# A Sharing-Aware L1.5D Cache for Data Reuse in GPGPUs

ASP-DAC 2019

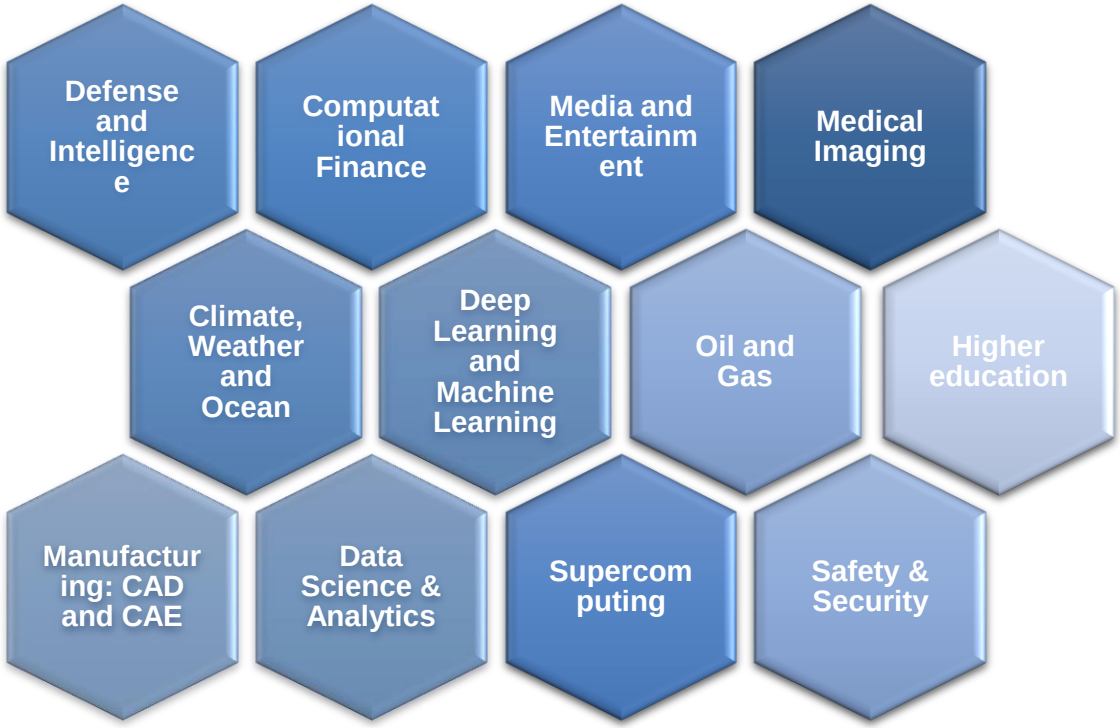
**Jianfei Wang**, Li Jiang, Jing Ke,
Xiaoyao Liang, Naifeng Jing*

Shanghai Jiao Tong University, Shanghai, China

# Outline

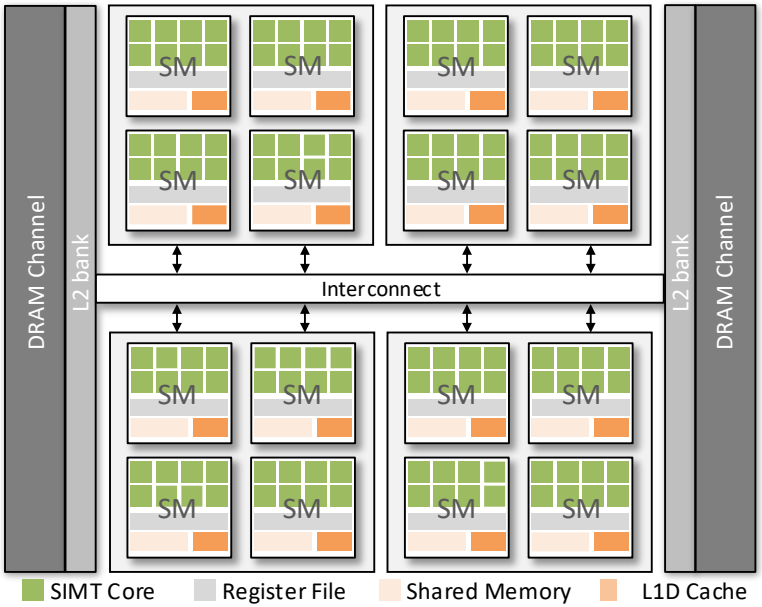
- Background and motivation
- Proposal: A Sharing-Aware L1.5D Cache
  - Structure and Layout
  - Sharing-aware Management
- Experiment Setups and Results
- Summary

# GPU Accelerated Computations



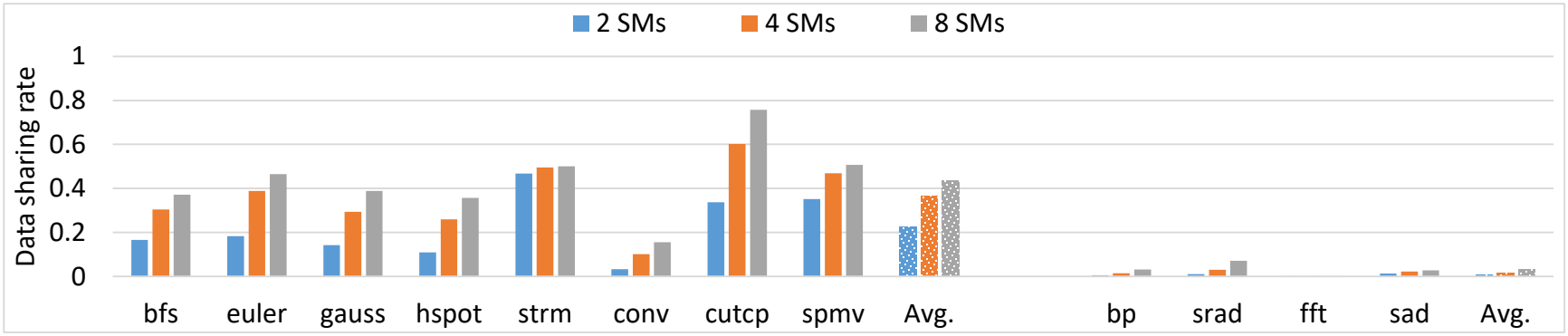

"From the TOP500 supercomputer list, about 56% computing power is from GPGPUs…"

# GPU Architecture Overview




- 4 clusters, each consisting of 4 SMs (GTX480/980)
  - Thousands of threads execute on massive CUDA cores under SIMT style
  - Capacity of L1D is much smaller than RF and Shmem

# Observations and Motivations



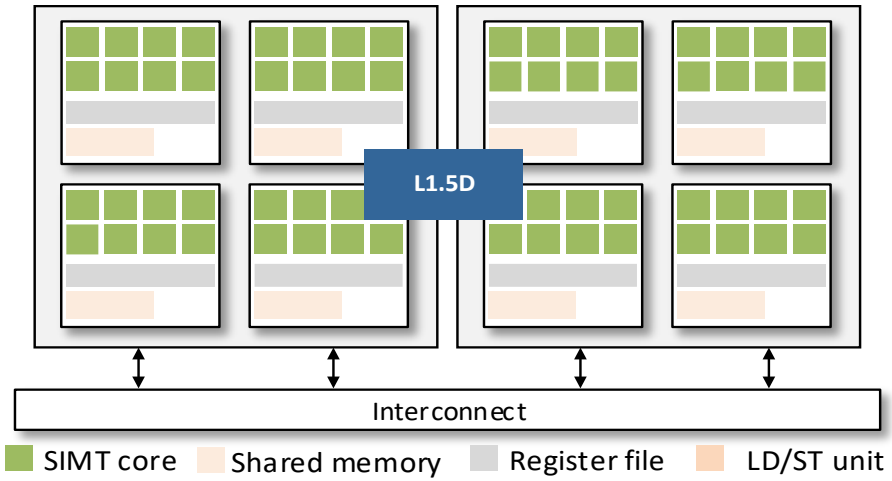

- Duplicated data among SMs since L1D is private
  - L1D missed requests can be served by neighboring SMs
  - 8 SMs: up to 75.7% with an average of 43.8%
  - 4 SMs: 36.4% on average（means 27.3% more data）
- Move L1D out and combine them as L1.5D cache

# A Sharing-Aware L1.5D Cache Overview

- Two challenges must be addressed for L1.5D
  - The increased latency due to the bigger capacity and longer wire distance
  - Sharable data thrashing problem due to the limited capacity and the default warp scheduler (intra-warp locality)

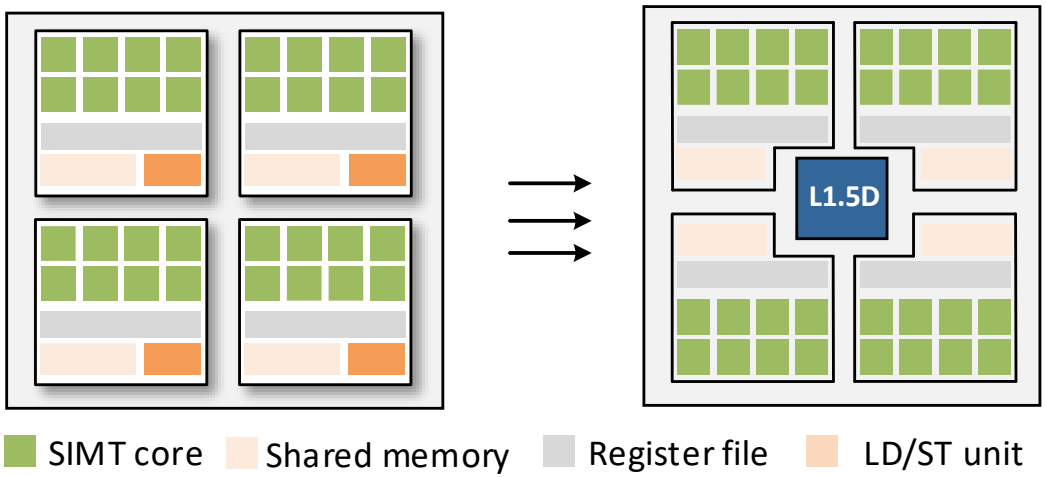
# L1.5D: Structure and Layout

- A compatible structure based on the actual 2D layout
  - “8-SM L1.5D”:  Highest sharing rate, cross-cluster communications, long latency and multi-ported structure is needed

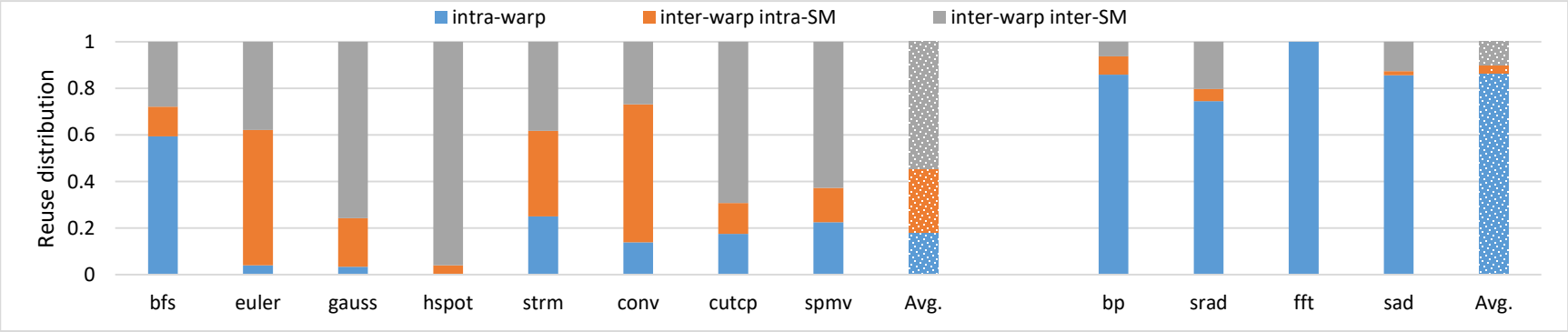


# L1.5D: Structure and Layout

- A compatible structure based on the actual 2D layout
  - “4-SM L1.5D”: Symmetric accesses, shorter wire latency
  - “2-SM L1.5D”: Same merits but least sharing rate



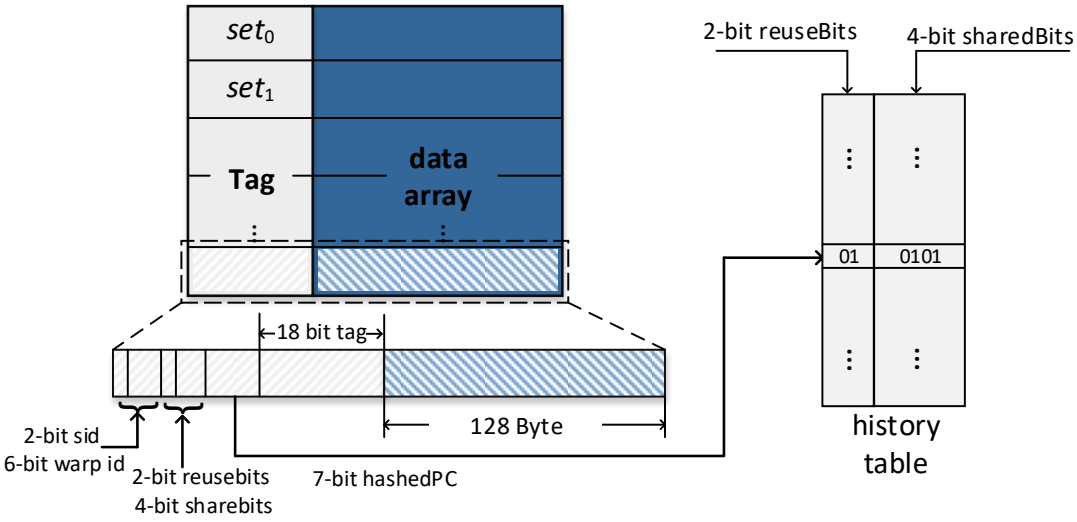
# L1.5D: Behavior Analysis




- Taxonomy of cache requests based on reusability
  - Inter-warp reuse (sharable data) is dominating
  - Greedy-then-oldest scheduler is skilled at intra-warp
- Sharable data should be identified and protected

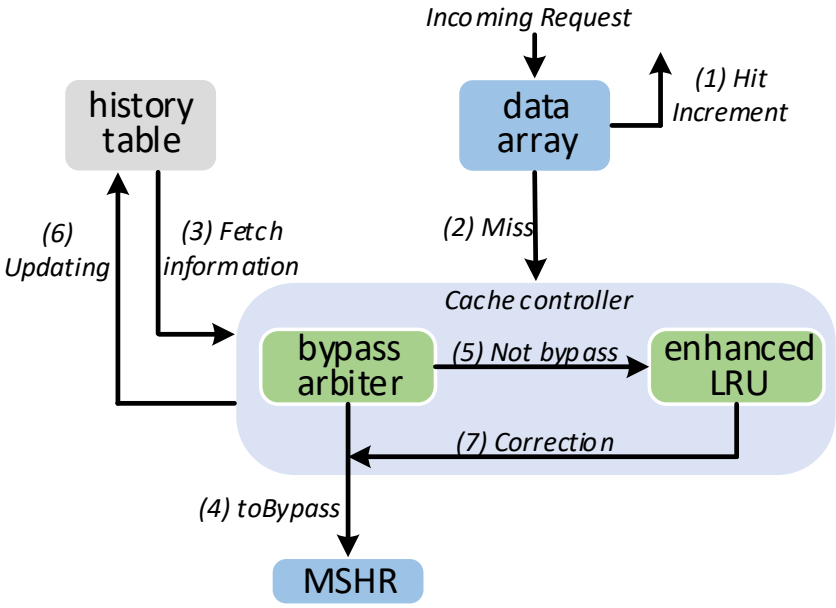
# L1.5D: Sharing-aware Management

- A history table to record access information
  - Same PCs lead to similar behaviors [ISCA2001, MICRO2010] and such correlation is extended to sharing possibility
  - History table is indexed by PCs
  - 2-bit *reusebits* and 4-bit *sharebits* for recording



# Sharing-aware Management（cont.）

- 1. Hit, retrieved as before
- 2. Miss, fetch reuseBits from HT
- 3. reusebits comparison
- 4. Bypass, record in MSHR and self correction
- 5. Not bypass, get sharebits from HT and protect shared ones
- 6. Victim blocks, update the HT
- 7. Correct bypass decision in MSHR



# Experiment Setting

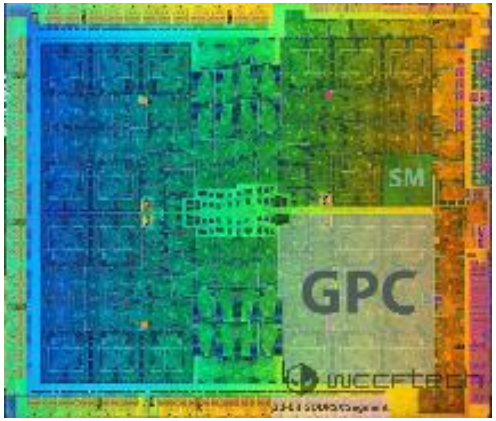
- Simulator: GPGPU-Sim v3.2.2
- Benchmarks: 12 apps. from Rodina and Parboil

| | Baseline | L1.5D |
|---|---|---|
| #SMs | 16, 4 per cluster | |
| Warp size | 32 threads | |
| Scheduler | 2 GTO warp scheduler per SM | |
| TLP | 2048 threads, 64 warps, 32 CTAs per SM | |
| On-chip memory | **L1D: 16KB per SM<br>tag: ≈1.25KB, 4-way<br>History table: n/a** | **L1.5D: 64KB per cluster<br>≈2.44KB, 4-way per cluster<br>≈0.1KB, 128-entry** |
| | 128B cache line, LRU, 48KB shared memory | |
| L2 cache | Unified, 128KB x 16, 128 line size, 8-way, LRU | |

- Cost: Extra 1.19KB tag bits to store ID, access information and hashedPC and 768 bits for the history table per cluster

# Experiment Setting (cont.)

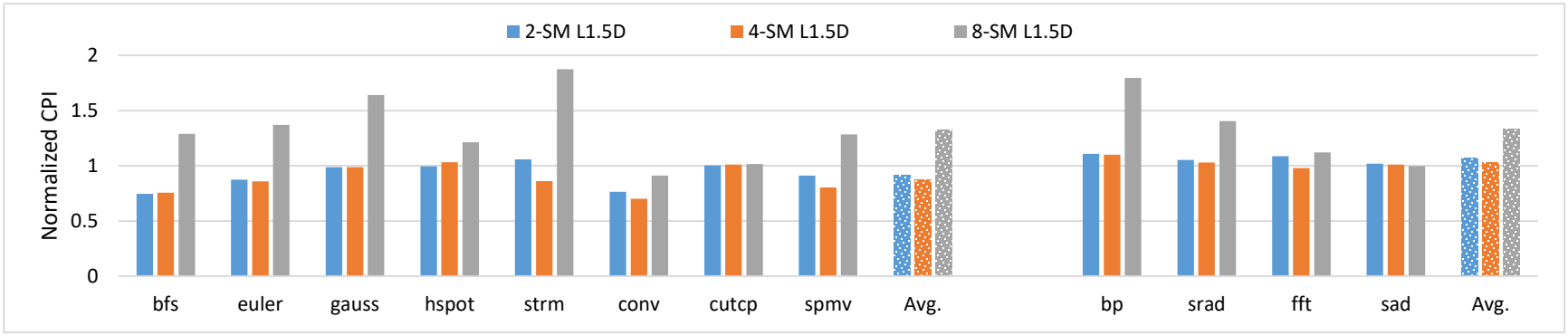
- To learn the L1.5D latency, we measure the wire length on a die photo and calculate the planar wire latency
- Using CACTI 6.5 to get cache access time and energy
- “8-SM L1.5D”: Latency of remotest SMs conservatively



Die photo of GTX480

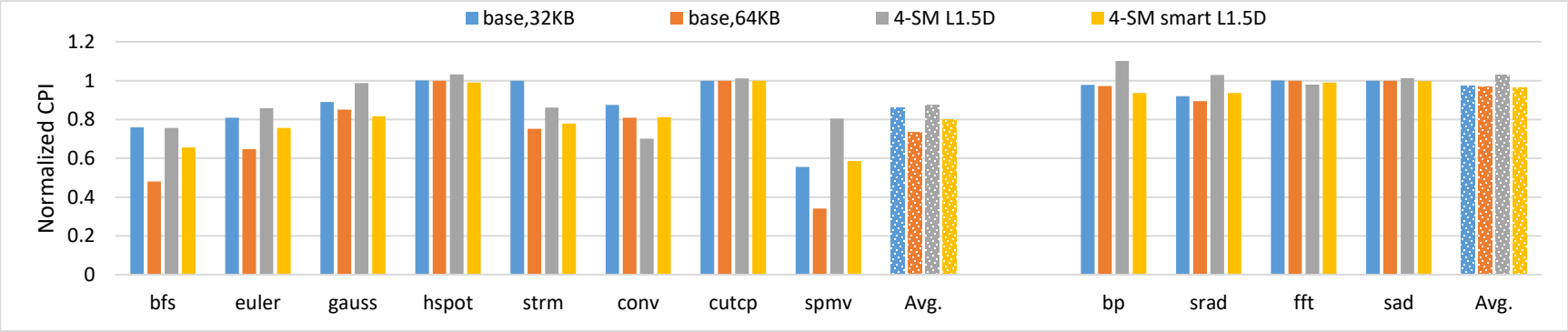
| Option | capacity | access time(ns) | wire length(mm) | latency |
|---|---|---|---|---|
| Baseline | 16KB | 1.17 (2 cycles) | 0.7 (1 cycle) | 4 cycles |
| 2-SM | 32KB | 1.19 (2 cycles) | 1.0 (2 cycles) | 6 cycles |
| 4-SM | 64KB | 1.23 (2 cycles) | 1.3 (2 cycles) | 6 cycles |
| 8-SM | 128KB | 1.44 (2 cycles) | 5.6 (8 cycles) | 18 cycles |

# Evaluation on Performance



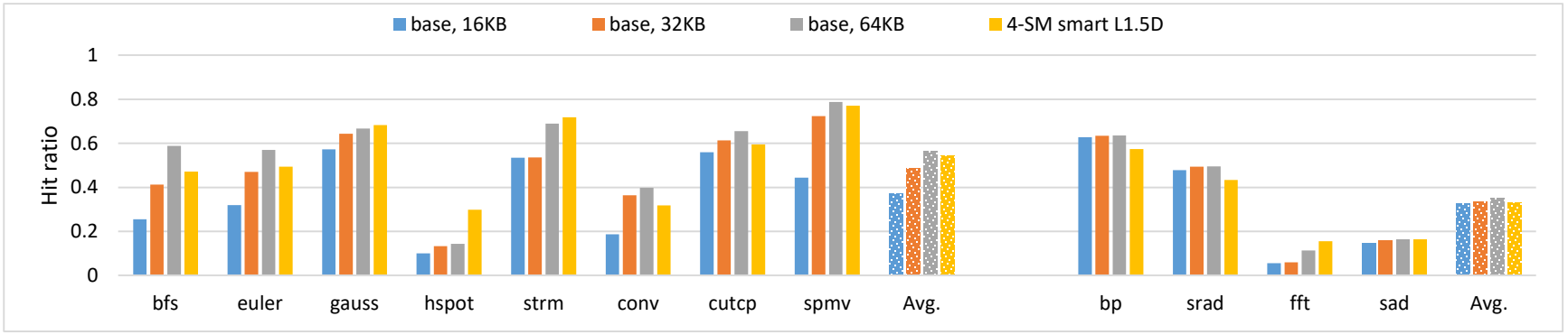

- 8-SM L1.5D: longer latency, degrades about 32.5%
- 4-SM L1.5D: 12.3% improvement for highly sharable applications, degrades about 3.1% for others
- 2-SM L1.5D: less sharing possibility , less improvement
- Cache sensitivity: “spmv” gains 19.6% while “hspot” gets little

# Comparisons on Performance



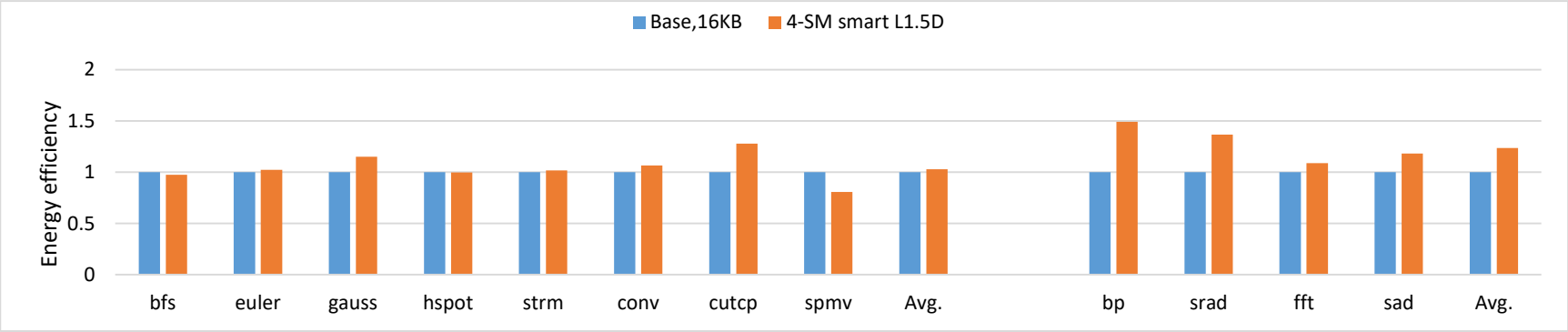

- Sharing-aware management brings extra 7.8% gains
- Totally 20.1%, better than baseline with 32KB L1D
- Lowly sharable applications: “bp” and “srad” get small improvements due to bypass technique

# Comparisons on Memory Statistics




- Hit rate increase of 16.9% against 16KB-L1D baseline
- "conv“: achieve the most performance improvement along with twice hit rate increase

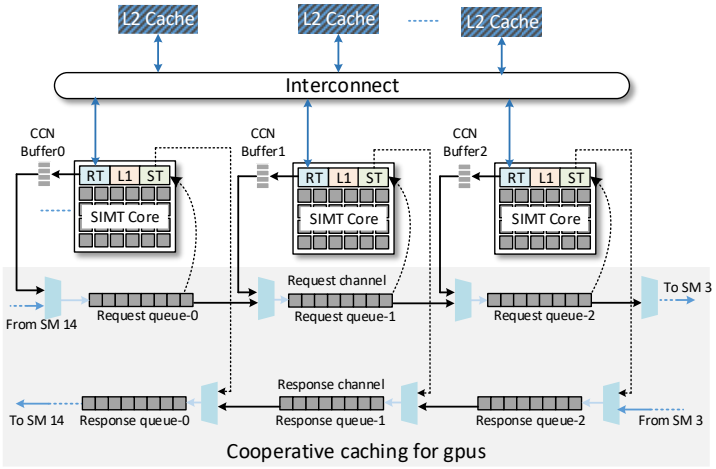
# Evaluation on Energy




- Assuming all memory requests hit in L2 conservatively
- Less running time, less static power consumption
- About 2% more energy for highly sharable applications

# Related work

- Most similar: S. Dublish et al
  - Same observation, ring network
  - Nondeterministic Response time
  - Duplication still exists




- Cache thrashing: Dynamical memory request reordering [HPCA14], cache aware thread block scheduling[MICRO12], thread block throttle [MICRO14], cache bypassing or prioritization [HPCA15,ISCA15]…

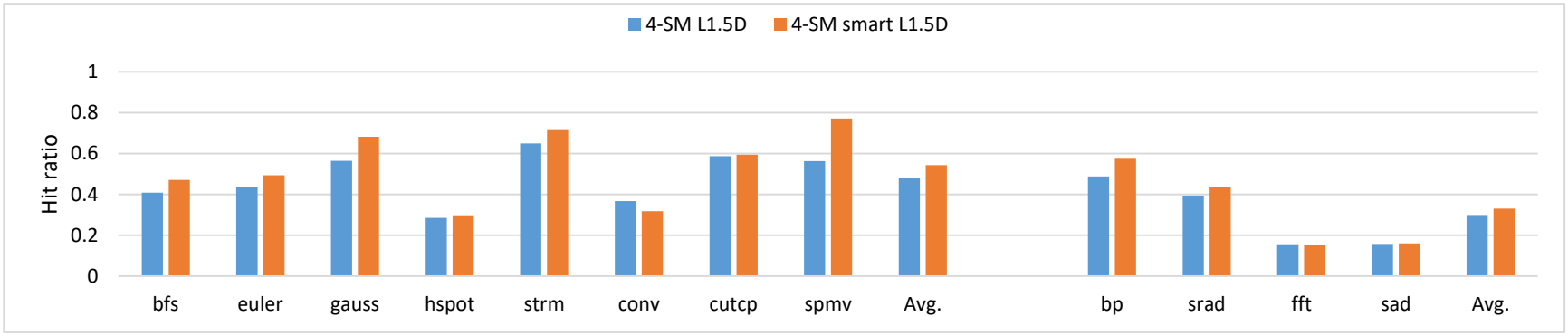
# Summary

- L1D is far from needed and duplication is a waste
- A shared L1.5D substitutes some private L1D
- Layout compatibility meets timing requirement
- Sharing-aware management protects sharable data
- Achieve an average of 20.1% CPI improvement with 16.9% hit rate increase for high sharing apps

# Thank you！

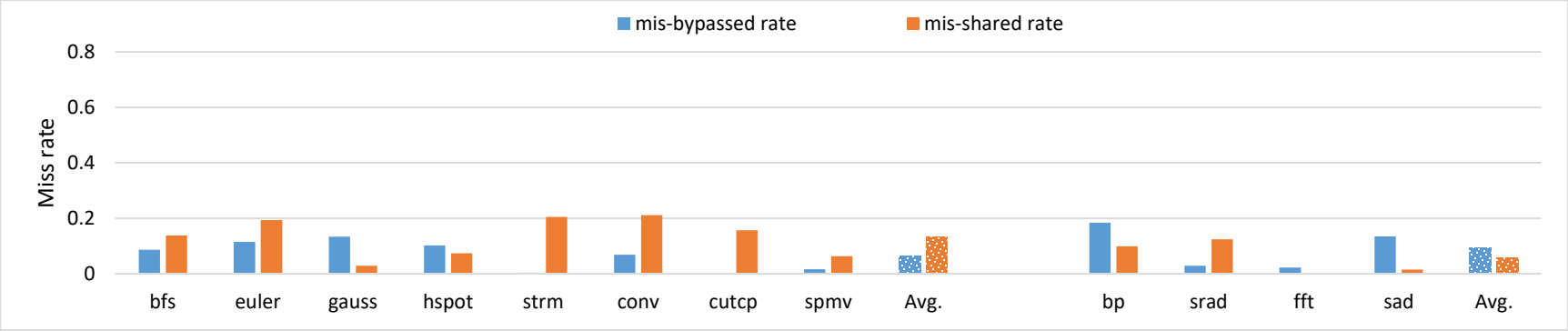
Q&A

# Backup: More Detailed Memory




- Bare L1.5D performs poorer at L1D hit rate
- About 6% gap between them

# Backup: Miss Prediction Rate




- To learn the bypassing and sharing prediction creditability
- Miss-bypassed: requested again in the following 1K cycles
- Miss-shared: number of blocks that is insufficiently shared
- Both are low, 6.5% and 13.4% respectively