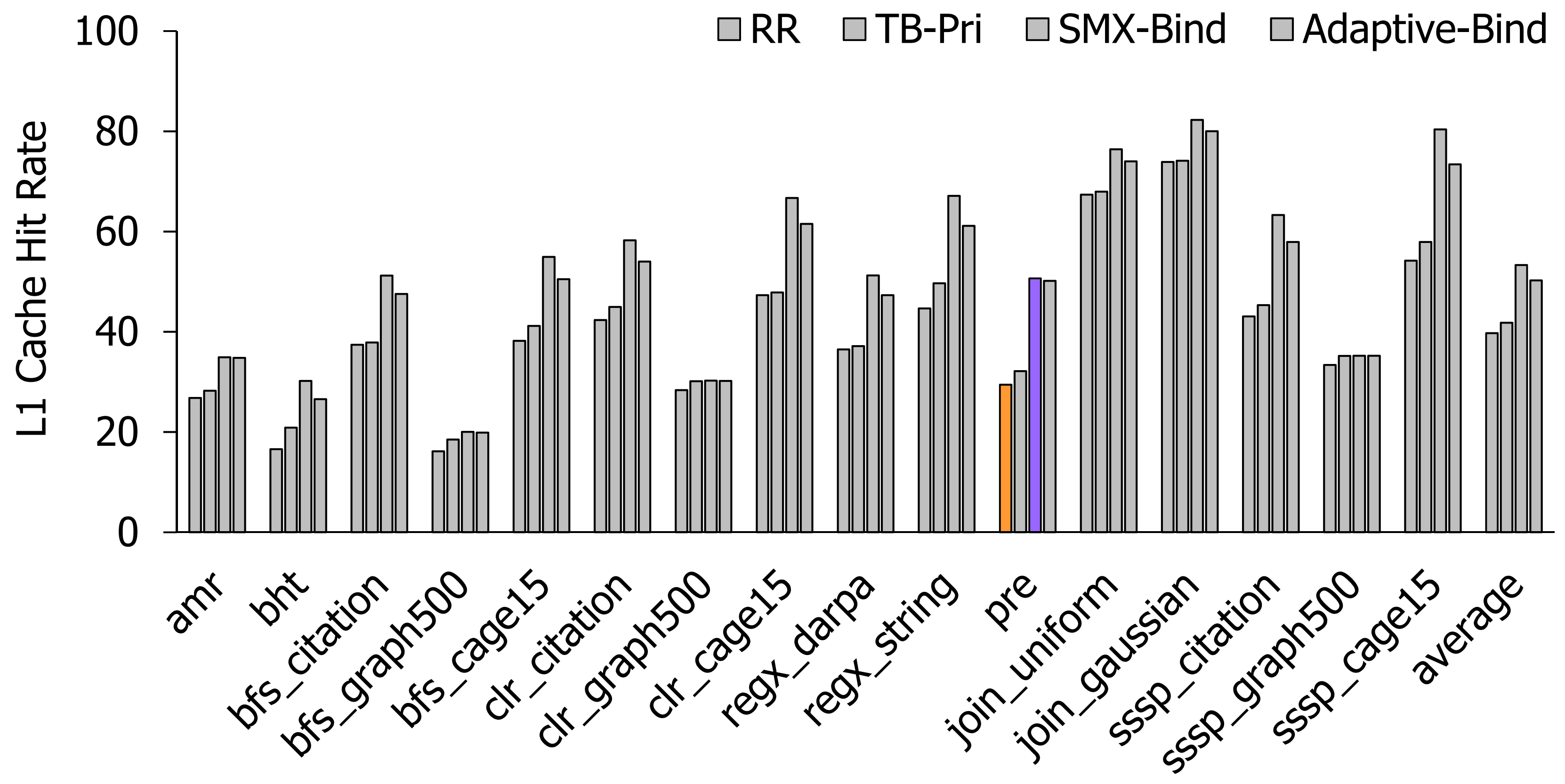


Performance of LaPerm: L1 Cache



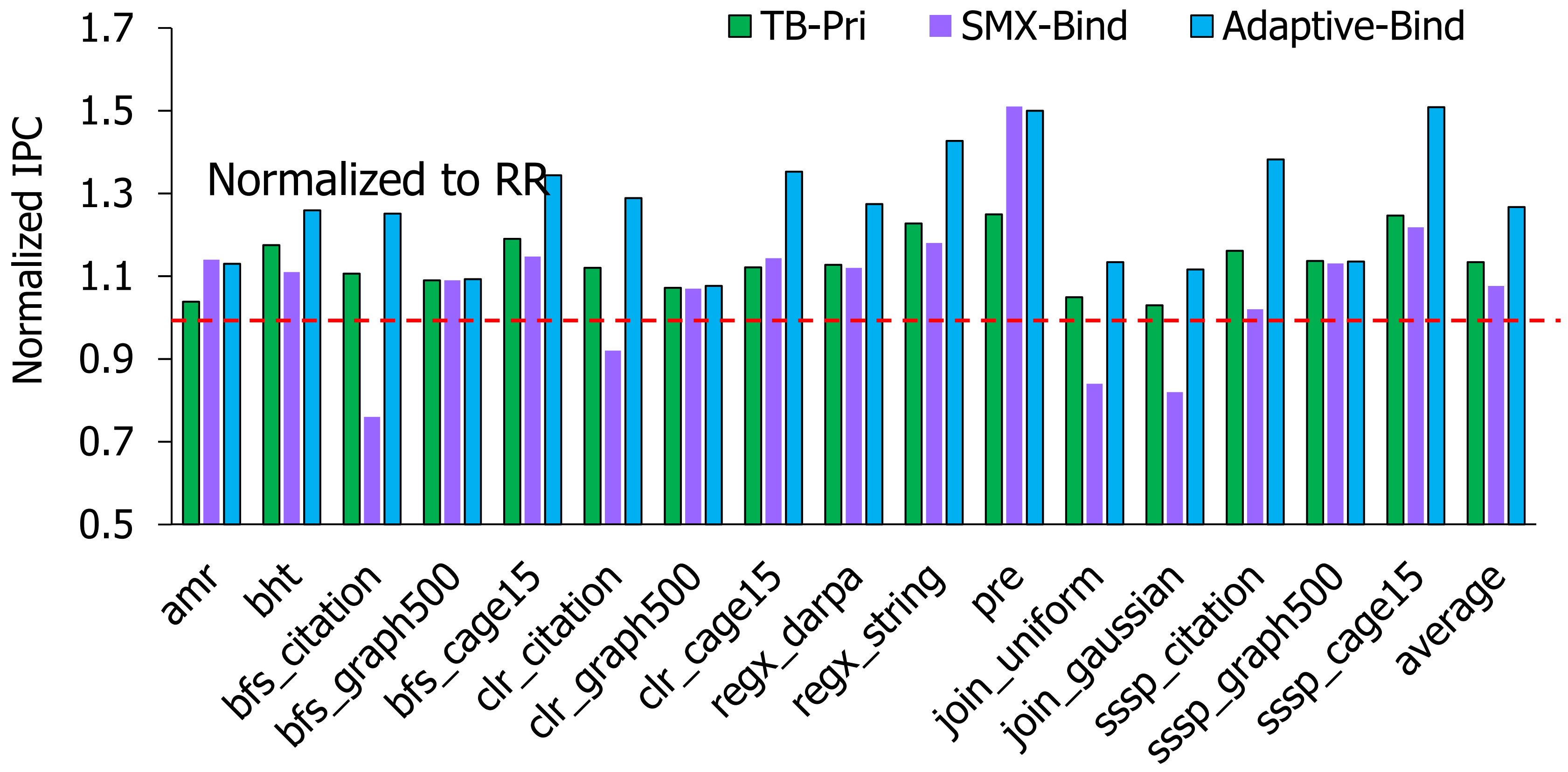
- L1 Cache hit rate benefits mainly from binding child TBs to parents' SMXs

Performance of LaPerm: L1 Cache

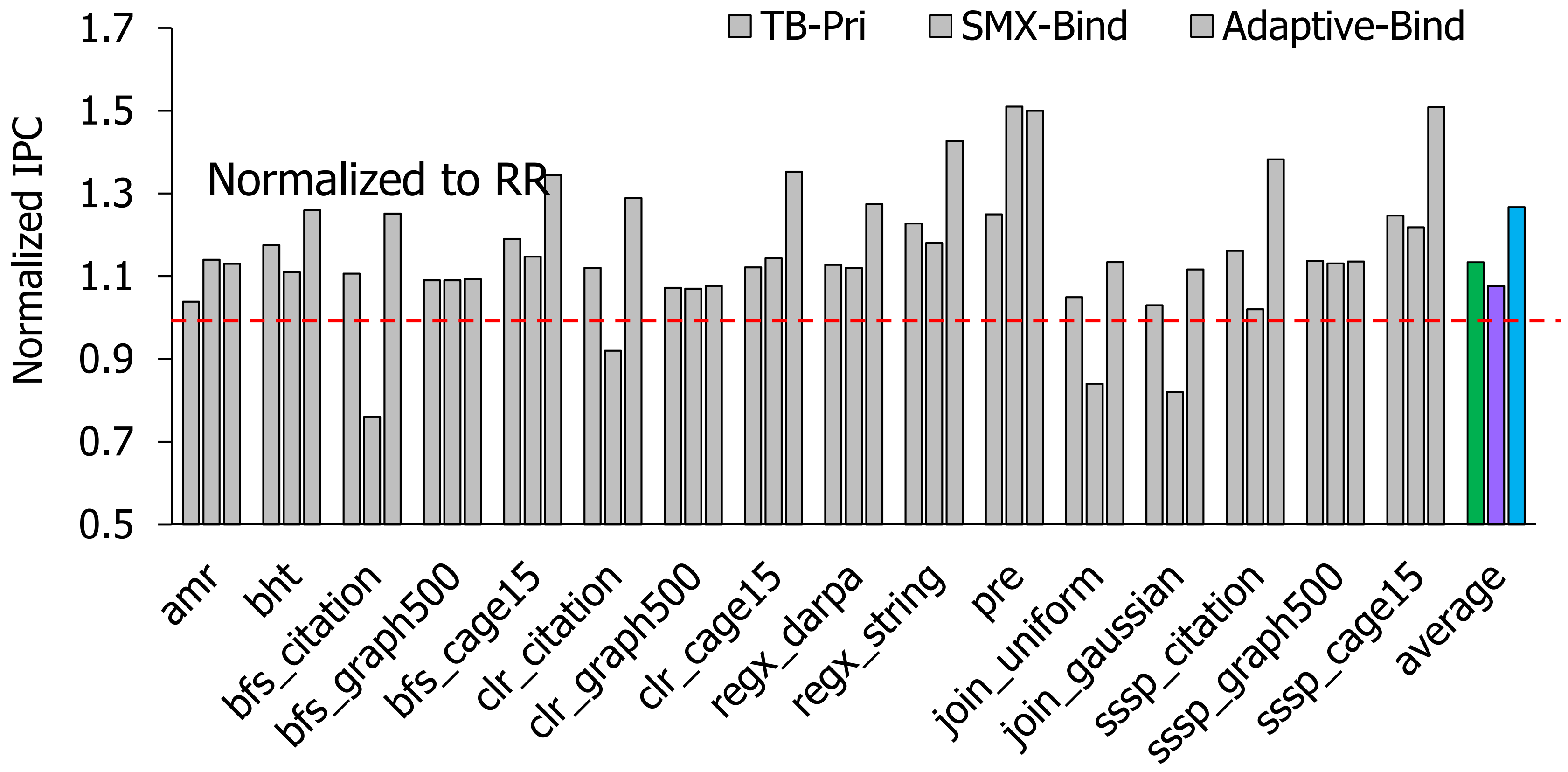


- L1 Cache hit rate benefits mainly from binding child TBs to parents' SMXs
- Product recommendation *pre* has good child-sibling data locality

Performance of LaPerm: IPC



Performance of LaPerm: IPC



- TB-Pri: IPC (1.13x) increases due to higher L2 cache hit rate
- SMX-Bind: IPC decreases (1.08x) from TB-Pri due to higher L1 cache hit rate but SMX load balancing
- Adaptive-Bind: Overall IPC (1.27x) increases because of both memory efficiency and load balancing

Conclusion

- LaPerm: Locality-aware thread block scheduler for dynamic parallelism
  - Exploit new memory reference locality in the parent-child launching

- Three scheduling decisions
  - Increase L1/L2 cache locality while maintaining SMX load balance
  - Achieve overall memory system efficiency and performance improvement

THANK YOU!

QUESTIONS?