Priority-Based Cache Allocation in Throughput Processors

Dong Li*, Minsoo Rhu*, Daniel R. Johnson, Mike O'Connor, Mattan Erez, Doug Burger, Donald S. Fussell and Stephen W. Keckler

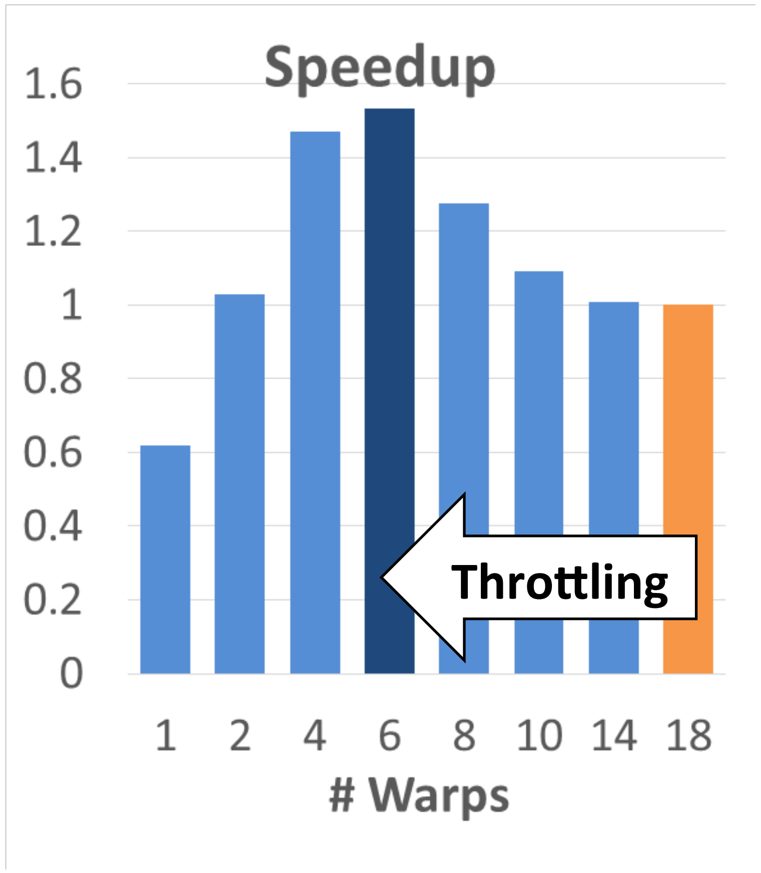
*First authors Li and Rhu have made equal contributions to this work and are listed alphabetically

Introduction

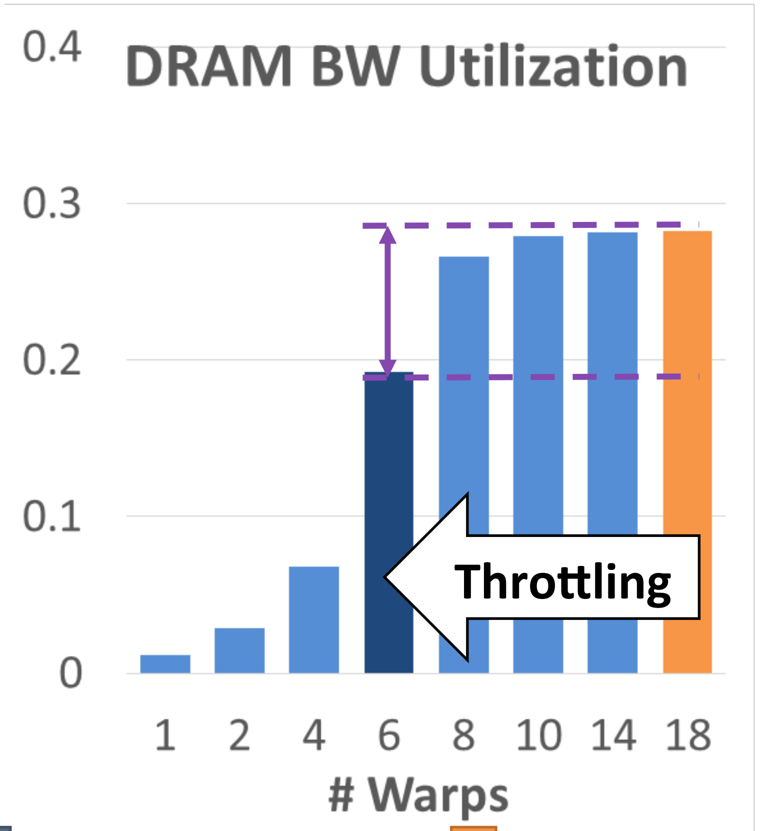
- Graphics Processing Units (GPUs)
  - General purpose throughput processors
  - Massive thread hierarchy, warp=32 threads
- **Challenge**:
  - Small caches + many threads → contention + cache thrashing
- Prior work: throttling thread level parallelism (TLP)
  - **Problem 1:** under-utilized system resources
  - **Problem 2:** unexplored cache efficiency

Motivation: Case Study (CoMD)

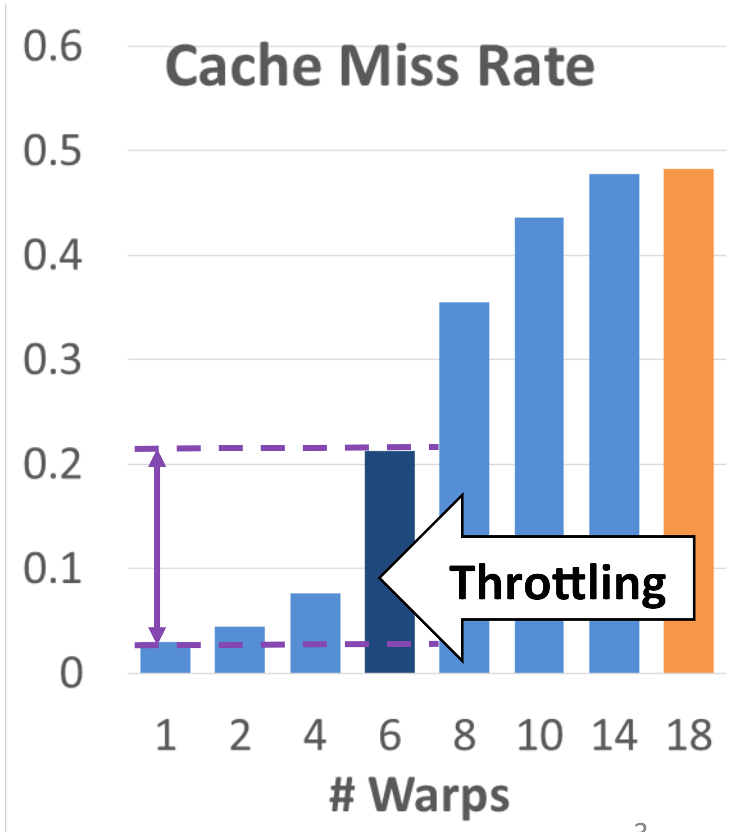
Throttling improves performance



Observation 1: Resource under-utilized



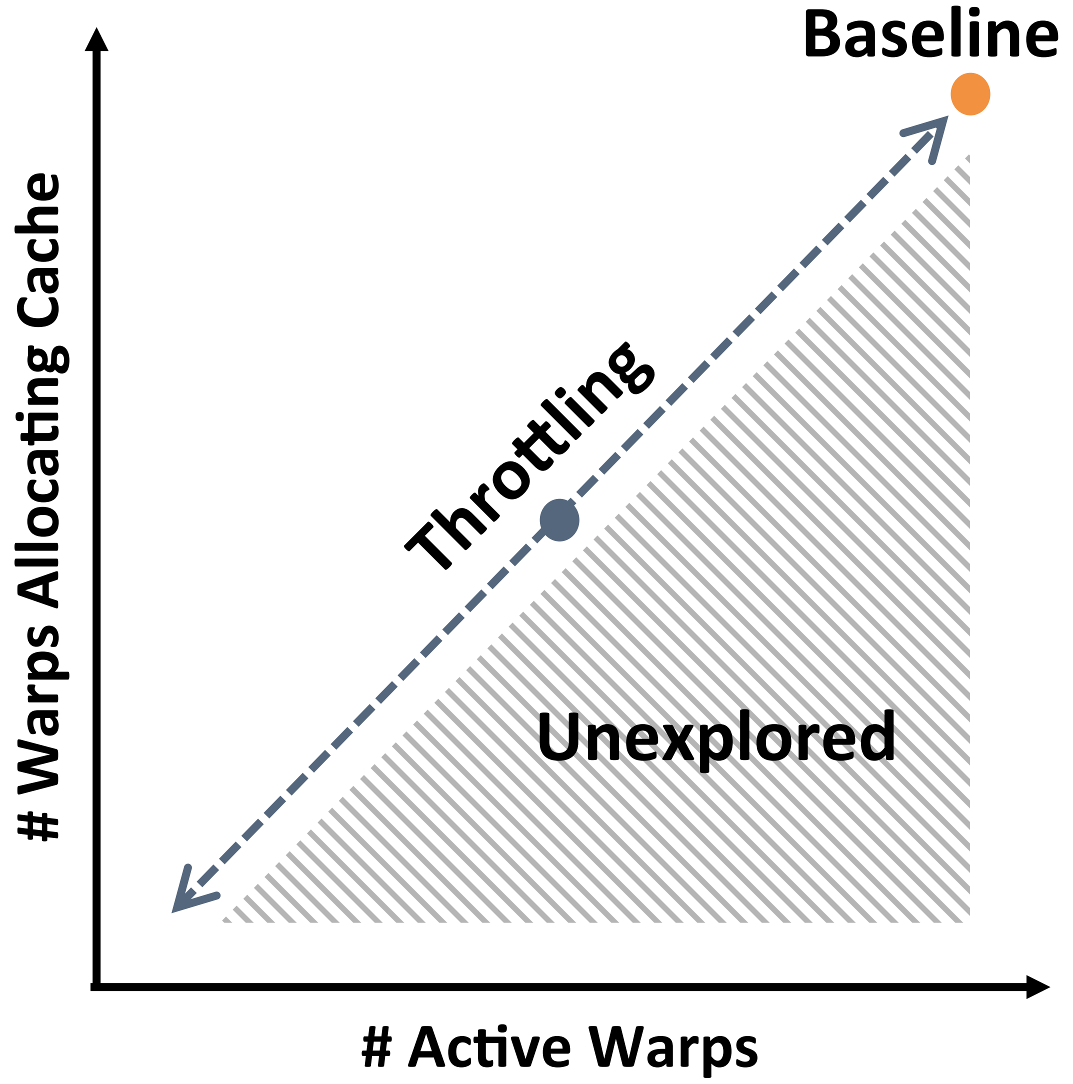
Observation 2: Unexplored cache efficiency



▇ Best Throttled ▇ max TLP (default)

Motivation

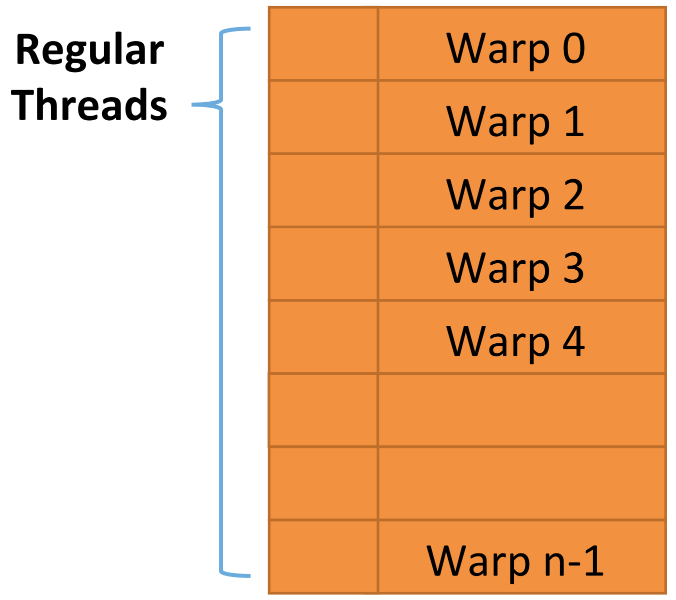
- Throttling
  - Tradeoff between cache efficiency and parallelism.
  - # Active Warps = # Warps Allocating Cache
- Idea
  - Decoupling cache efficiency from parallelism



Our Proposal: Priority Based Cache Allocation (PCAL)

Baseline

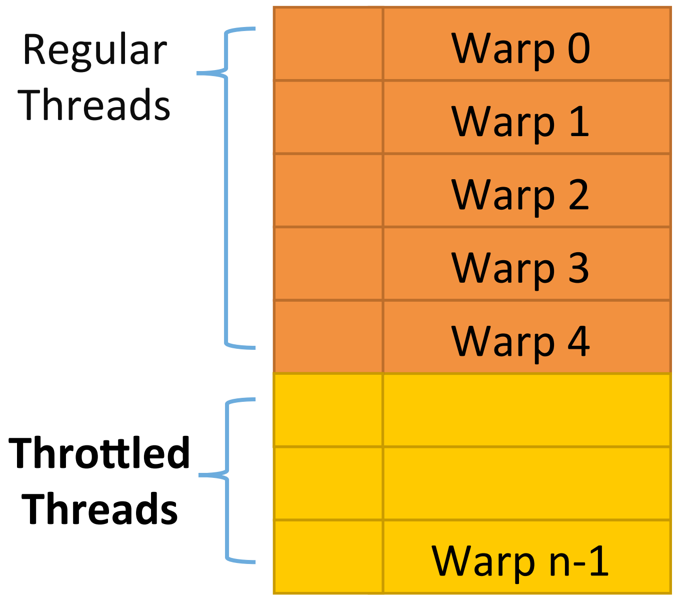
Standard operation



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)

TLP Throttling

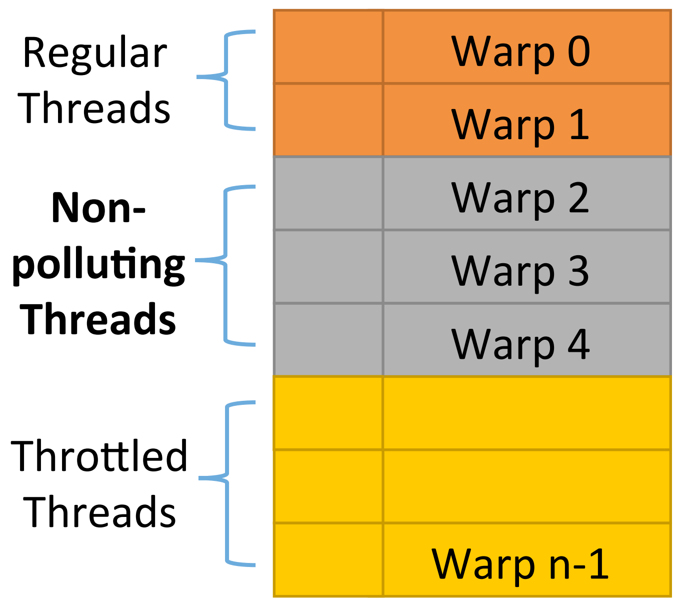
TLP reduced to improve performance



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)
- Throttled threads: throttled (not runnable)

Proposed Solution: PCAL

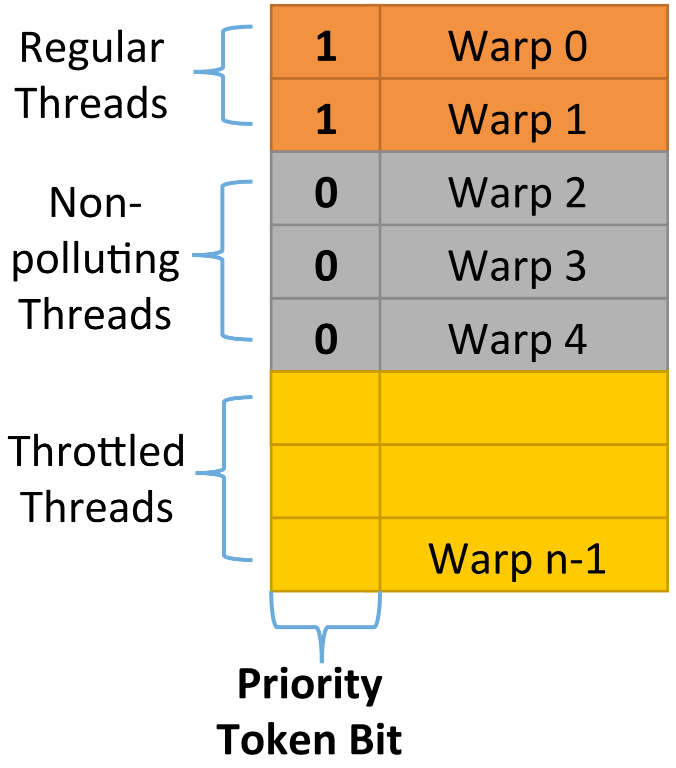
Three subsets of threads



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)
- Non-polluting threads: runnable, but prevented from *polluting* L1 cache
- Throttled threads: throttled (not runnable)

Proposed Solution: PCAL

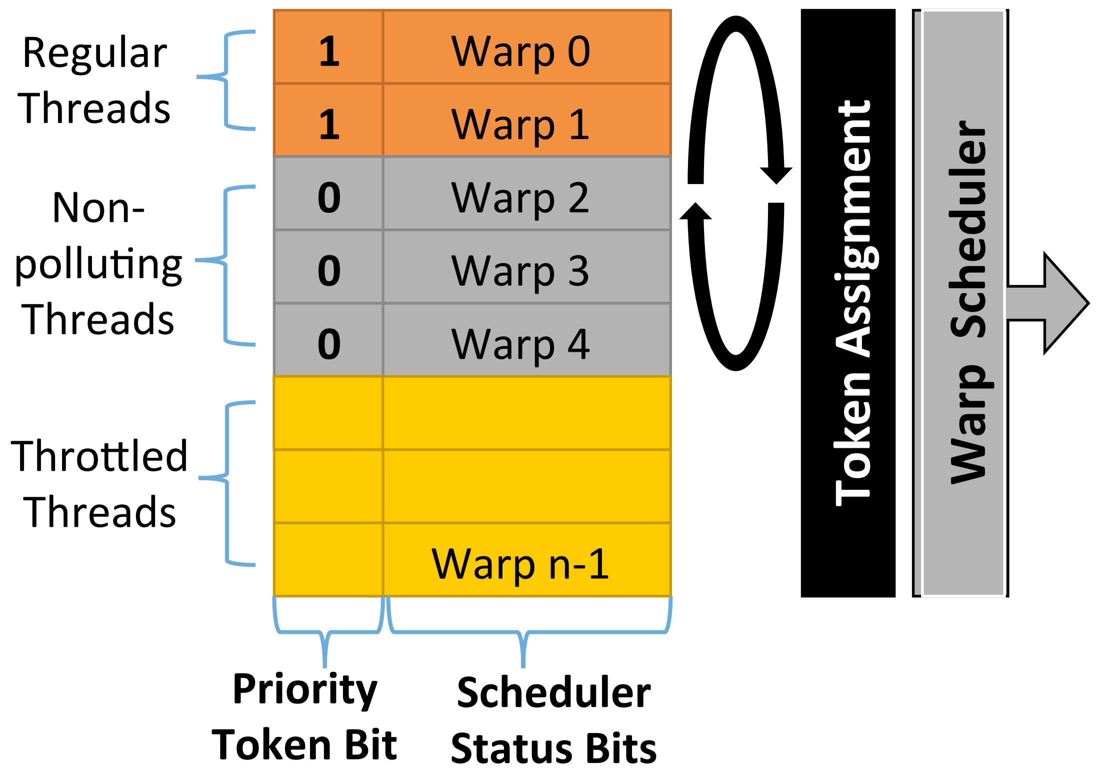
Tokens for privileged cache access



- **Tokens** grant capabilities or privileges to some threads
- Threads with tokens are allowed to ***allocate and evict*** L1 cache lines
- Threads without tokens can read and write, but ***not allocate or evict***, L1 cache lines

Proposed solution: Static PCAL

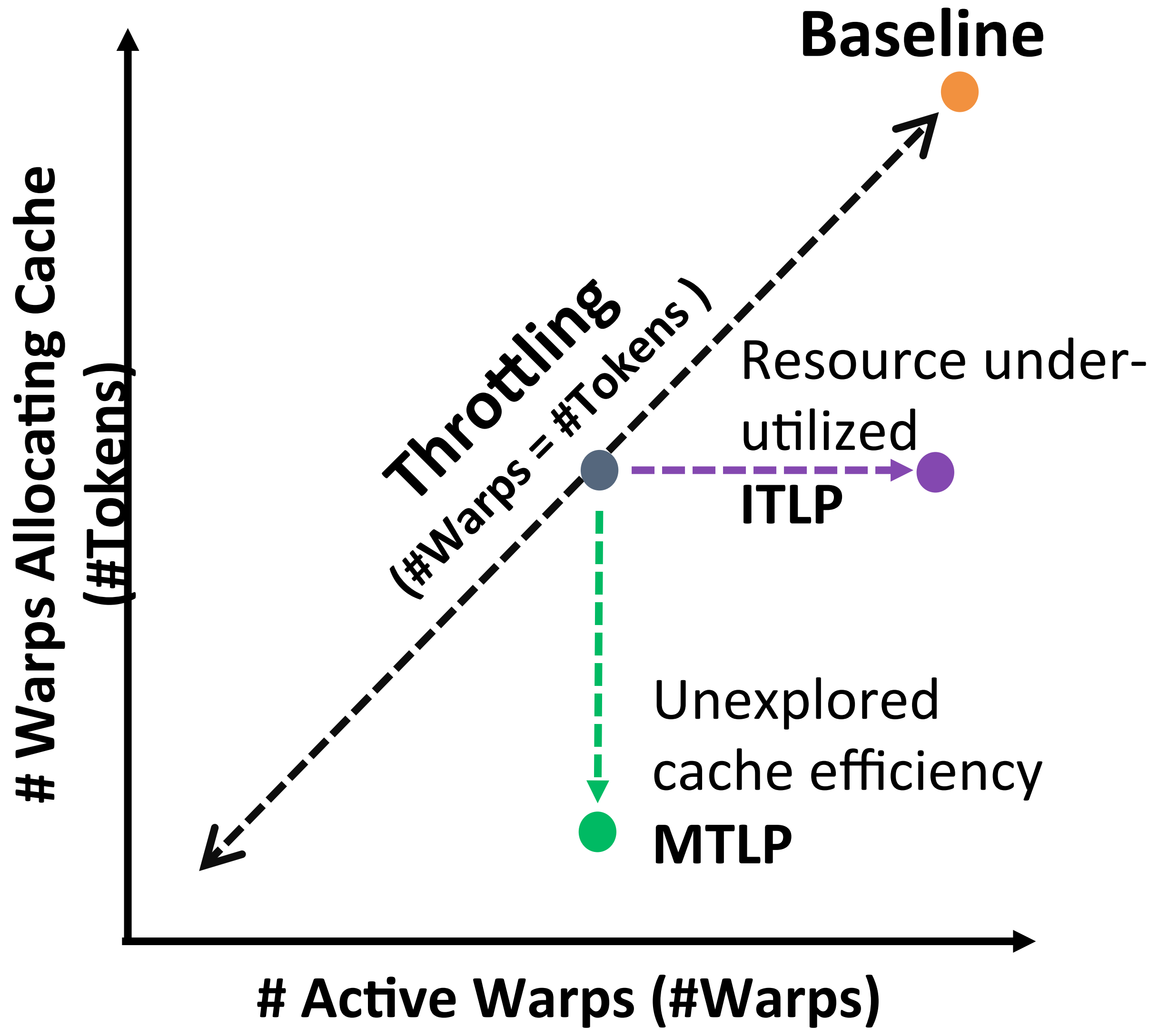
Enhanced scheduler operation



- HW implementation: simplicity
  - Scheduler manages per-warp token bits
  - Token bits sent with memory requests
- Policies: token assignment
  - Assign at warp launch, release at termination
  - Re-assigned to oldest token-less warp
- Parameters supplied by software prior to kernel launch

Two Optimization Strategies

to exploit the 2-D (#Warps, #Tokens) search space



- **1. Increasing TLP (ITLP)**
  - Adding non-polluting warps
  - Without hurting regular warps
- **2. Maintaining TLP (MTLP)**
  - Reduce #Token to increase the efficiency
  - Without decreasing TLP

Proposed Solution: Dynamic PCAL

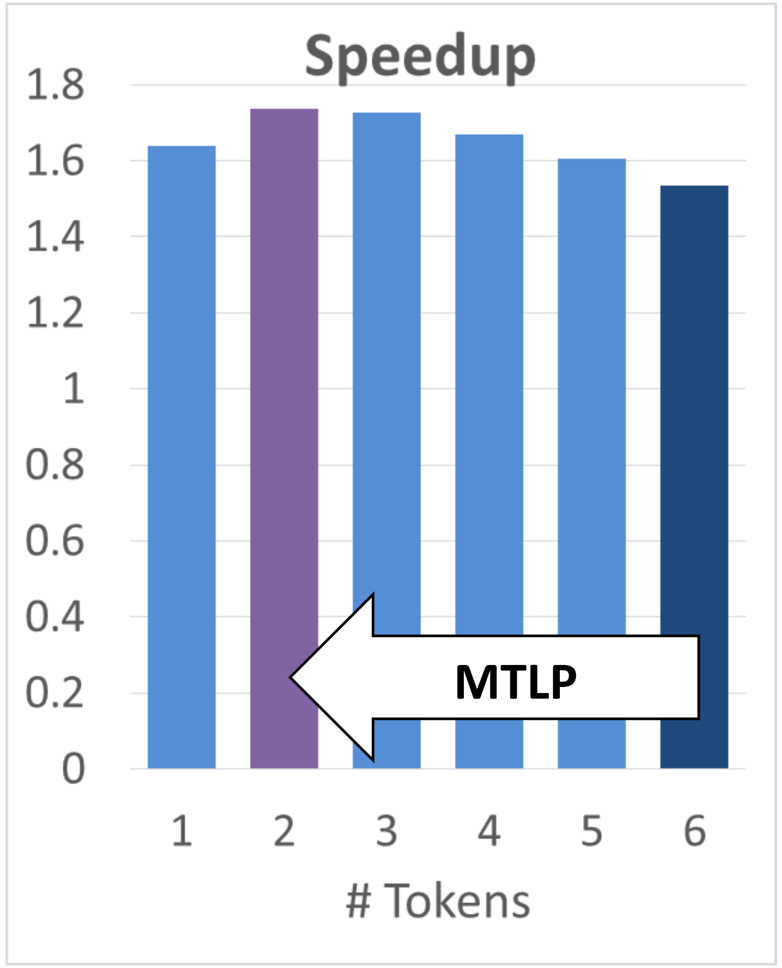
- Decide parameters at run time
  - Choose MTLP/ITLP based on resource usage performance counter
  - For MTLP: search **#Tokens** in parallel
  - For ITLP: search **#Warps** in sequential
- Please refer to our paper for more details

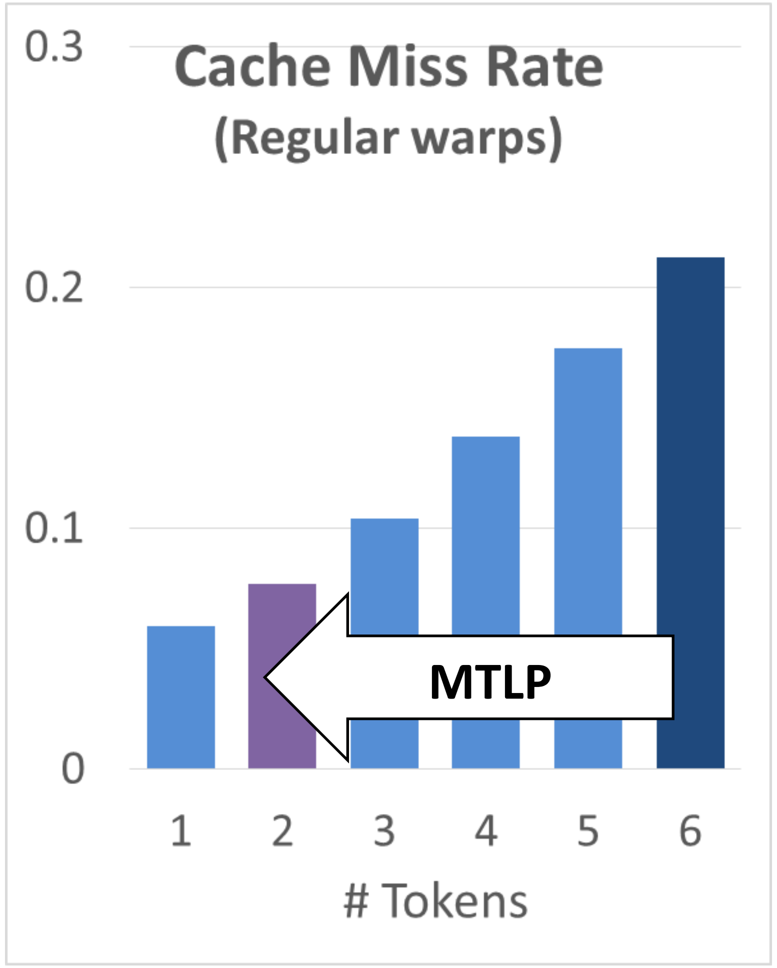
Evaluation

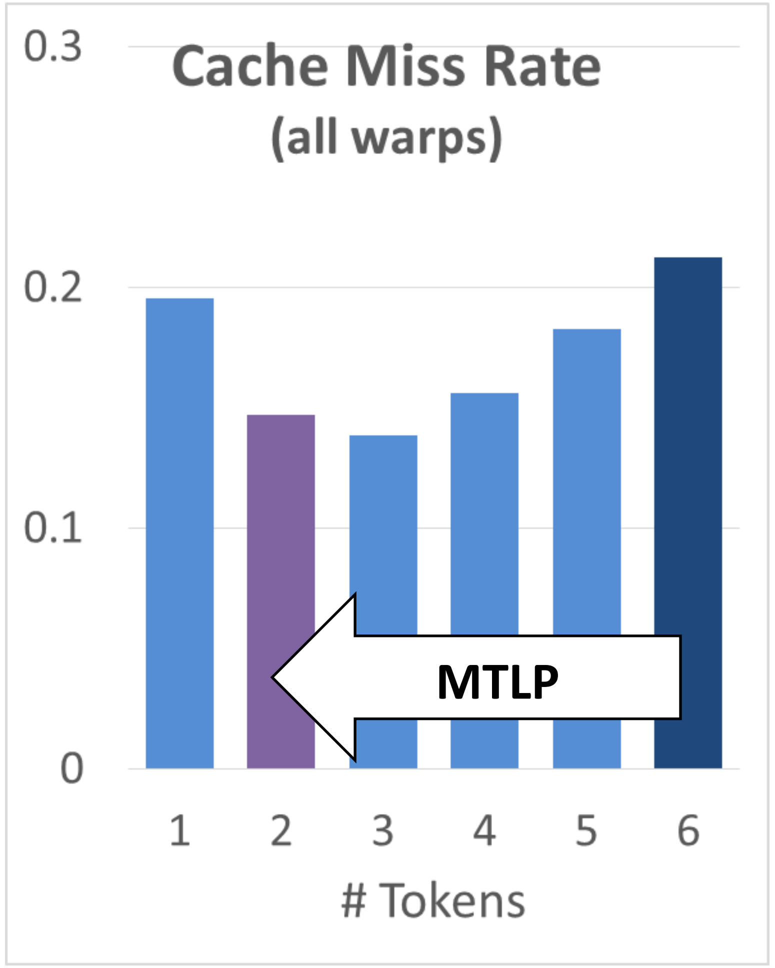
- Simulation: GPGPU-Sim
  - Open source academic simulator, configured similar to Fermi (full-chip)
- Benchmarks
  - Open source GPGPU suites: Parboil, Rodinia, LonestarGPU. etc
  - Selected benchmark subset shows *sensitivity to cache size*

MTLP Example (CoMD)

MTLP: reducing #Tokens, keeping #Warps =6



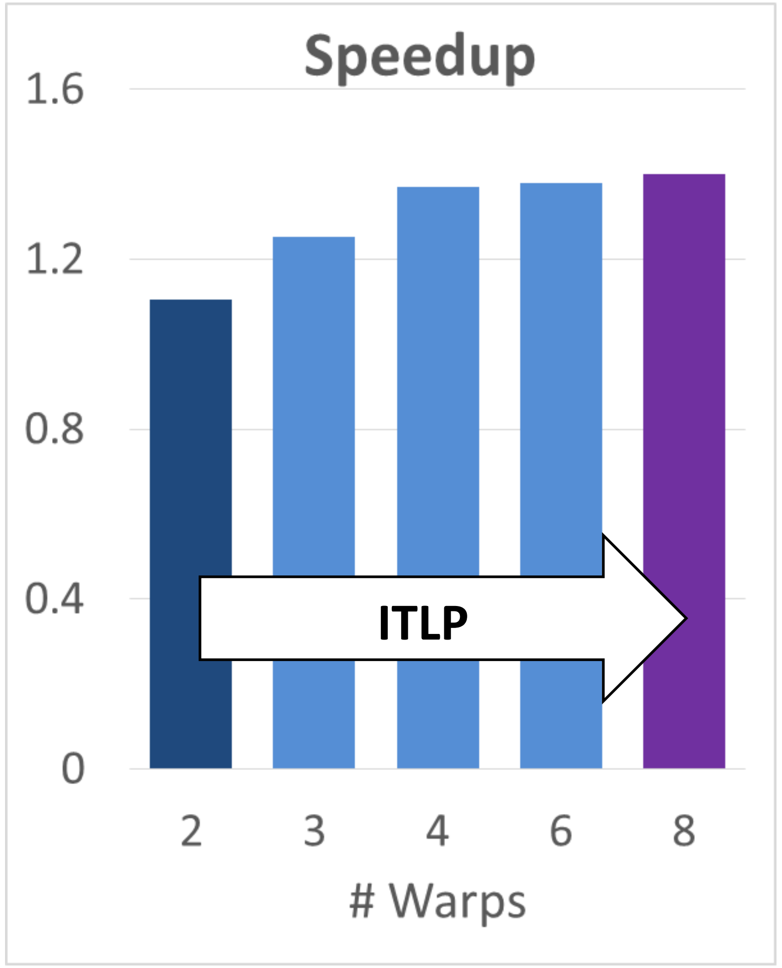


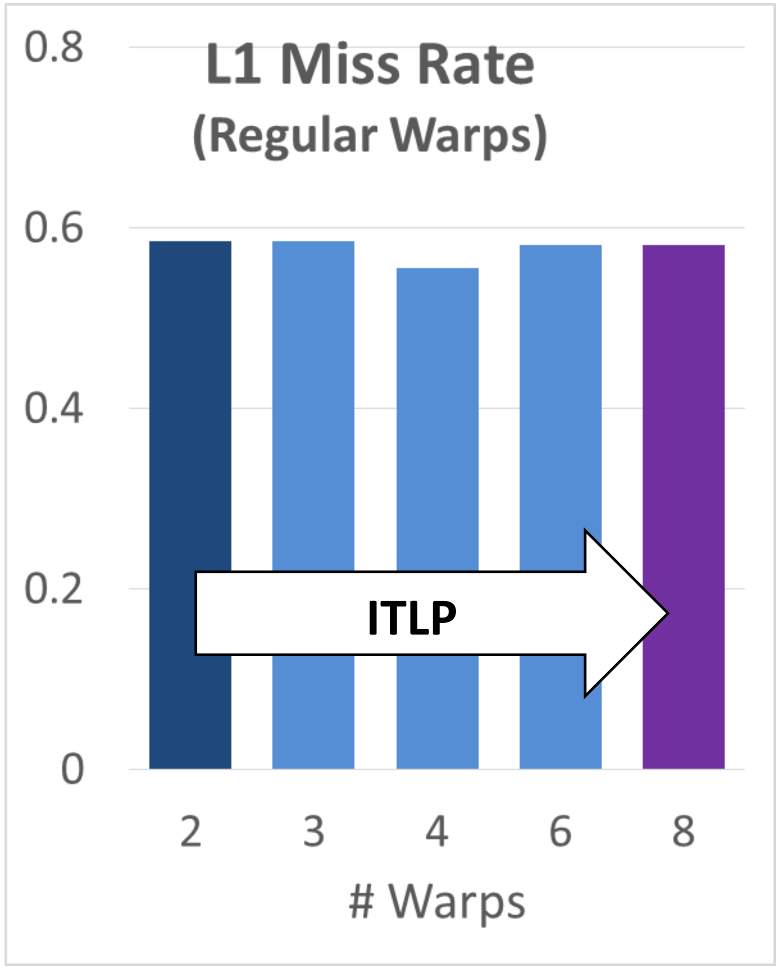


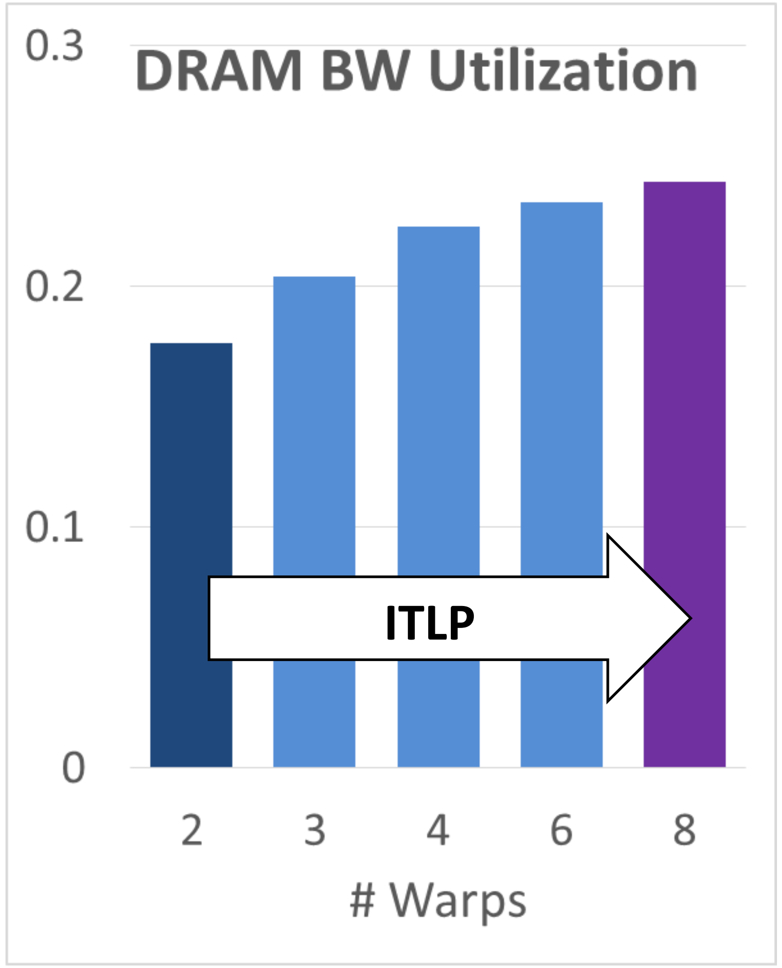
🟪 PCAL with MTLP ⬛ Best Throttled

ITLP Example (Similarity-Score)

ITLP: increasing #Warps, keeping #Tokens=2

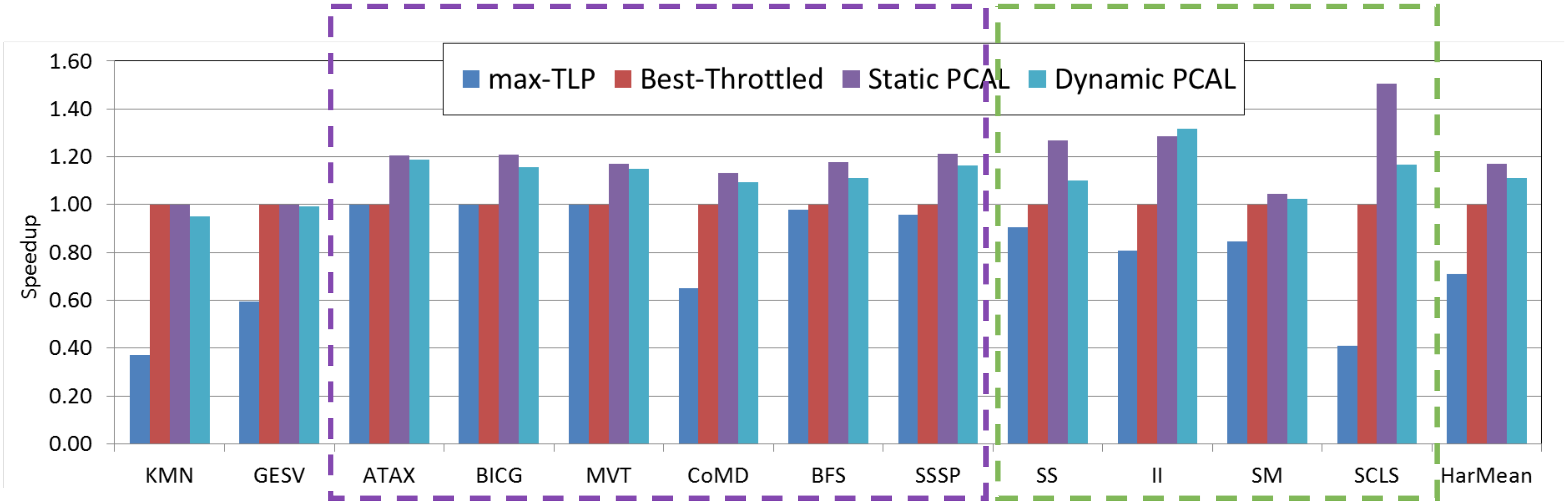






🟪 PCAL with ITLP 🟦 Best Throttled

PCAL Results Summary



- Baseline: best throttled results
- Performance improvement: Static-PCAL: 17%, Dynamic-PCAL: 11%

Conclusion

- Existing throttling approach may lead to two problems:
  - Resources underutilization
  - Unexplored cache efficiency
- We propose PCAL, a simple mechanism to rebalance cache efficiency and parallelism
  - MTLP: alleviates cache thrashing while maintaining parallelism
  - ITLP: increases parallelism without hurting regular warps
- Throughput improvement over best throttled results
  - Static PCAL: 17%
  - Dynamic PCAL: 11%