COLAB: Collaborative and Efficient Processing of Replicated Cache Requests in GPU

Bo-Wun Cheng[1], En-Ming Huang[1], Chen-Hao Chao[1], Wei-Fang Sun[1],
Tsung-Tai Yeh[2], and Chun-Yi Lee[1]
[1]National Tsing Hua University, Hsinchu, Taiwan
[2]National Yang Ming Chiao Tung University, Hsinchu, Taiwan

Contribution

- **We propose Cache line Ownership Lookup tABle (COLAB), an architecture that allows replicated cache requests to be redirected and serviced efficiently within a cluster by utilizing the cache line ownership information.**
- **The incorporation of COLAB is able to reduce the GPU NoC read traffic by an average of 38% and improve the overall throughput by an average of 43% while incurring minimal overhead.**

Outline

- Introduction
- Methodology
- Experimental Results
- Conclusion
- References

Introduction

Introduction

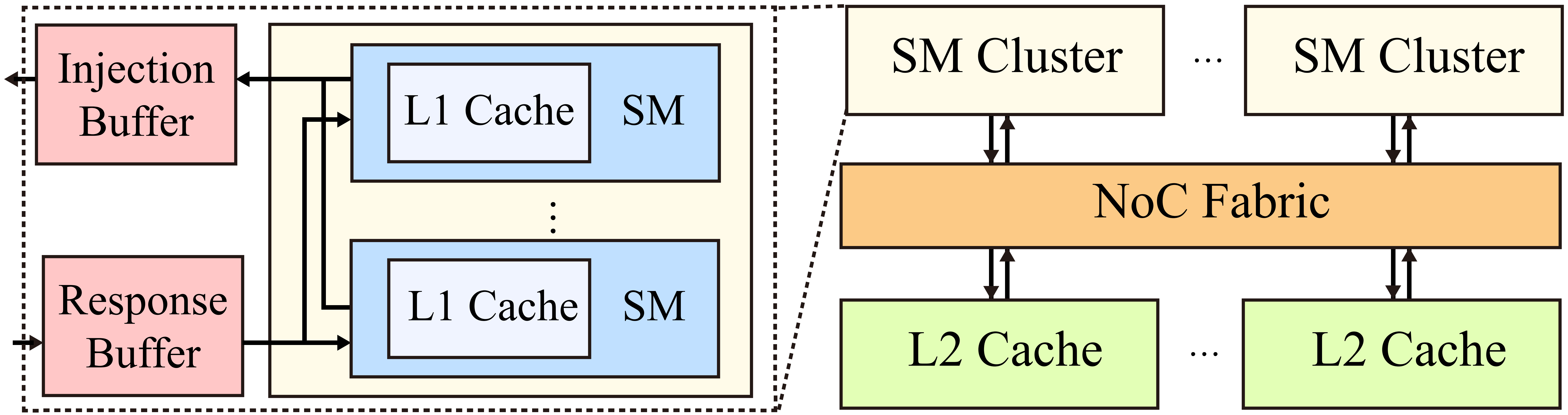
Graphics Processing Unit (GPU)

- Parallel processors that are originally built for graphics rendering.
- GPUs are increasingly utilized in general-purpose computing thanks to their:
  - Parallel processing power
  - Ease of programming
- Workloads that take advantage of GPUs' computing power:
  - Machine learning
  - Graph algorithm
  - Scientific computing

Introduction

Graphics Processing Unit (GPU) Architecture

Modern graphics processing unit (GPU) architectures feature a number of Stream Multiprocessor (SM) clusters, each of which consists of multiple SMs. [1]



Introduction

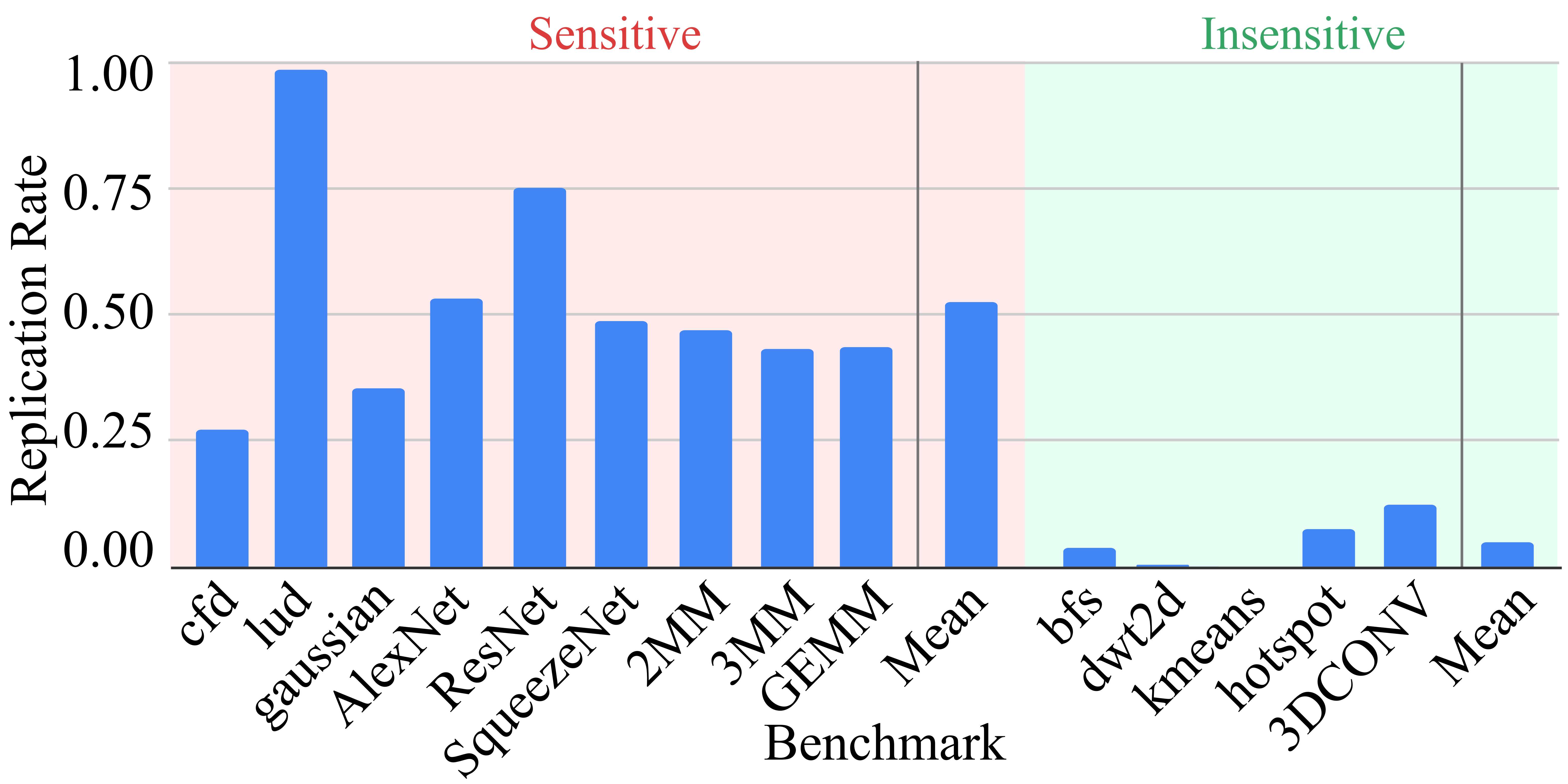
Motivation - The issue with private L1 caches

- Network-on-chip (NoC) congestion
  - The NoC fabric has become the performance bottleneck in GPUs [2]
  - NoC bandwidth is crucial for GPU performance
- Replicated cache requests:
  - Multiple SMs request for the same cache line
  - The cache line is fetched repeatedly across the NoC from L2
  - Inefficient usage of NoC bandwidth
- Replicated cache requests exacerbate NoC congestion, leading to performance degradation

Introduction

Motivation - Quantifying replicated cache requests

For replication-sensitive benchmarks, an average of 52% of cache misses could have been serviced by other SMs within the cluster.



Introduction

Aim - capturing replicated cache requests

- By capturing replicated cache requests and servicing them between SMs, we can:
  - Prevent replicated cache requests from being directed to L2 across NoC
  - Reduce NoC traffic
  - Ease NoC congestion
  - Reduce the number of stalled cycles caused by waiting for NoC
  - Improve overall GPU performance
- Given that 52% of requests can potentially be serviced within the cluster, the potential performance benefit is significant.

Introduction

Proposed method - COLAB

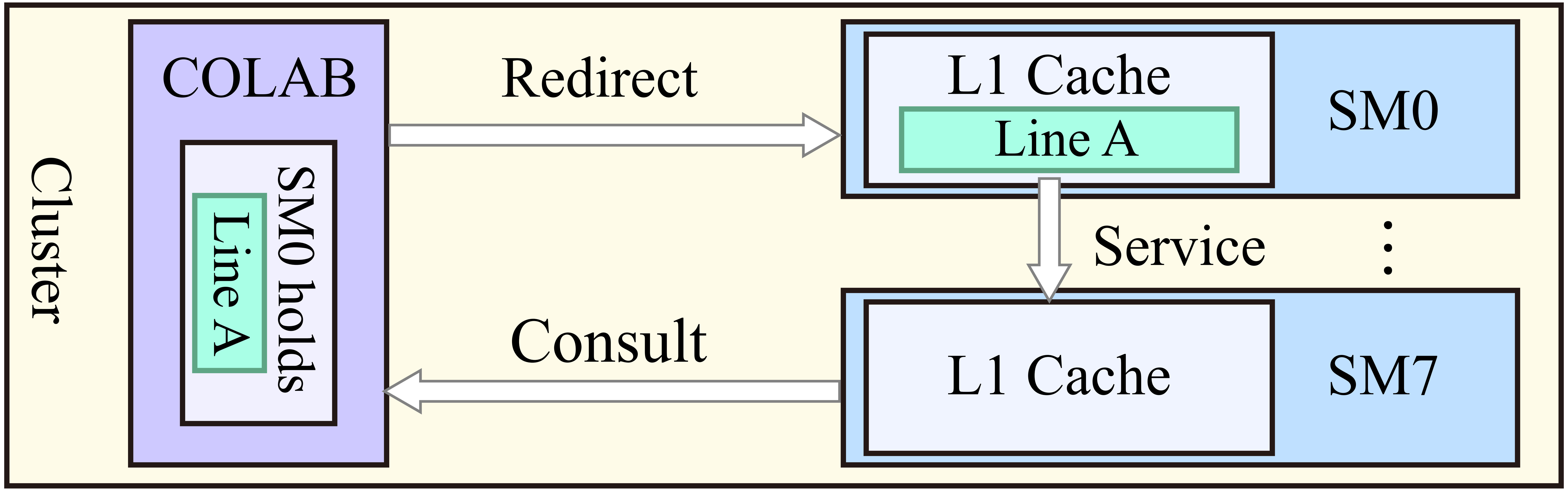
- In this work, we propose incorporating COLAB to achieve our goal of capturing replicated cache requests.
- It does so by keeping track of the ownership information of cache lines stored in the cluster.
- By consulting COLAB, a replaced cache request can know which SM within the cluster holds a copy of the requested line.
- COLAB can redirect the request to one of the SMs according to the information it stores

Methodology

Methodology

Overview

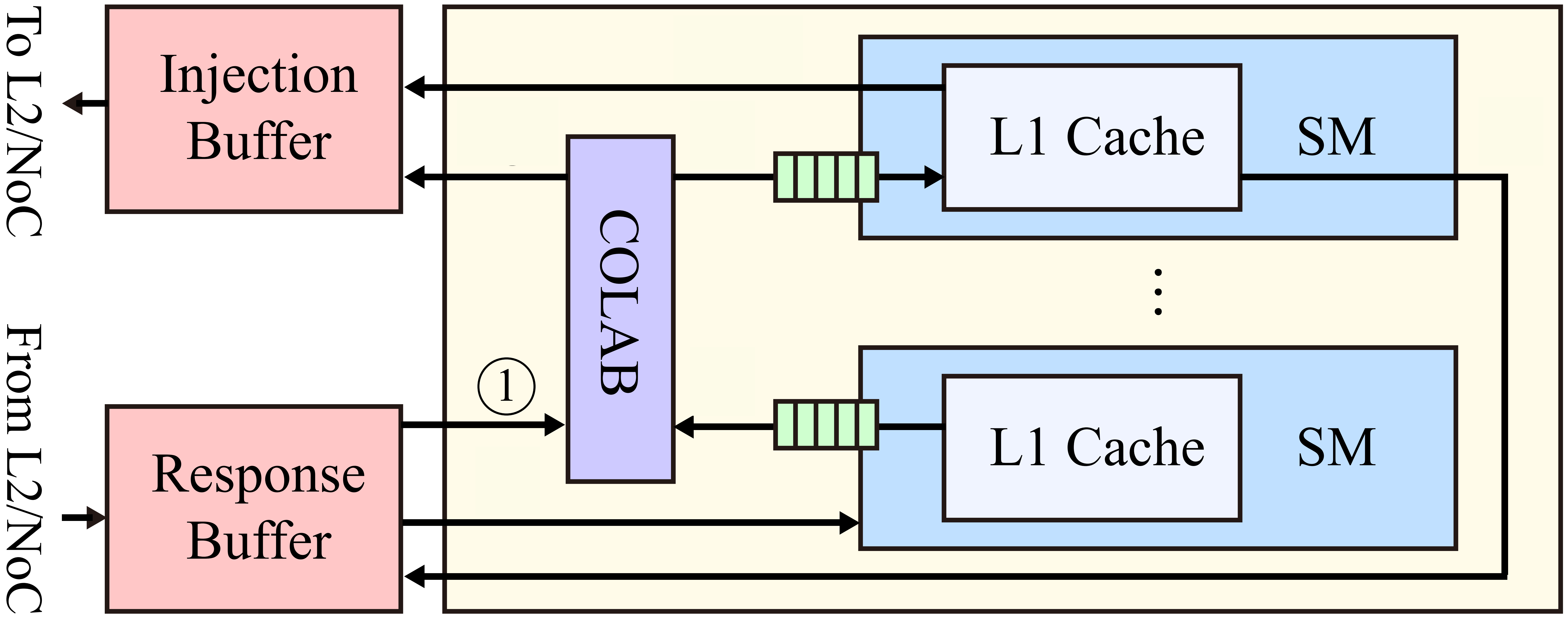
- **COLAB keeps track of the ownership information of cache lines**
- **By consulting COLAB, a cache request can be redirected.**



Methodology

Workflow - Updating COLAB

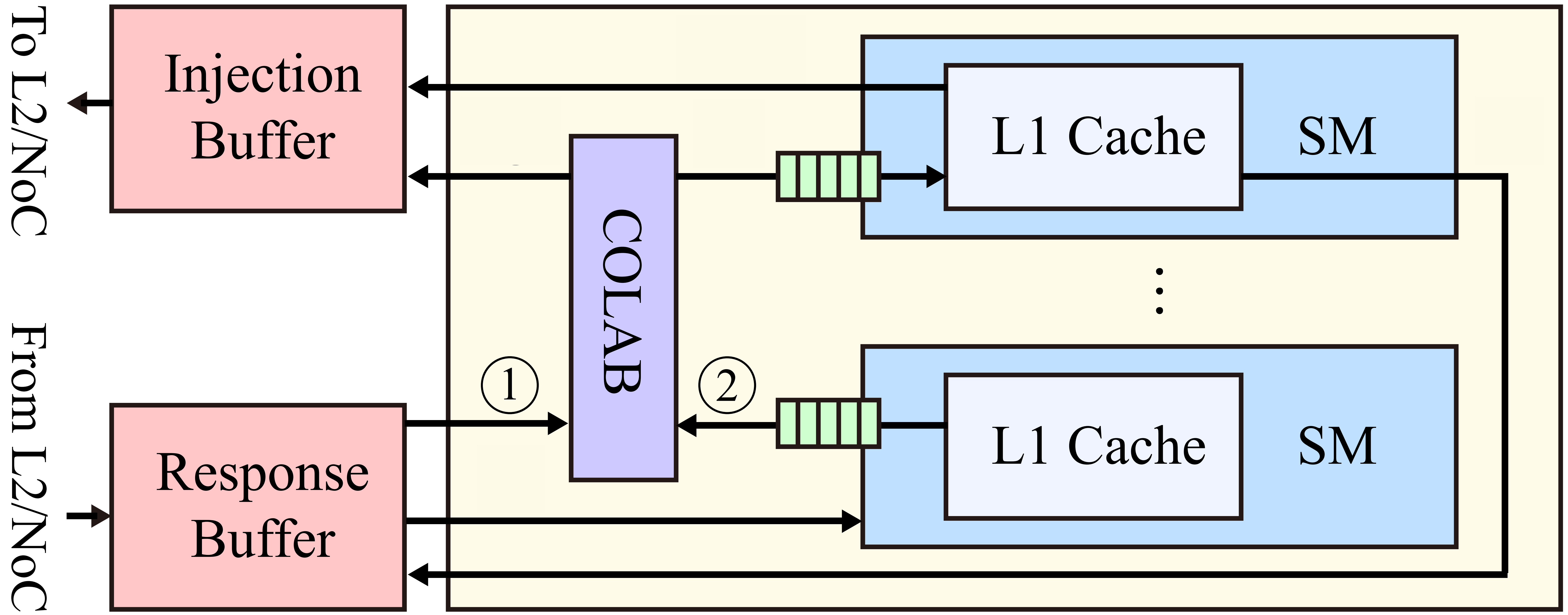
- As a cache request return from L2 via the NoC, COLAB is updated to associate the cache line with the requesting SM.



Methodology

Workflow - Accessing COLAB

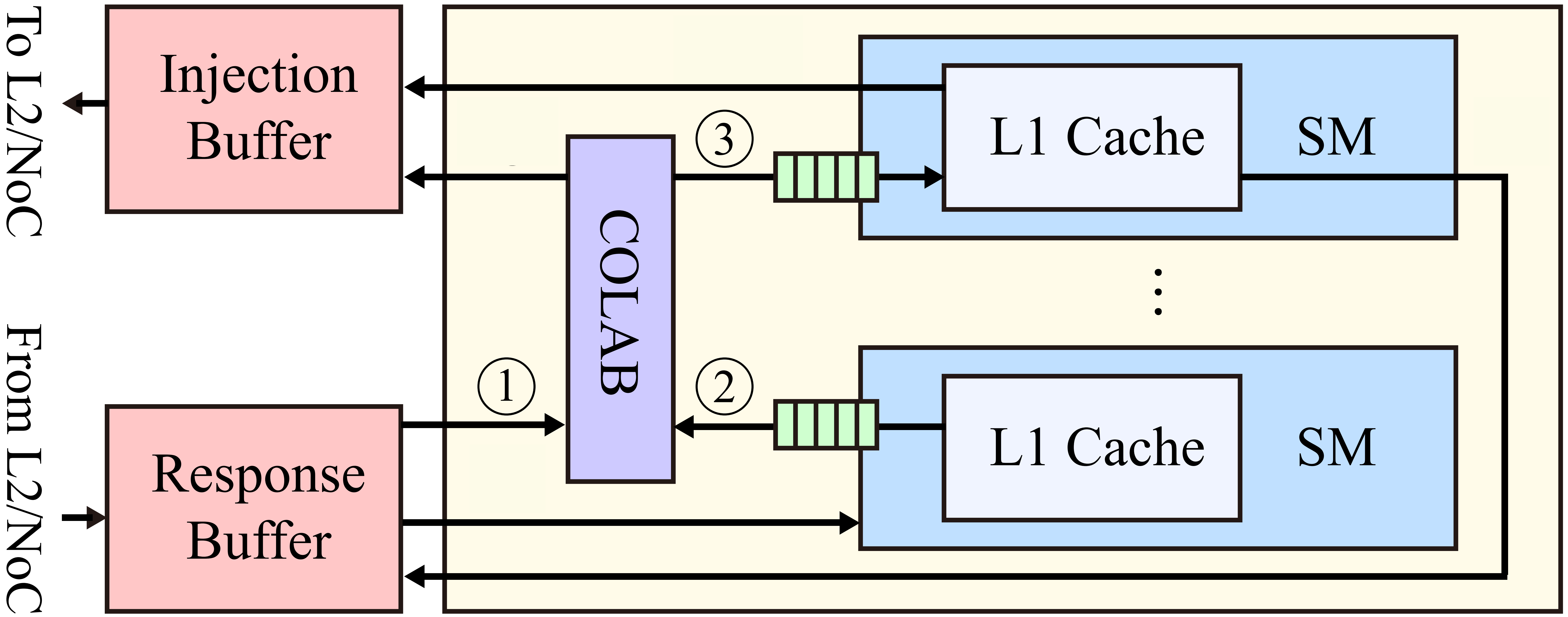
- As a cache request misses the L1 cache, it is sent to the input queue of COLAB to access COLAB.
- If input queue is full, the request is directed to the L2 cache via the NoC .



Methodology

Workflow - Redirect Request and L1 Access

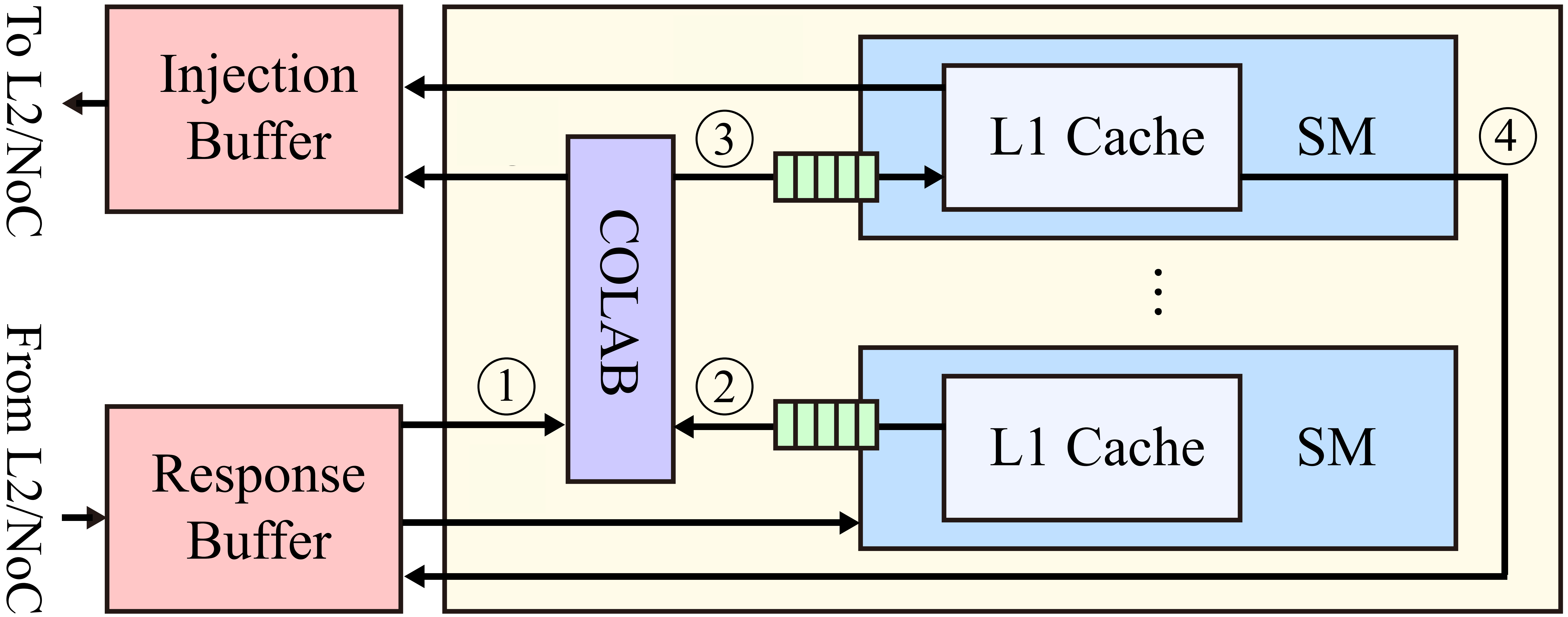
- If the request results in a hit in COLAB, it is directed to the access queue of the SM indicated by COLAB.
- A normal L1 lookup is performed.



Methodology

Workflow - Returning Requested Line

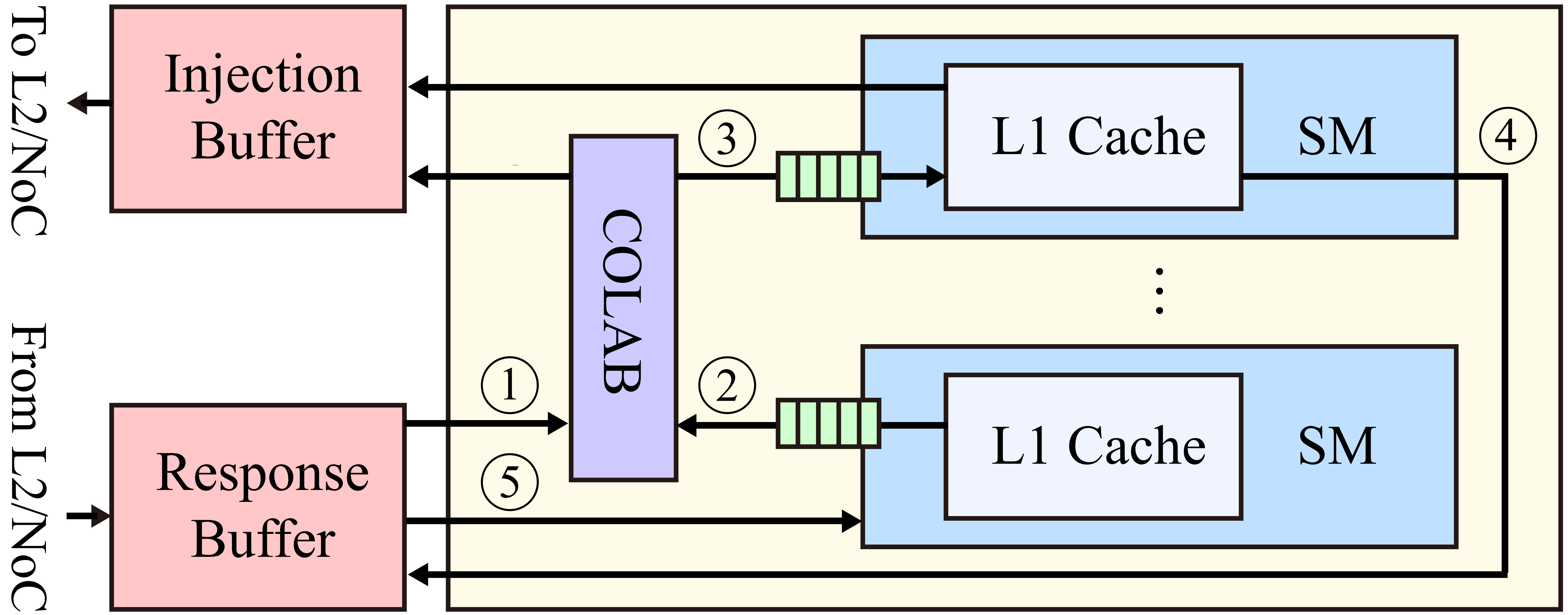
- **If the L1 lookup results in a hit, the requested cache line is pushed to the tail of the response buffer to be sent to the requesting SM.**



Methodology

Workflow - Service of Replicated Request

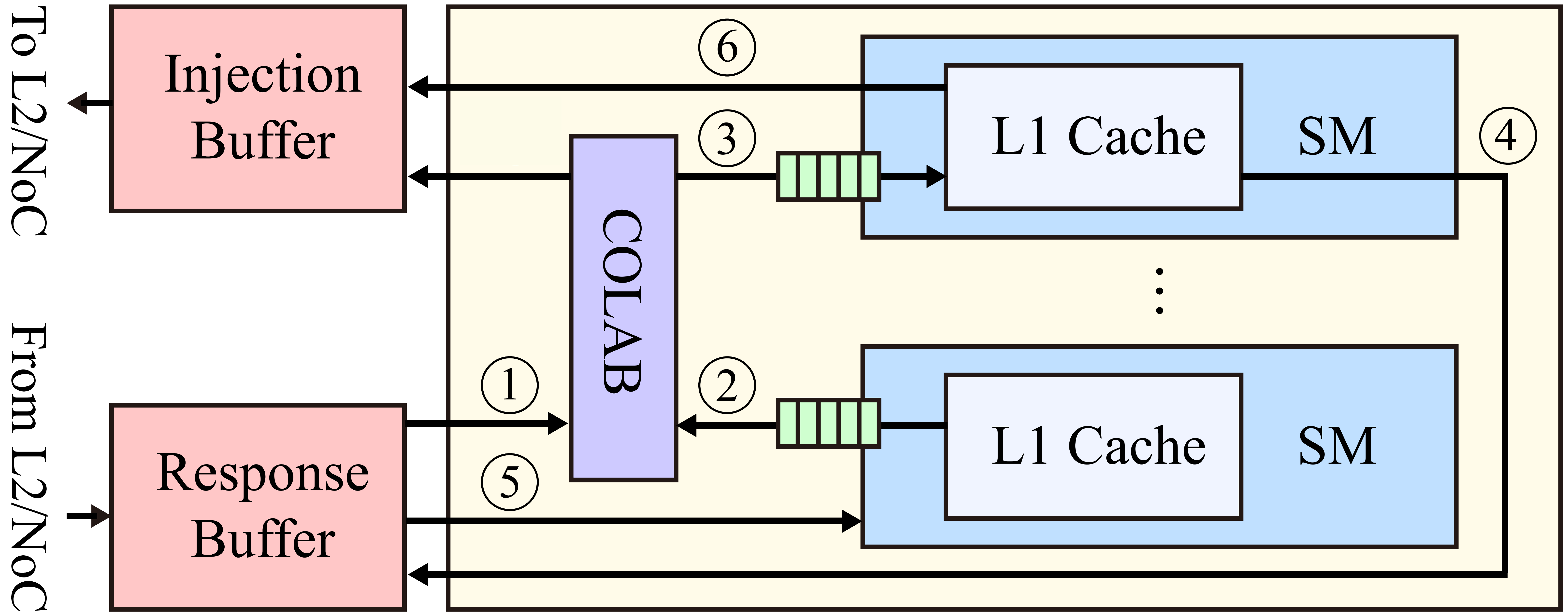
- As the request reaches the head of the response buffer, it is sent to the requesting SM, completing the inter-SM access process.
- The requested line is not cached by the requesting SM.



Methodology

Workflow - In Case of A L1 Miss

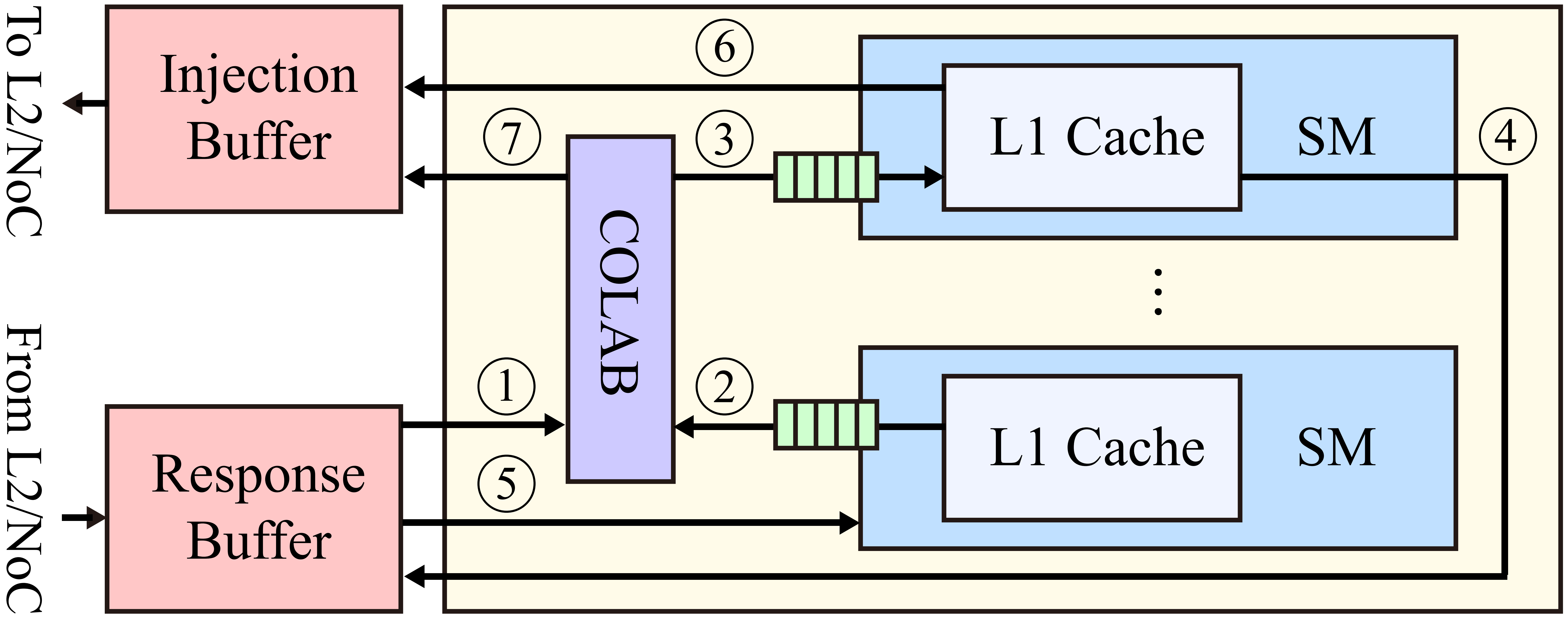
- **If the L1 access result in a miss due to outdated information in COLAB, the request is sent to the L2 cache via the NoC.**



Methodology

Workflow - In Case of A COLAB Miss

- **If the access to COLAB results in a miss or if the access queue of the targeted SM is full, the request is sent to the L2 cache via the NoC.**



Methodology

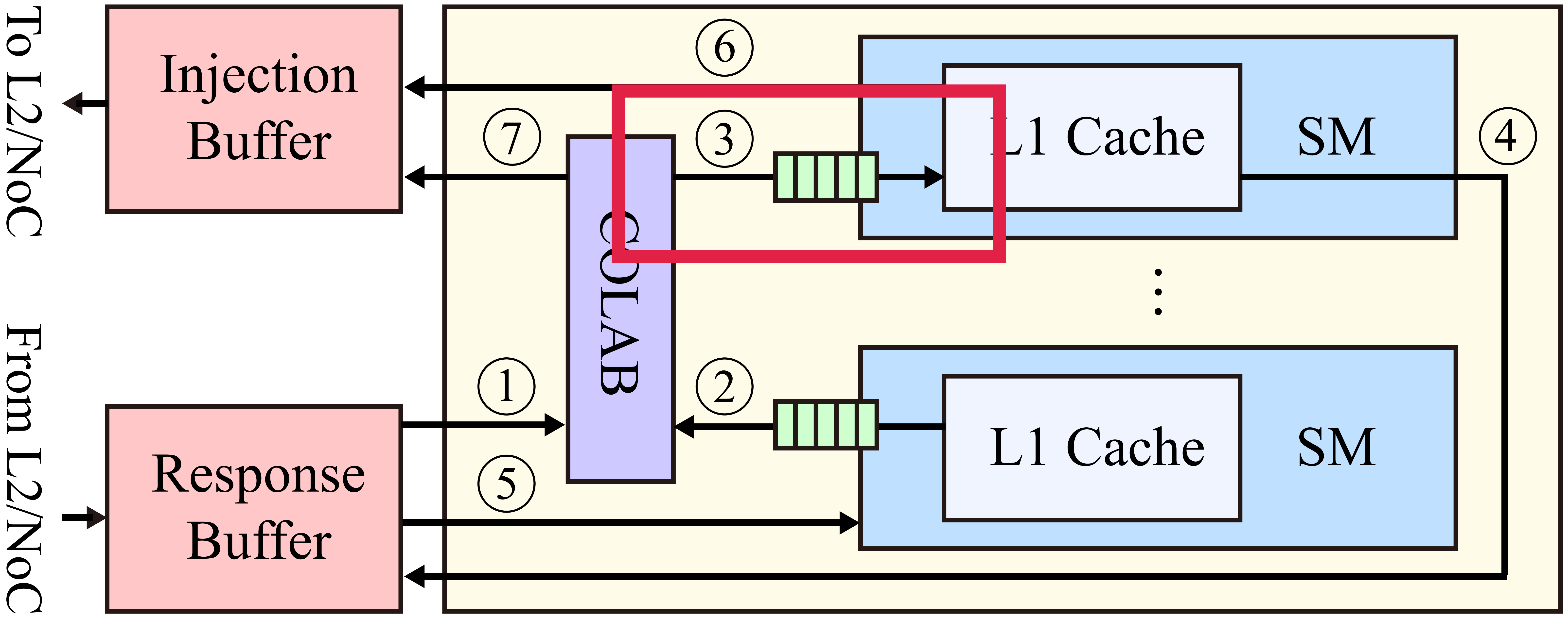
Workflow - False-Positive and False-Negative Errors

- **False-Negative:**
  - **An entry in COLAB may be evicted before its corresponding line is evicted from the L1 cache.**
  - **Miss opportunity.**
- **False-Positive:**
  - **COLAB does not update when a line is evicted from the L1 cache:**
  - **Add extra latency to cache request**
  - **Occupy L1 bandwidth**

Methodology

Workflow - Arbitration Policy between COLAB and Local L1 Requests

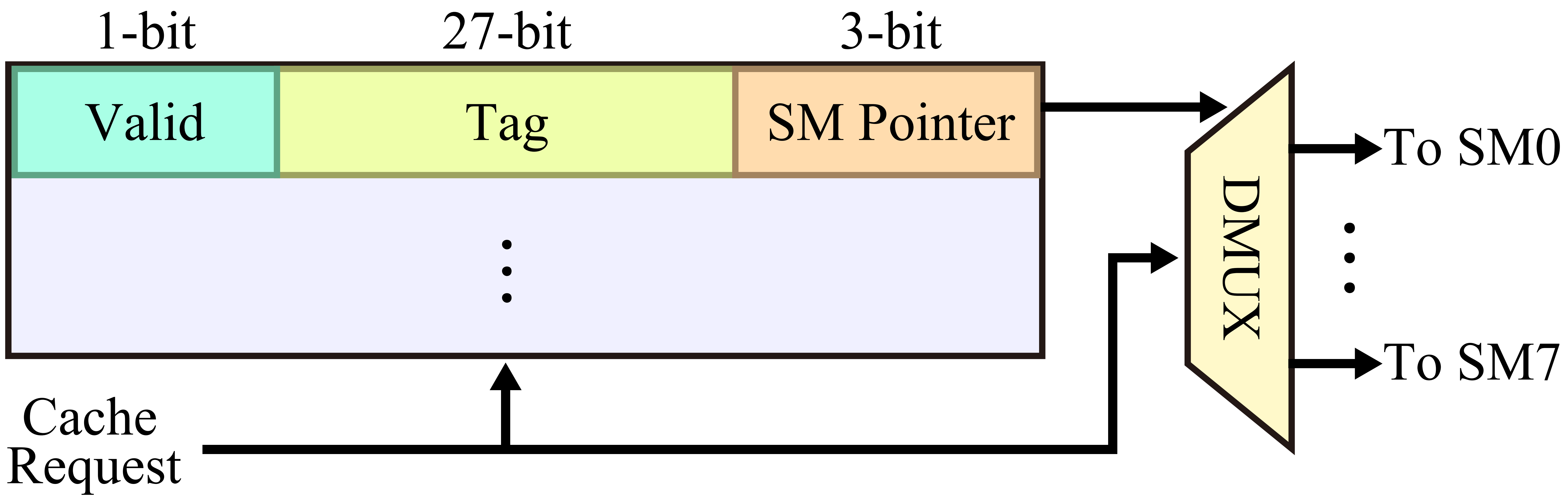
- **L1 cache access bandwidth is shared.**
- **We prioritize COLAB requests over L1 requests.**



Methodology

COLAB Organization

- **Similar to a set-associative cache**
- **Stores a SM pointer instead of the data line.**
- **Utilizes DMUX to route requests to SMs.**



Methodology

Hardware Overhead

- The number of entries in COLAB matches the total number of the L1 cache entries within a cluster.
- Each COLAB is equipped with a 32-entry input queue.
- Each SM is equipped with an 8-entry access queue.
- The total overhead is only 2% of the L1 cache capacity.

Module	# of entries	Size/Entry	Total size
COLAB	$64\,(set) \times 6\,(way) \times 8\,(SM) = 3,072$	31 bits	11.625KB
Queues	$32\,(input) + 8\,(access) \times 8\,(SM) = 96$	8 bytes	0.75KB

Experimental Results

Experimental Results

Experimental Setup - Tools

- The GPU architecture is simulated using GPGPU-sim [6]
- The power consumption and latency of COLAB are estimated conservatively using CACTI [7]
- The power consumption of the rest of the GPU components is calculated using GPUWattch [8]
- The benchmarks used in our experiments are selected from the Rodinia [9], Tango [10], and Polybench [11] benchmark suites

Experimental Results

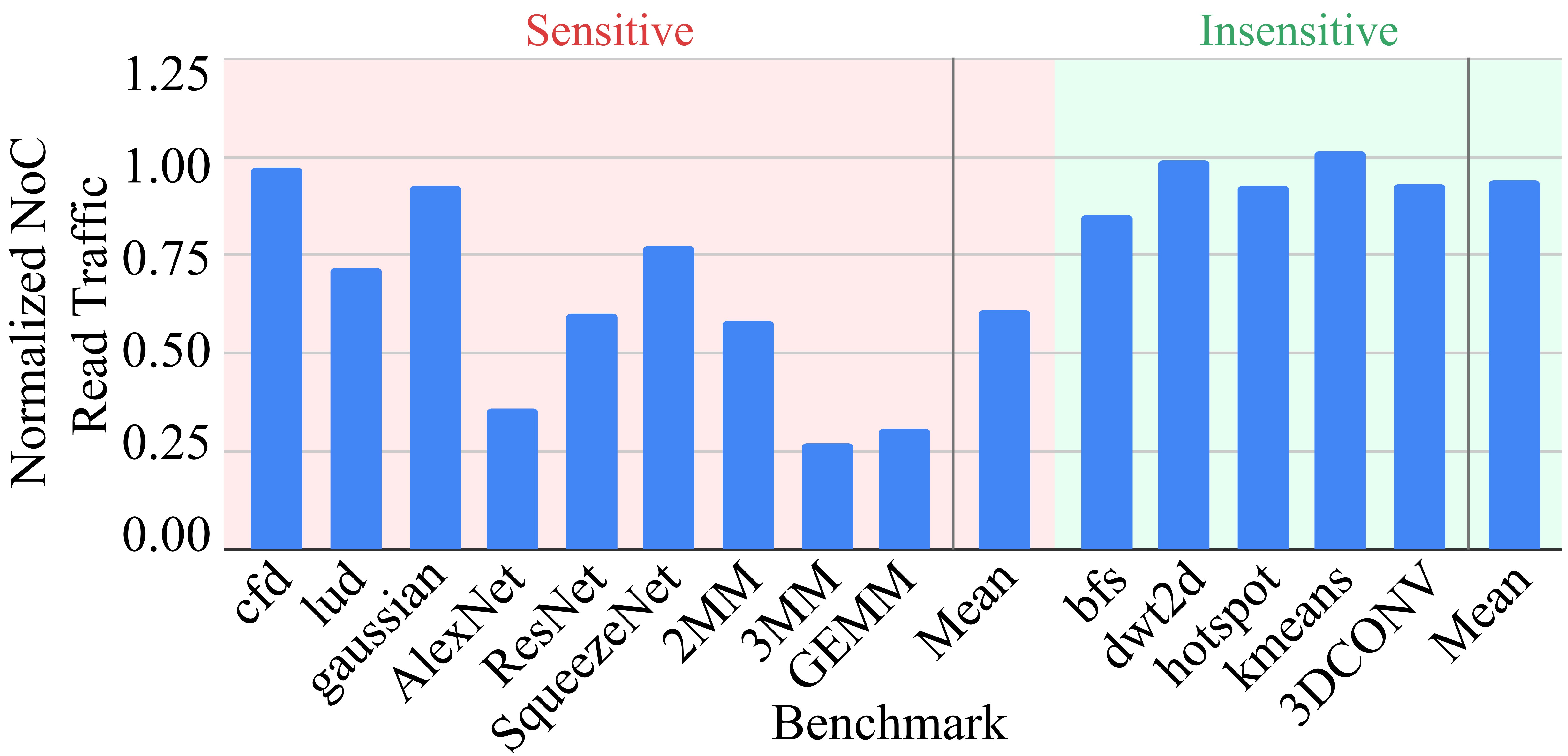
Experimental Setup - GPU Configuration

# of SMs	80, 8 per cluster @1,481MHz
Resources/SM	96KB shared memory, 64KB register, Max. 2048 thread Max. 32 thread blocks.
L1 Caches/SM	Data: 48KB, 6-way, latency=28 cycles, Texture: 48KB, 24-way. Constant: 12KB, 2-way. Instruction: 4KB, 4-way.
L2 Cache	128KB 16-way/Channel (2.75MB total), 128 bytes per line, Latency=120 cycles.
NoC	10×22 crossbar @2,962MHz
DRAM	11 partitions of GDDR5 @2,750MHz. Hynix GDDR5 timing
COLAB	Per-cluster 64-set 48-way lookup table latency=8 cycles

Experimental Results

Experimental Results - NoC Read Traffic Reduction

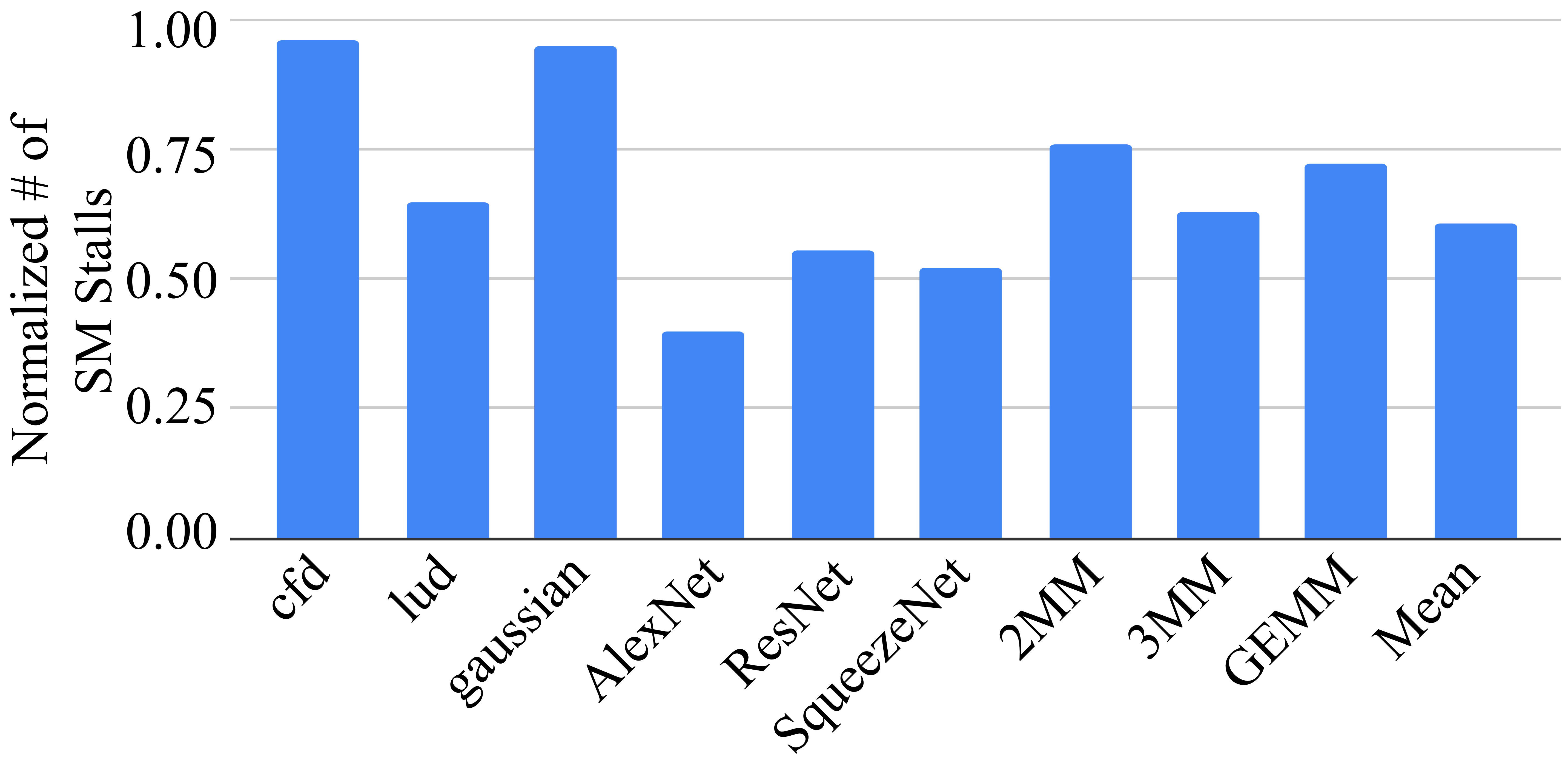
On average, the incorporation of COLAB can reduce 38% of NoC read traffic for replication-sensitive benchmarks by capturing replicated cache requests.



Experimental Results

Experimental Results - Stalls Reduction

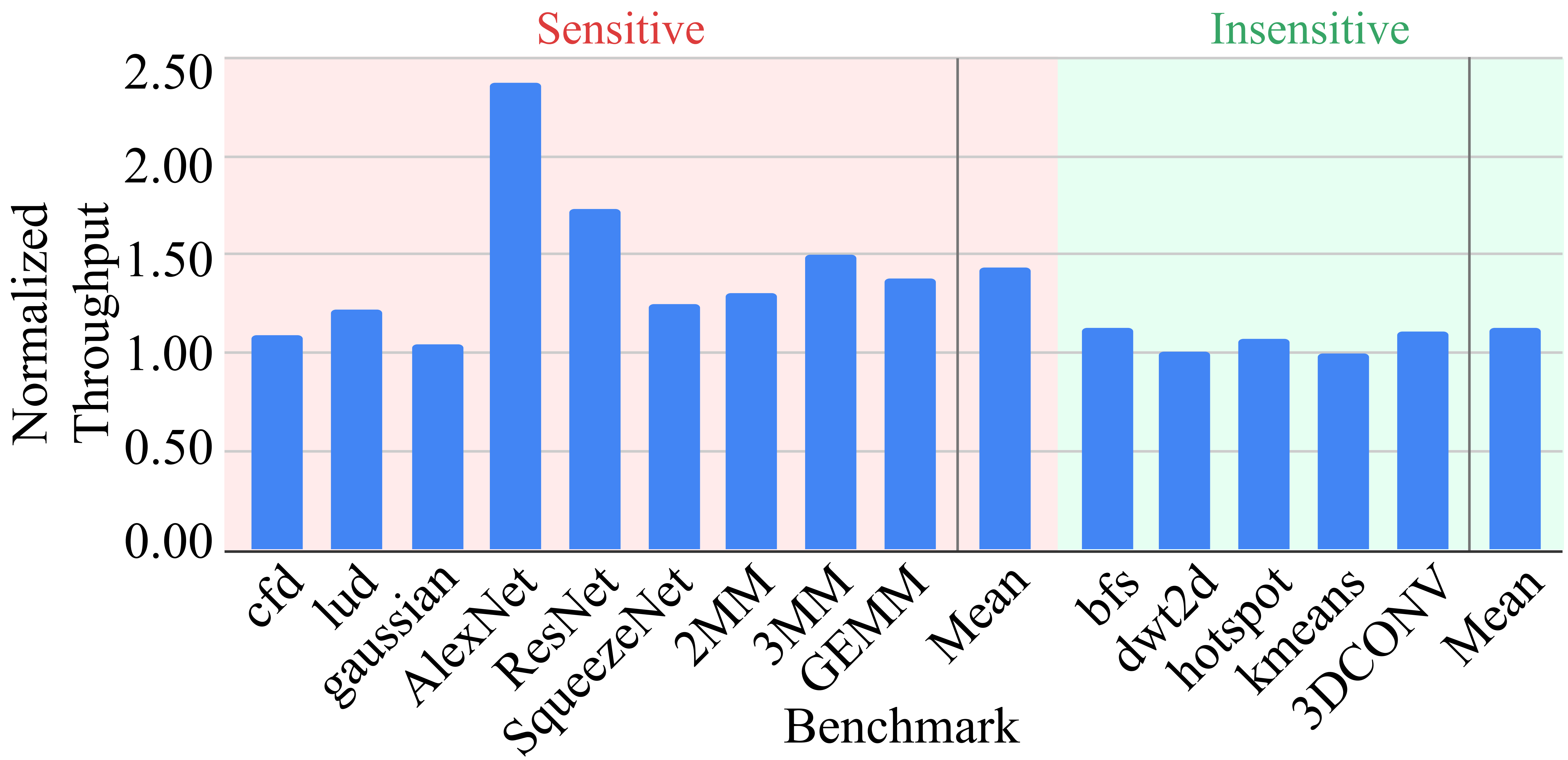
By reducing traffic, the number of stalled cycles due to NoC congestion is reduced by 40% for replication-sensitive benchmarks.



Experimental Results

Experimental Results - Performance Improvement

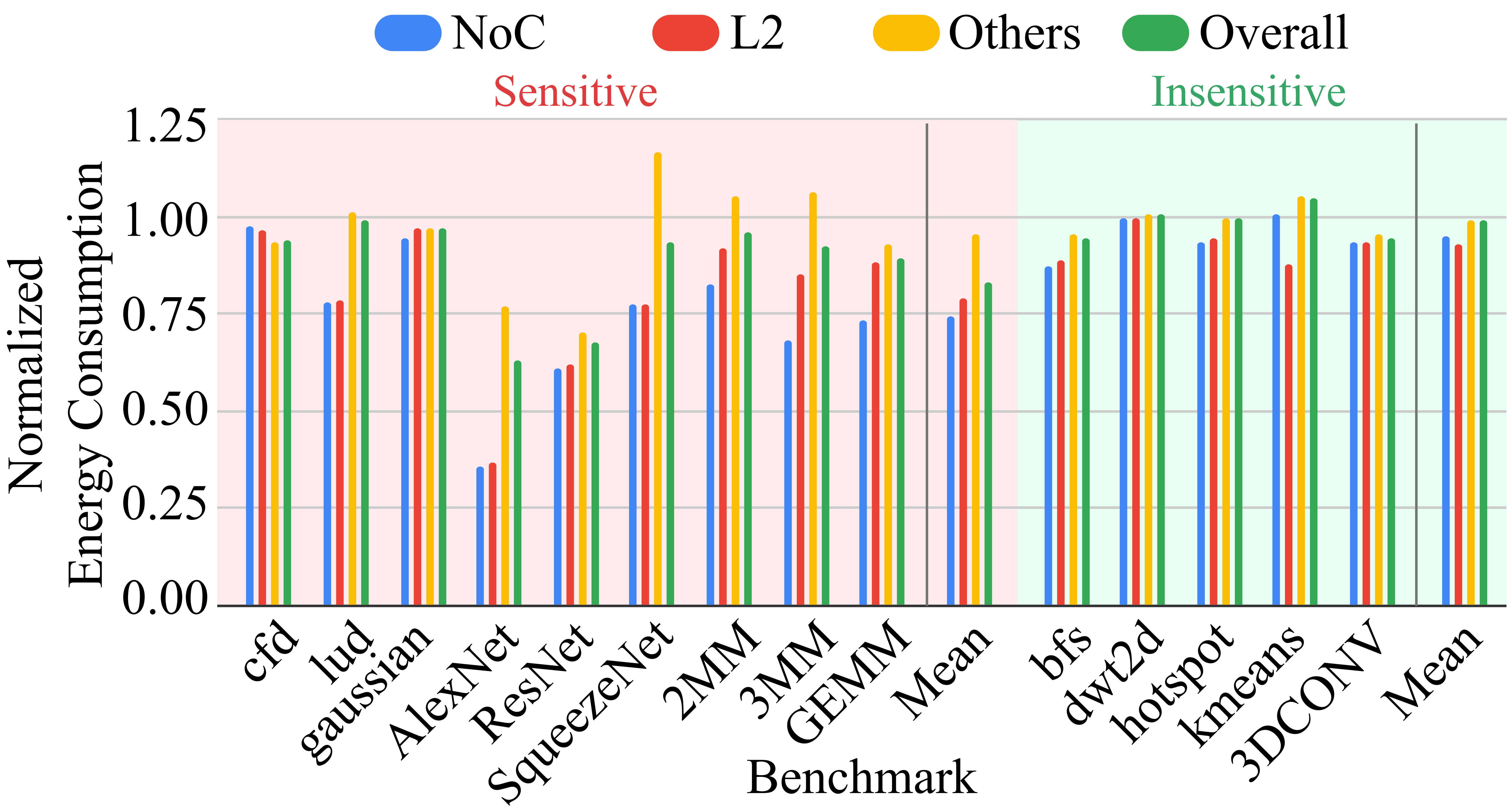
By reducing the number of stalled cycles, the computing throughput of the GPU is improved by an average of 43% for the replication-sensitive benchmarks.



Experimental Results

Experimental Results - Energy Evaluation

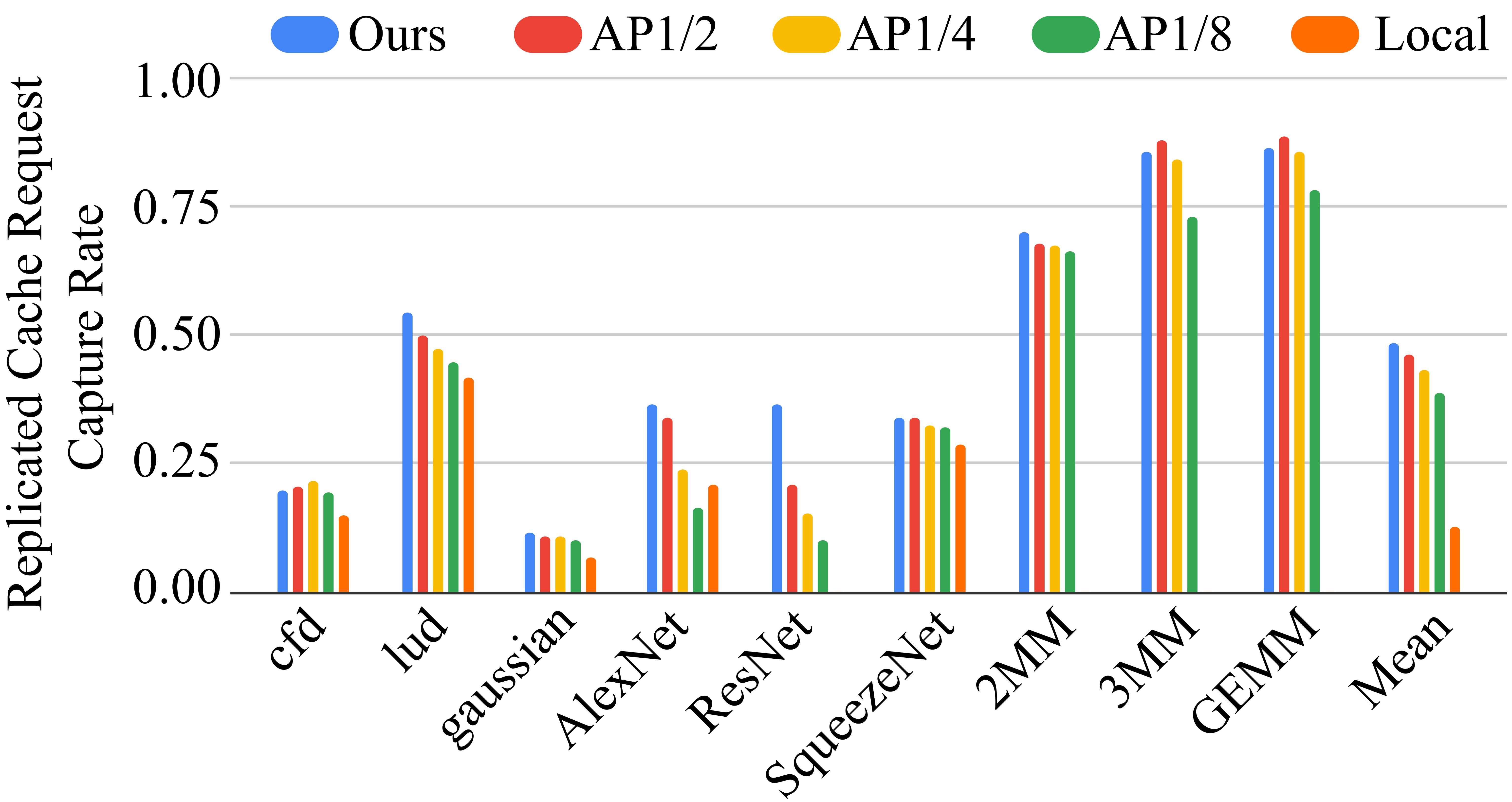
By reducing the NoC traffic and L2 accesses, COLAB can reduce the overall energy consumption of the replication-sensitive benchmark by 17%.



Experimental Results

Ablation Analysis - Arbitration Policy

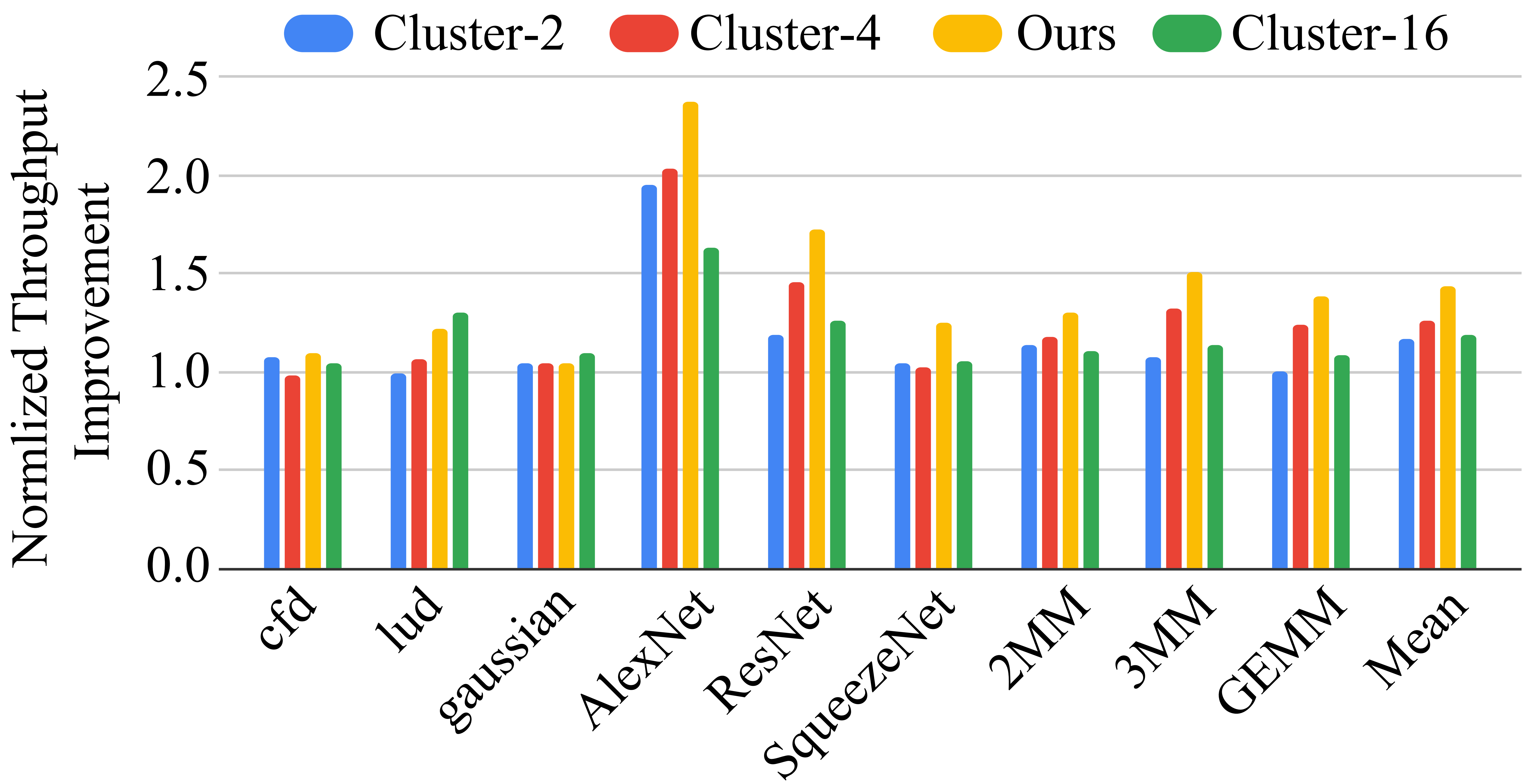
The proposed arbitration policy that prioritizes COLAB requests allows COLAB to capture the most replicated cache requests compared to other policies.



Experimental Results

Ablation Analysis - Cluster Size

COLAB is able to provide performance improvement regardless of the cluster size. However, the improvement is most significant when the cluster size is 8.



Conclusion

Conclusion

- We propose COLAB, an addition to the baseline GPU architecture that captures and redirects replicated cache requests within an SM cluster.
- By servicing replicated cache requests cooperatively within an SM cluster, COLAB is able to prevent replicated cache requests from entering the NoC network and consuming its precious bandwidth.
- Our experimental results demonstrate that COLAB can indeed reduce NoC read traffic and GPU stalled cycles and improve overall GPU performance while incurring limited hardware overhead.
- Our ablation analysis validates the design choices made while designing COLAB.

References

References

- [1] NVIDIA. 2020. NVIDIA AMPERE GA102 GPU ARCHITECTURE. Retrieved July 27, 2022 from https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu- architecture- whitepaper-v2.pdf
- [2] Saumay Dublish, Vijay Nagarajan, and Nigel Topham. 2016. Cooperative Caching for GPUs. ACM Trans. Archit. Code Optim. 13, 4, Article 39 (dec 2016), 25 pages.
- [3] Mohamed Assem Ibrahim, Hongyuan Liu, Onur Kayiran, and Adwait Jog. 2019. Analyzing and Leveraging Remote-Core Bandwidth for Enhanced Performance in GPUs. In 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT). 258–271.
- [4] Jianfei Wang, Li Jiang, Jing Ke, Xiaoyao Liang, and Naifeng Jing. 2019. A Sharing- Aware L1.5D Cache for Data Reuse in GPGPUs. In Proceedings of the 24th Asia and South Pacific Design Automation Conference (Tokyo, Japan) (ASPDAC ’19). Association for Computing Machinery, New York, NY, USA, 388–393.
- [5] Mohamed Assem Ibrahim, Onur Kayiran, Yasuko Eckert, Gabriel H. Loh, and Adwait Jog. 2021. Analyzing and Leveraging Decoupled L1 Caches in GPUs. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). 467–478.

References

- [6] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 473–486.
- [7] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. 2007. Op- timizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In 40th Annual IEEE/ACM International Symposium on Microarchitec- ture (MICRO 2007). 3–14.
- [8] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling Energy Optimizations in GPGPUs. In Proceedings of the 40th Annual International Symposium on Computer Architecture (Tel-Aviv, Israel) (ISCA '13). Association for Computing Machinery, New York, NY, USA, 487–498.
- [9] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang- Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In 2009 IEEE International Symposium on Workload Characterization (IISWC). 44–54.
- [10] Aajna Karki, Chethan Palangotu Keshava, Spoorthi Mysore Shivakumar, Joshua Skow, Goutam Madhukeshwar Hegde, and Hyeran Jeon. 2019. Detailed Charac- terization of Deep Neural Networks on GPUs and FPGAs. In Proceedings of the 12th Workshop on General Purpose Processing Using GPUs (Providence, RI, USA)
- [11] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. 2012. Auto-tuning a high-level language targeted to GPU codes. In 2012 Innovative Parallel Computing (InPar). 1–10.

Q & A