TUHH
Technische Universität Hamburg

Data Locality and its Limit in GPU Caches

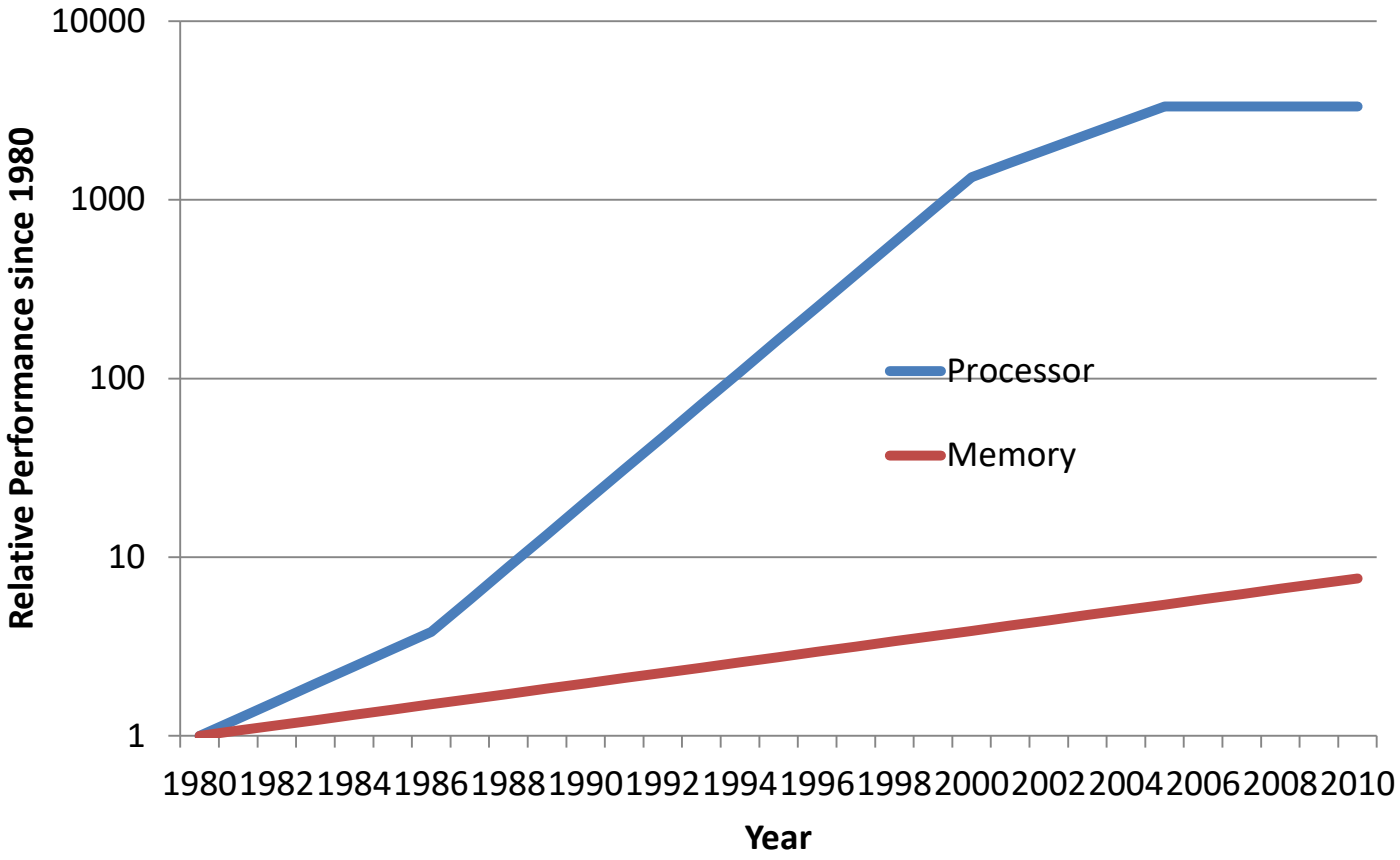
Prof. Dr.-Ing. Sohan Lal

Institute of Massively Parallel Systems
School of Electrical Engineering, Computer Science and Mathematics
Technische Universität Hamburg

Outline

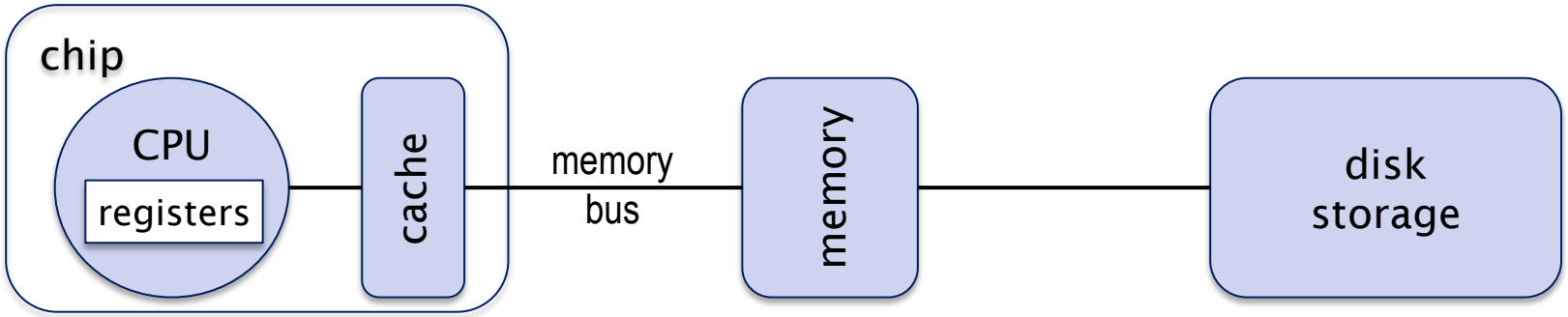
- Cache basics
- CPU vs. GPU caches
- Data over fetch due to memory divergence
- Quantitative study of locality in GPU caches
  - Temporal locality
  - Spatial locality
- Summary
- Sample Questions

Processor-memory Performance Gap



How to Solve?

- Use faster memory
  - SRAM instead of DRAM
  - But SRAM less dense and more costly than DRAM
- Use **memory hierarchy**
  - Place SRAM **cache** on-chip with execution units
  - Should contain code & data that is often used presently and in near future



Size:	1000B	64KB	4-16GB	4-16TB
Access time:	300ps	1ns	50-100ns	5-10ms

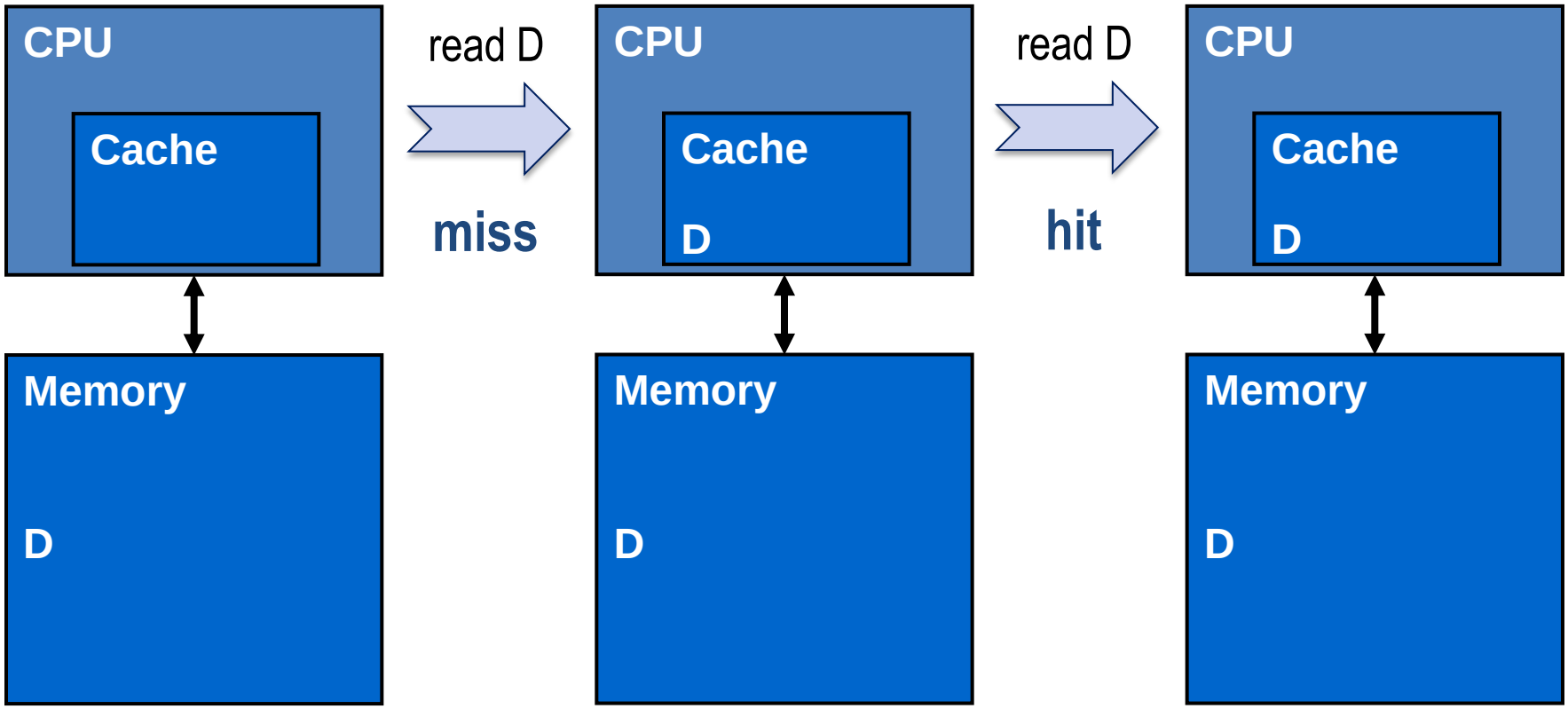
Principle of Locality

- Programs exhibit locality
  - **Temporal locality**: if code or data is referenced, likely that it will be needed again soon
    - e.g. code in a loop, scalar accesses inside a loop
  - **Spatial locality**: if code or data is referenced, likely that adjacent code or data will be needed soon
    - e.g. basic block, scanning an array

```
for (i=0; i<1000; i++)
   x[i] = x[i]+s;
```

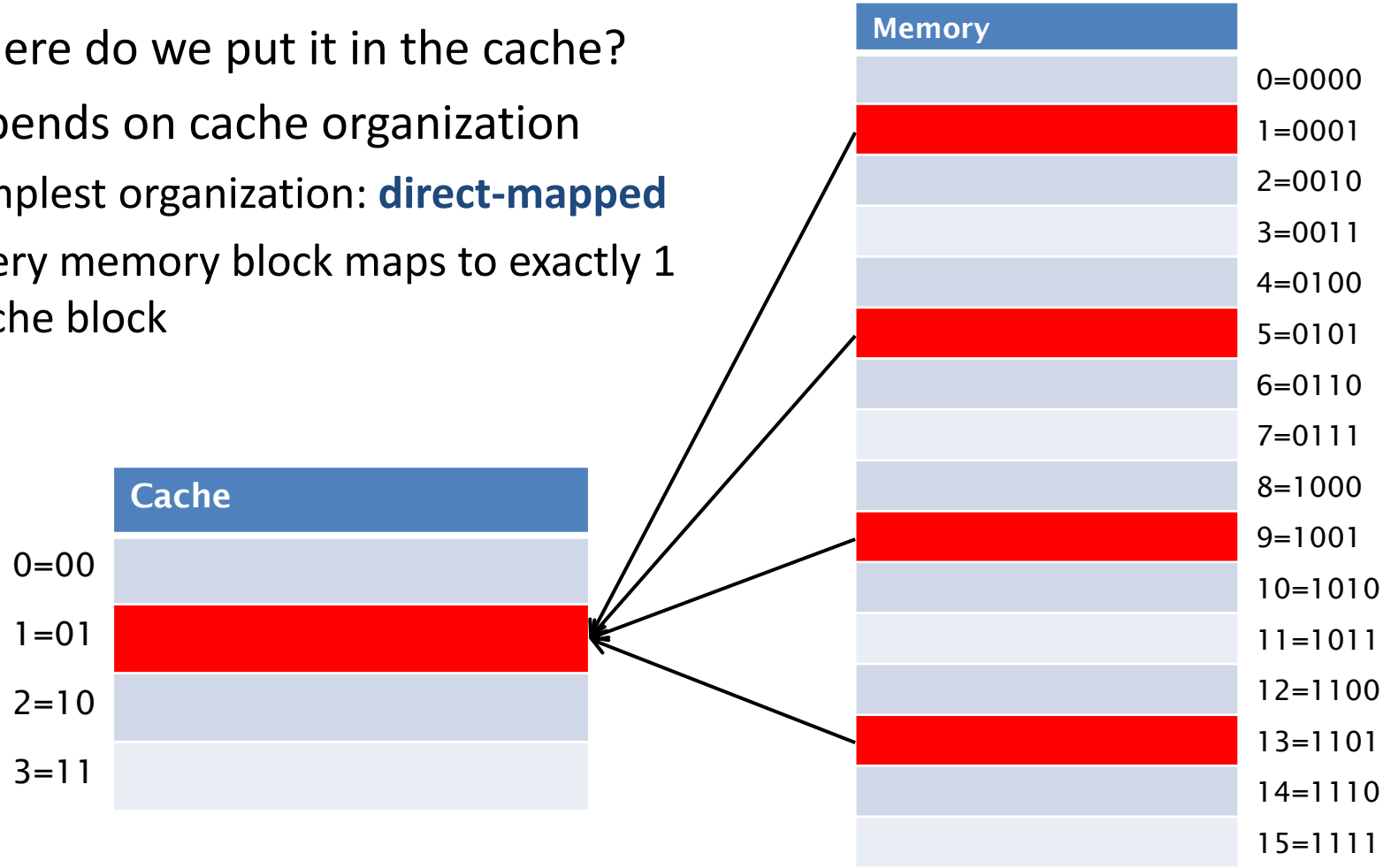
Top-level Answers

- Q: When do we bring contents of memory location into cache?
- A: On demand; when information not found in cache



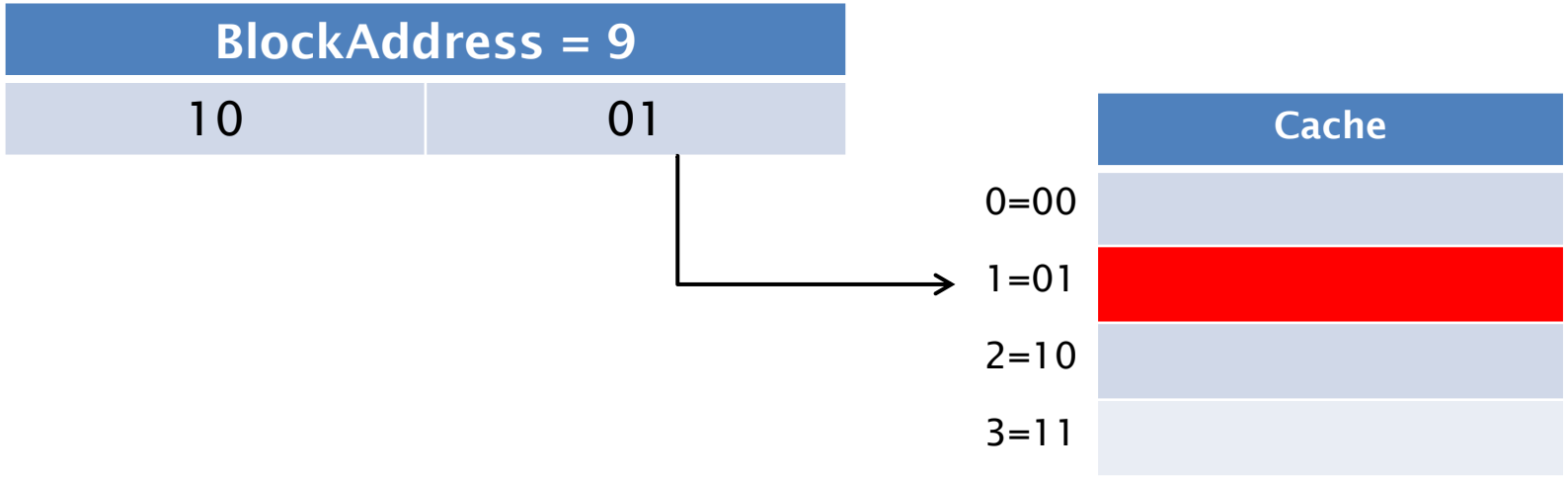
Top-level Answers

- Q: Where do we put it in the cache?
- A: Depends on cache organization
  - Simplest organization: **direct-mapped**
  - Every memory block maps to exactly 1 cache block



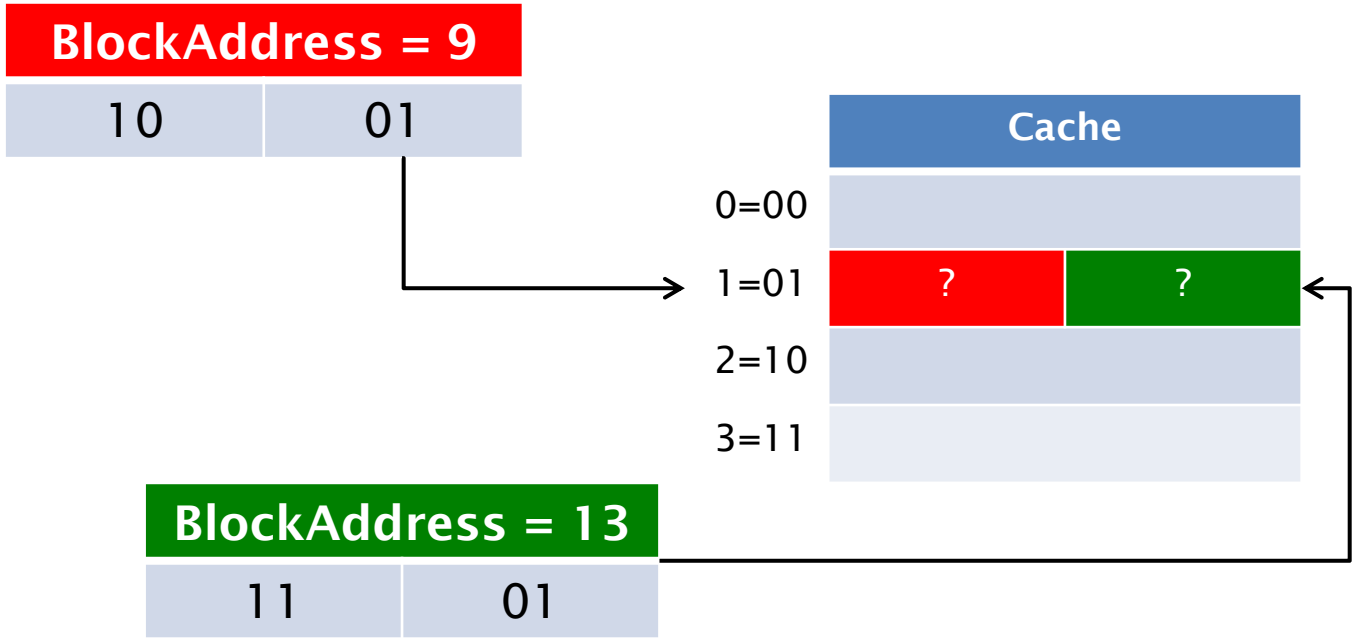
Direct-mapped Cache

- Direct-mapped cache uses mapping
  - CacheIndex = (BlockAddress) % (#CacheBlocks)
- If (#CacheBlocks) 2^k
  - CacheIndex k least significant bits of BlockAddress



Direct-mapped Cache

- How do we know it's there?
- How do we know requested block is in cache?



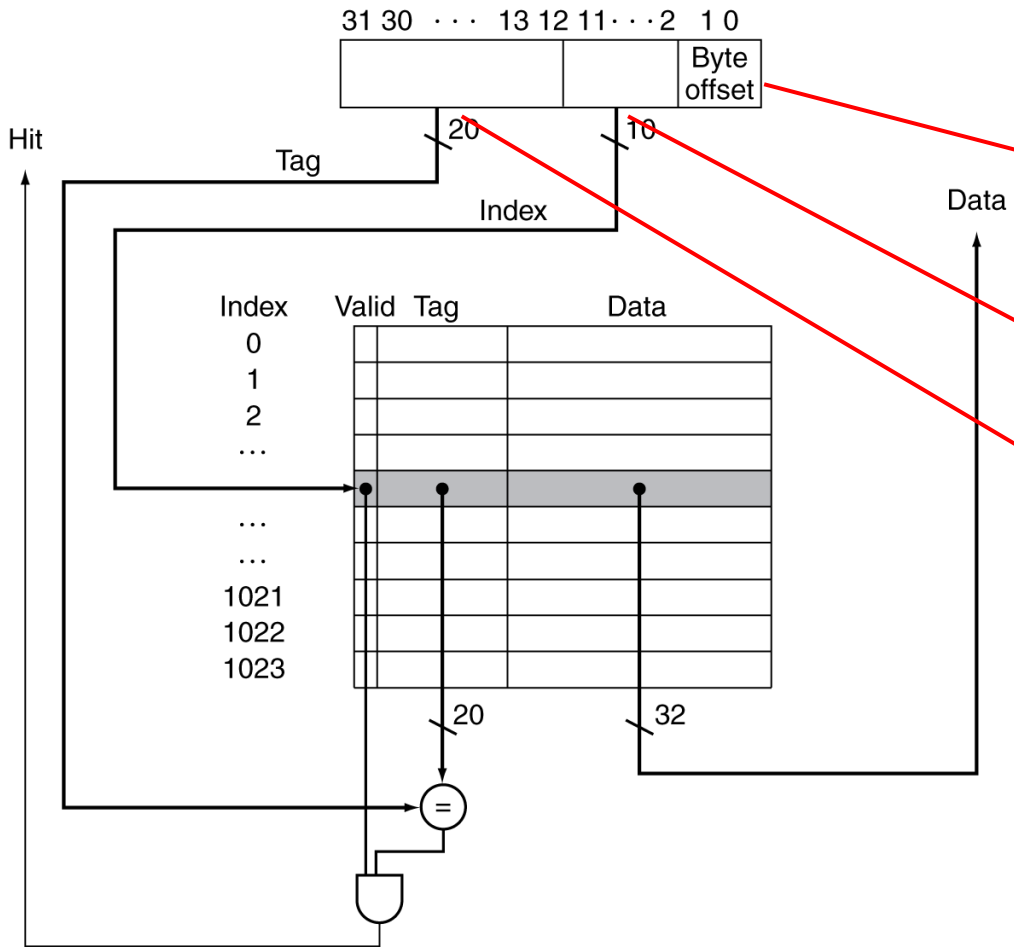
Sample Questions

- When do we bring memory block into cache?
  - On demand
- Where do we put it?
  - Direct-mapped: CacheIndex = BlockAddress % (#CacheBlocks)
  - (#CacheBlocks) = 2^k → CacheIndex = last k bits of BlockAddress
- How do we know cache block corresponds to requested memory block?
  - Cache block has tag
  - Tag part of BlockAddress should match Tag of cache block

Valid-bit

- Must know if cache block contains valid data
  - At power-on, all blocks invalid
  - At program start, all blocks invalid
- Every cache block has valid bit in addition to tag and data

Direct-Mapped Cache Organization Example



- Block size = 1 word = 4 bytes = 32 bits
  - Byte offset length = 2 bits
- # cache blocks = 2^{10} = 1024
  - Index length = 10 bits
- Tag length = address length – index length – byte offset = 32 – 10 – 2 = 20 bits

Cache Equations for DM Cache

- BlockAddress = ByteAddress / BytesPerBlock (/ = integer division)
- CacheIndex = BlockAddress % #CacheBlocks
- Tag = BlockAddress / #CacheBlocks

Hit or Miss?

- Processor generates byte addresses
  - 88 104 88 104 64 12 64 72
- It has DM cache with 4 blocks of 4 bytes
- For each access, hit or miss?

1. Compute block addresses (BlockAddress = ByteAddress / BlockSize)
2. Compute cache indices (CacheIndex = BlockAddress % #CacheBlocks)
3. Compute tags (Tag = BlockAddress / #CacheBlocks)

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	

Hit or Miss?

Cache before		
V	Tag	Data
0		
0		
0		
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m								

Cache after		
V	Tag	Data
0		
0		
1	5	MemBlock[22]
0		

Hit or Miss?

Cache		
V	Tag	Data
0		
0		
1	5	MemBlock[22]
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m							

Cache		
V	Tag	Data
0		
0		
1	6	MemBlock[26]
0		

Hit or Miss?

Cache		
V	Tag	Data
0		
0		
1	6	MemBlock[26]
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m						

Cache		
V	Tag	Data
0		
0		
1	5	MemBlock[22]
0		

Hit or Miss?

Cache		
V	Tag	Data
0		
0		
1	5	MemBlock[22]
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m	m					

Cache		
V	Tag	Data
0		
0		
1	6	MemBlock[26]
0		

Hit or Miss?

Cache		
V	Tag	Data
0		
0		
1	6	MemBlock[26]
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m	m	m				

Cache		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
0		

Hit or Miss?

Cache		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
0		

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m	m	m	m			

Cache		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
1	0	MemBlock[3]

Hit or Miss?

Cache before		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
1	0	MemBlock[3]

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m	m	m	m	h		

Cache after		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
1	0	MemBlock[3]

Hit or Miss?

Cache		
V	Tag	Data
1	4	MemBlock[16]
0		
1	6	MemBlock[26]
1	0	MemBlock[3]

Byte Address	88	104	88	104	64	12	64	72	
Block Address	22	26	22	26	16	3	16	18	
Cache Index	2	2	2	2	0	3	0	2	
Tag	5	6	5	6	4	0	4	4	
Hit or miss	m	m	m	m	m	m	h	m	

Cache		
V	Tag	Data
1	4	MemBlock[16]
0		
1	4	MemBlock[72]
1	0	MemBlock[3]

Cache Performance Metrics

- HitRate = #CacheHits / #CacheAccesses
- MissRate = #CacheMisses / #CacheAccesses = 1 – HitRate
- HitTime = time for a hit
- MissPenalty = cost of a miss
- AMAT = HitTime + MissRate x MissPenalty

Cache Miss Categories – 3 Cs Model

TUHH
Technische Universität Hamburg

- **Compulsory** — First access to a block is always a miss.
  - Also called cold start misses
  - Misses in infinite size cache
- **Capacity** — Cache cannot contain all blocks needed, capacity misses occur due to blocks being discarded and later retrieved
  - Misses in Fully Associative Cache with optimal replacement
- **Conflict** — Misses occurring because several blocks map to same set.
  - Also called collision misses.
  - All other misses

Improving Cache Performance

- AMAT = HitTime + MissRate x MissPenalty
- Reduce HitTime
  - Small and simple cache
- Reduce MissRate
  - Larger block size
  - Higher associativity
  - Larger cache
- Reduce MissPenalty
  - Multilevel caches
  - Give priority to read misses

Larger Block Size

- Blocks of 1 word do not exploit spatial locality
- More efficient to transfer n words than n single words
- Typical block size level-1 (L1) cache: 32B, 64B
- Larger block size reduces miss rate but increases miss penalty

Larger Block Size

TUHH
Technische Universität Hamburg

- Assumptions
  - HitTime = 1 cc
  - MissPenalty = 10 + (WordsPerBlock) cc
  - MissRate:

Block size	4B	8B	16B	32B	64B	128B
Miss Rate	10%	6%	4%	3%	2.5%	2.3%

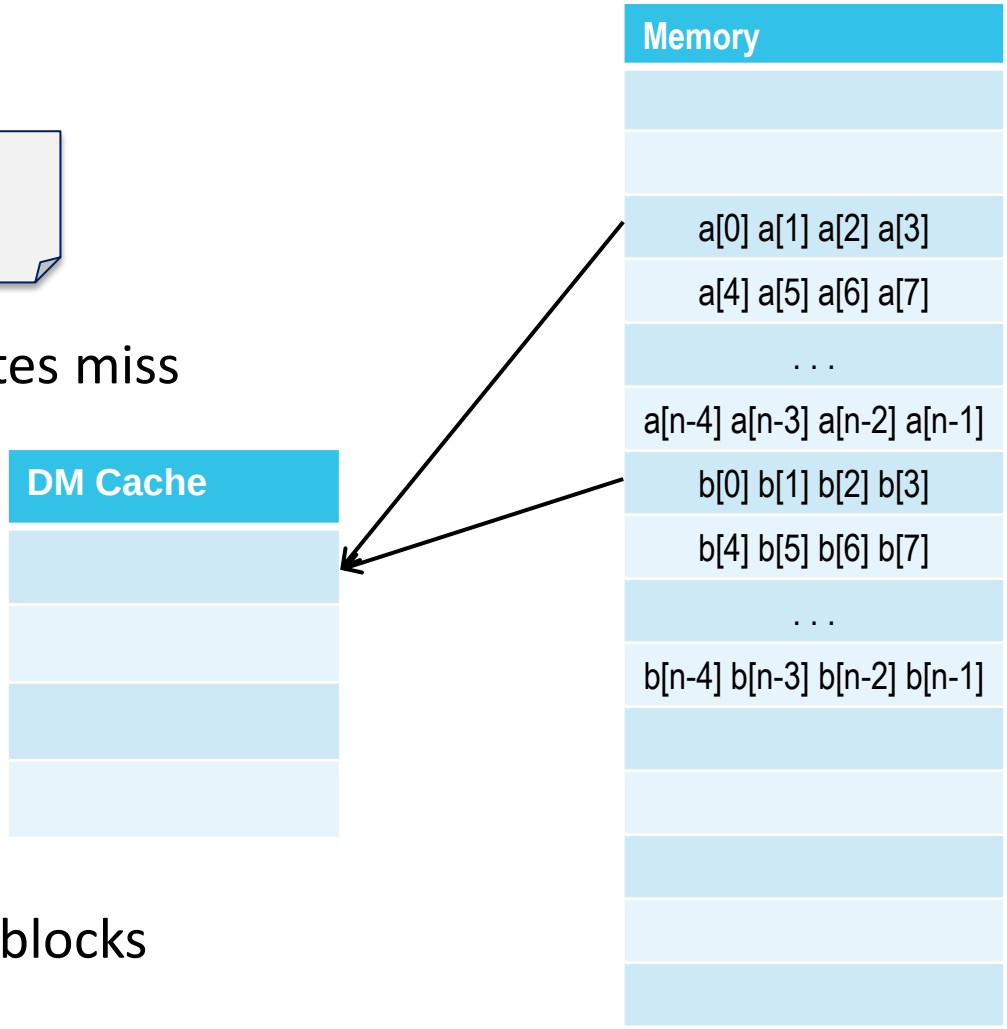
- Which block size has smallest AMAT?

Block size	4B	8B	16B	32B	64B	128B
Miss penalty	11	12	14	18	26	42
AMAT	2.1	1.72	1.56	**1.54**	1.65	1.97

Associative Caches - Motivation

```
for (i=0; i<n; i++)
   dotprod += a[i]*b[i];
```

- Every access to a and b generates miss



- Solution: flexible placement of blocks
  - associative caches

Possible Organizations of Cache w/ 4 Blocks

1-way set associative (= direct mapped)		
Set	Tag	Data
0		
1		
2		
3		

2-way set associative				
Set	Tag	Data	Tag	Data
0				
1				

4-way set associative (fully associative)								
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								

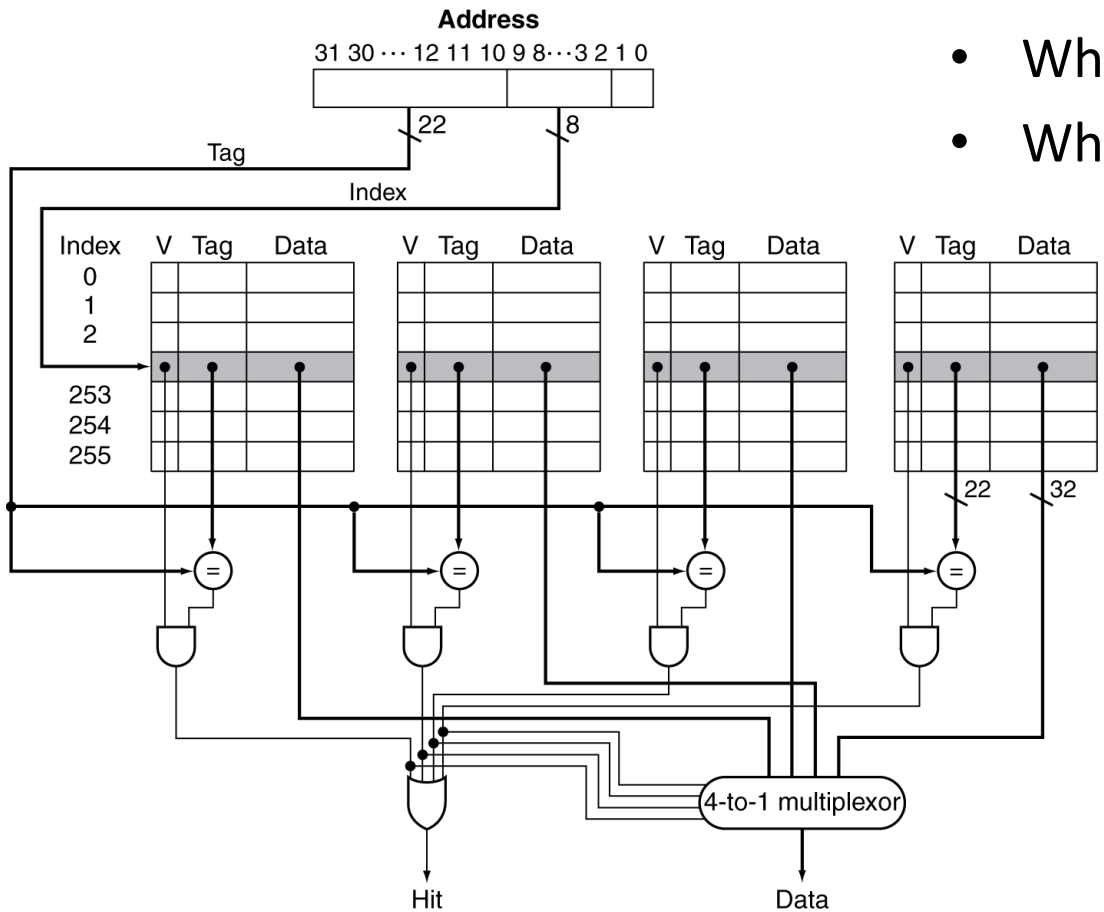
Cache Equations for Set-Associative Cache

- BlockAddress = ByteAddress / BytesPerBlock
- CacheIndex = BlockAddress % #Sets
  - DM cache → #Sets = #Blocks
  - Fully associative → #Sets = 1 → CacheIndex == 0
- Tag = BlockAddress / #Sets

- CacheSize = #Sets x #Ways x BytesPerBlock

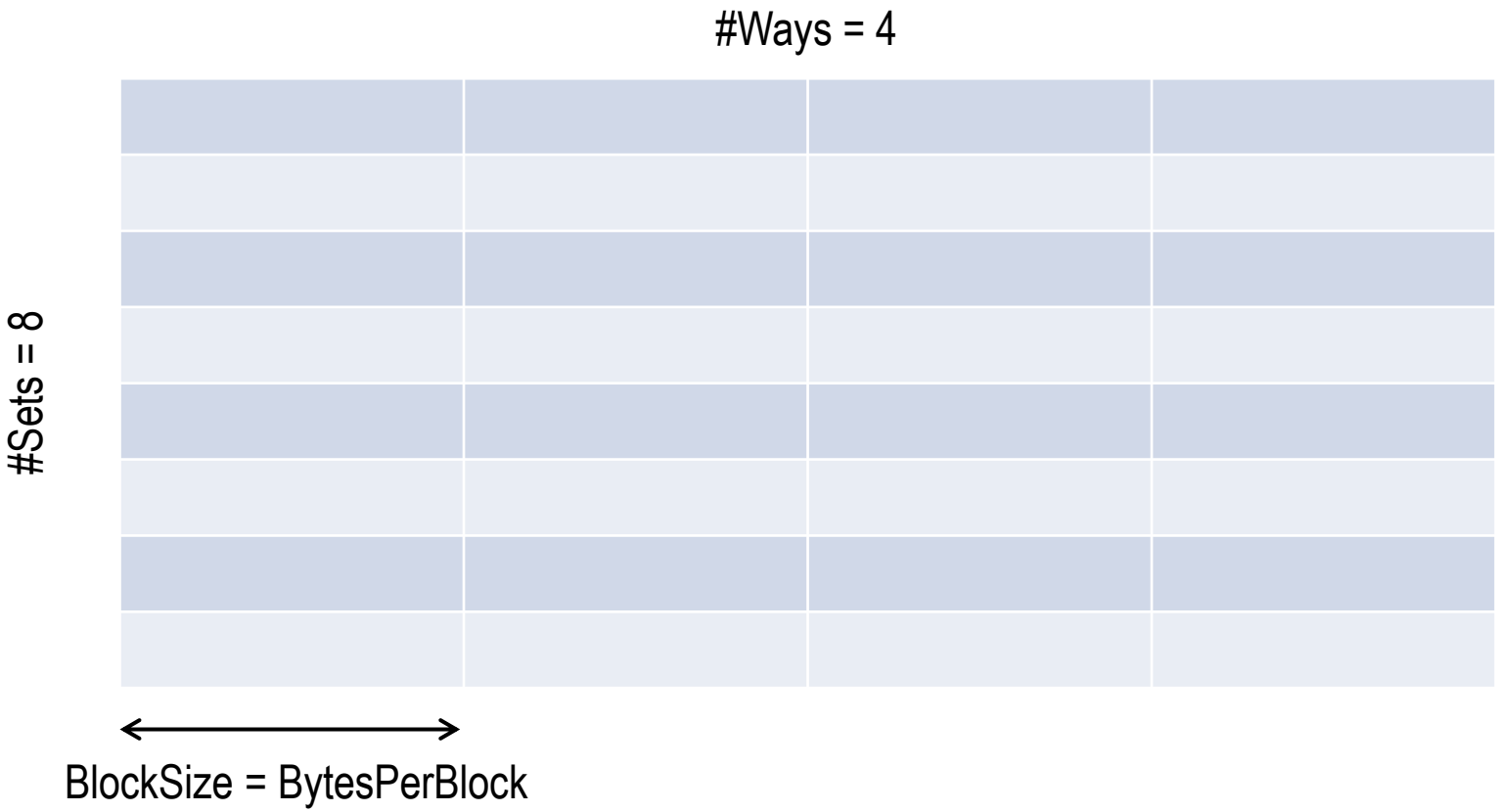
2-way set associative				
Set	Tag	Data	Tag	Data
0				
1				

4-Way Set-Associative Cache



- What is the block size?
- What is the size of this cache?

Cache Size = Cache Capacity

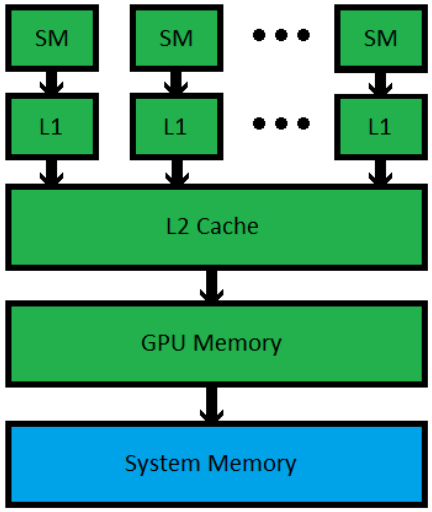


CacheSize = #Sets x #Ways x BytesPerBlock

Quantitative Study of data Locality in GPU Caches

Hardware-Managed Caches in GPUs

- First introduced in Fermi architecture
- Hardware-managed caches further accelerated use of GPUs
  - Caches reduce off-chip memory traffic
  - Cut-down pressure on memory bandwidth
- Using caches efficiently challenging
  - Caches shared by a large number of threads
  - Caches are small, especially per thread
  - Some works reported negative performance [2]
- Using caches efficiently even harder for **memory-divergent** workloads



Src: http://supercomputingblog.com/

CPU vs. GPU Caches

	CPU	GPU
Write policy	Write-back or write-through	L1 write-back for local accesses. Write-evict for global accesses.
Allocation policy	Write-allocate	No write-allocate for global accesses. Write-allocate for local accesses.
No. of levels	Typically 3	Typically 2
Per thread share	~KB - MB	A few bytes

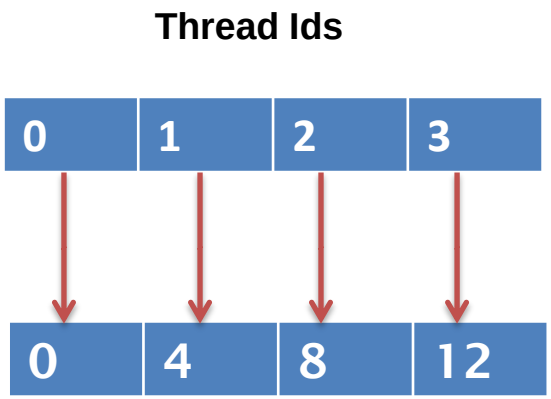
L1 and L2 Data Cache Size per Thread

Arch.	GPU/CPU	Max Threads (SM/GPU)	L1 (KB)	L2 (KB)	L1/Thread (B)	L2/Thread (B)
Tesla	GTX-8800	N/A	N/A	N/A	N/A	N/A
Fermi						32
Kepler						64
Maxwell	GTX-Titan X	2048/49152	24	2048	12	42.6
Pascal						
Volta						
Turing						
Ampere	RTX 3080	2048/139264	128	5120	64	37.6
Skylake	Intel Core i9-9900K	16	256	2048	16384	131072

- L3 cache is 16 MB i.e, 1MB/thread

- With only a few bytes per thread, it is extremely difficult to exploit locality for GPU caches

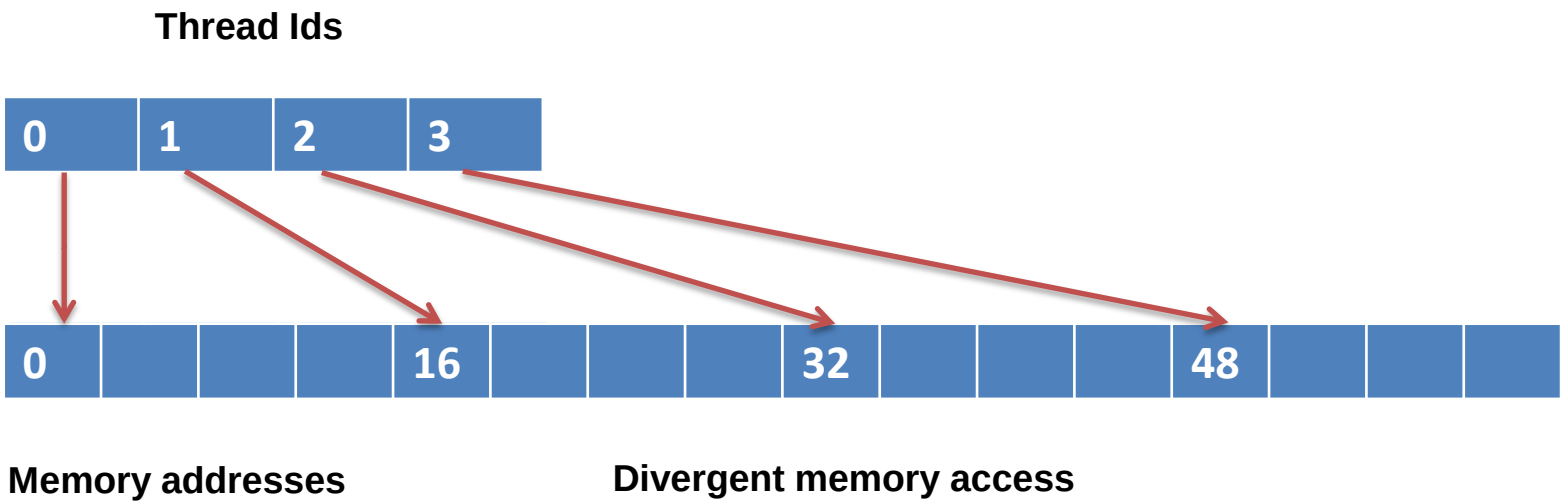
Memory Divergence Problem in GPUs (1/3)



Memory addresses

Coalesced memory access

Memory Divergence Problem in GPUs (1/3)



- Memory divergence causes data over-fetch
  - Wastage of **cache capacity** and **memory bandwidth**
- Memory divergence increases cache contention/thrashing
  - Massive multi-threading leads to cache contention
  - Divergence causes extra transactions (worst case one per thread)

Why Over-Fetch Data?

- GPUs use wider cache line size (typically 128 bytes)
- Caches help by exploiting temporal and spatial locality
-
- How much locality is exploited by GPUs?
- How much is lost due to cache thrashing?
- What is the limit of locality?

➢**We conduct quantitative study of locality in GPU caches**

Locality Classification

Locality Category	Description
Intra-warp	Locality from threads of the same warp
Inter-warp	Locality from different warps
Intra-CTA	Locality from warps of the same CTA
Inter-CTA	Locality from different CTAs

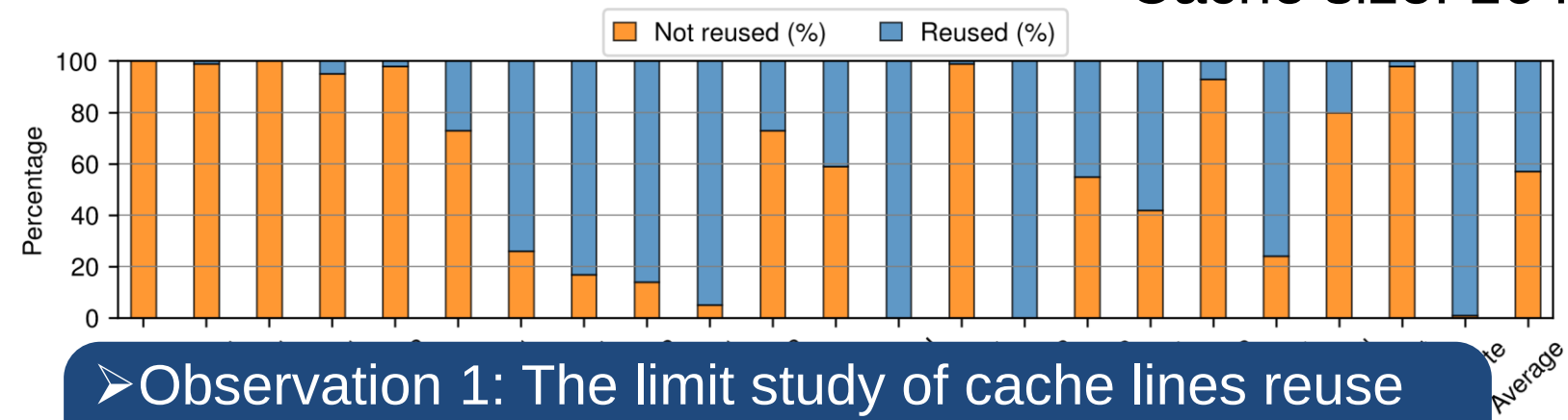
Experimental Setup

TUHH
Technische Universität Hamburg

- gpgpu-sim cycle accurate simulator
  - Configured to model a NVIDIA GPU
  - Modified to report several counters such as L1/L2 cache accesses, warp id, CTA id, bytes accesses within a cache line
- Benchmarks
  - Memory-divergent benchmarks from various domains such as data analytics, image processing, graph analytics, machine learning

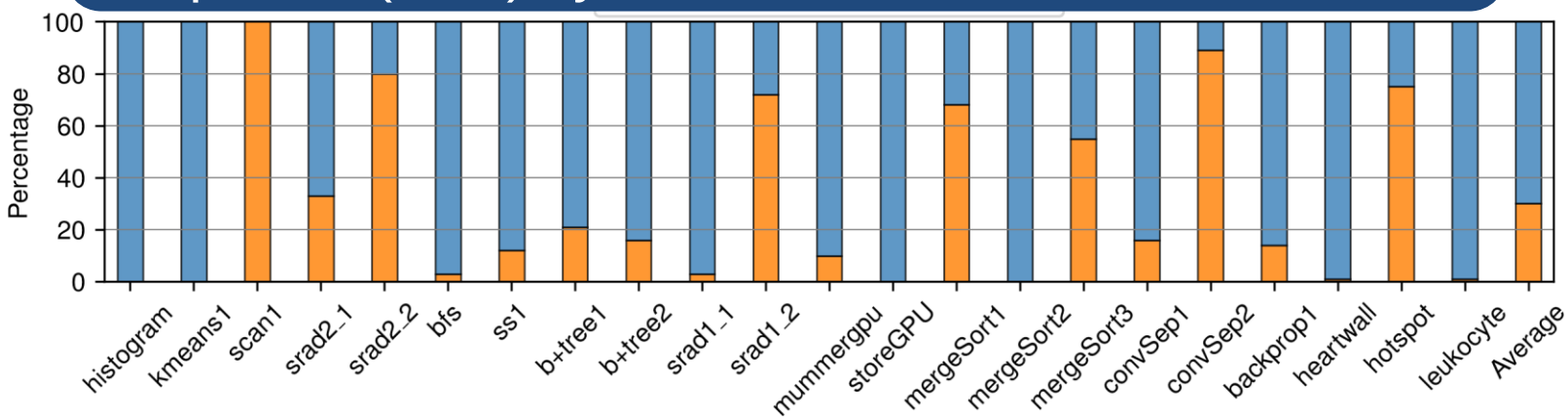
GPU L1 Data Cache line Reuse

Cache size: 16 KB

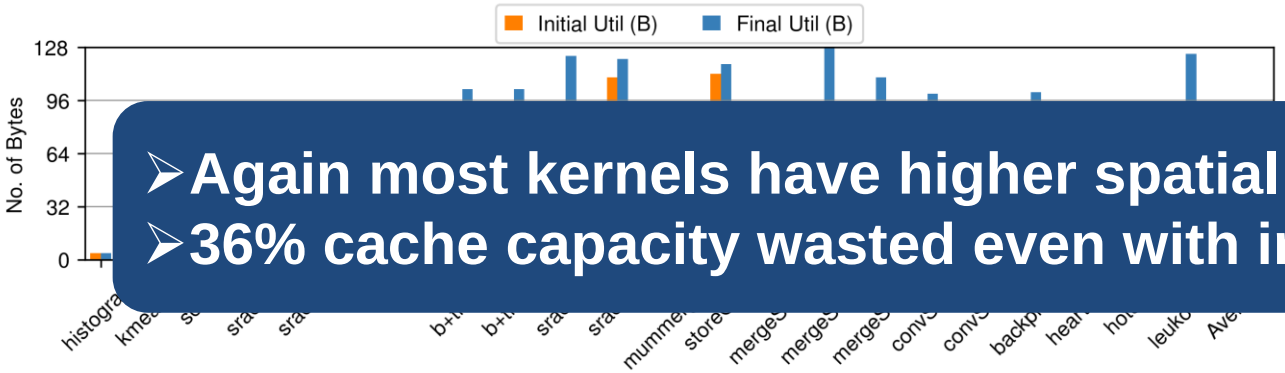


- Observation 1: The limit study of cache lines reuse shows a much higher reuse (70%) than currently exploited (43%) by GPUs.

nite



L1 Cache Spatial Utilization



Default cache size (16KB)

il 46 bytes

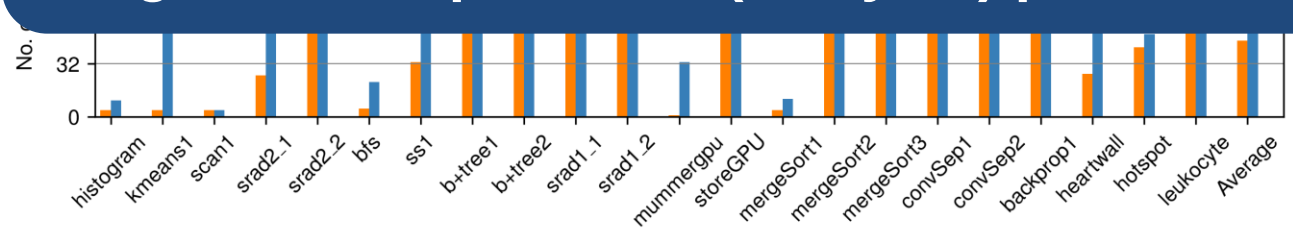
65 bytes

city, other

es wasted

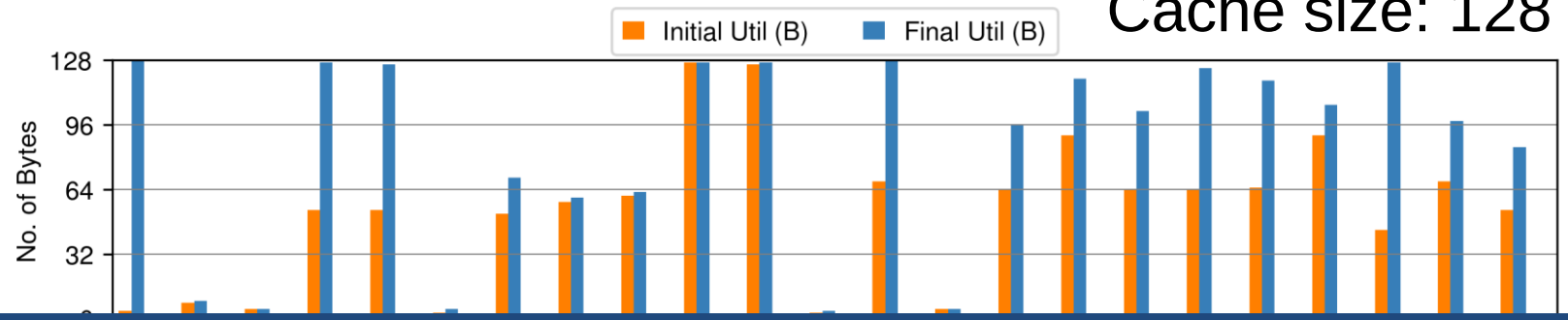
- Again most kernels have higher spatial locality
- 36% cache capacity wasted even with infinite size

- Low initial spatial util (46 bytes) justifies demand-fetched caches in Maxwell, Pascal, Volta, etc.
- Higher final spatial util (81 bytes) presents opportunities

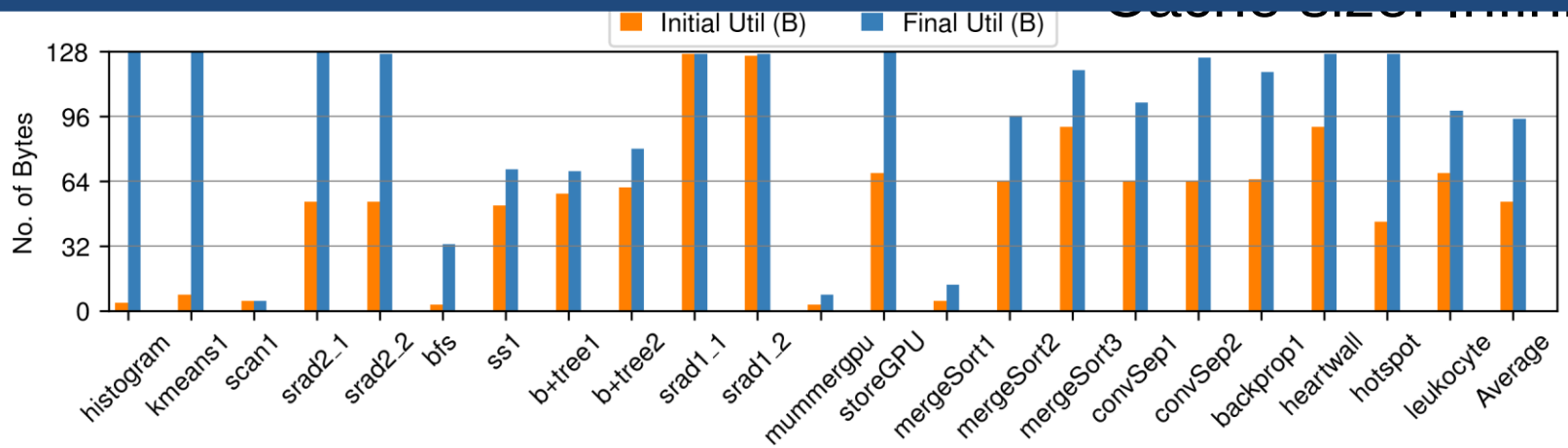


L2 Cache Spatial Utilization

Cache size: 128 KB

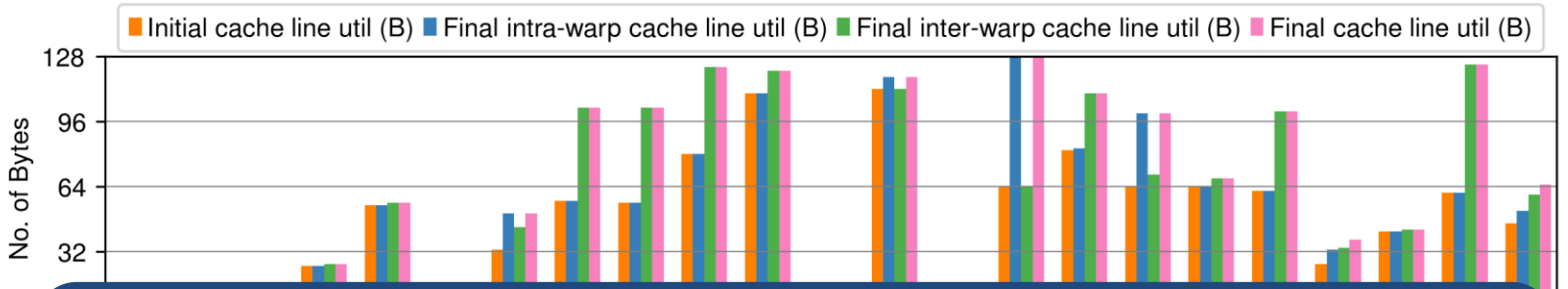


- We know the spatial utilization of cache lines and their limit
- But how much spatial locality is exploited at warp and CTA levels?



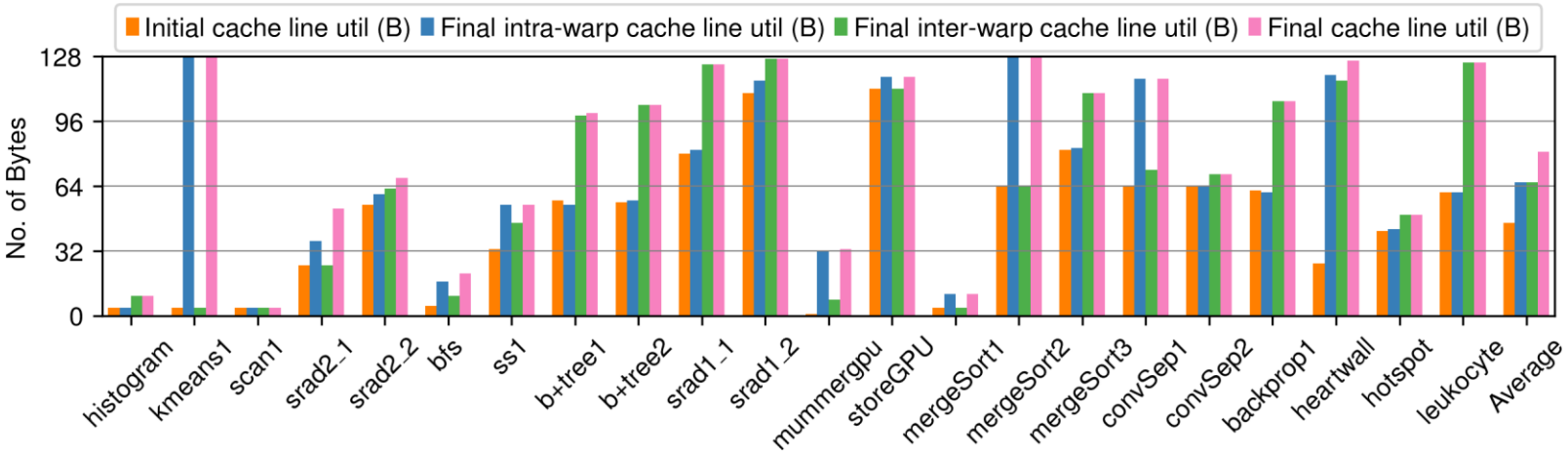
L1 Spatial Utilization at Warp and CTA Levels

Cache size: 16 KB



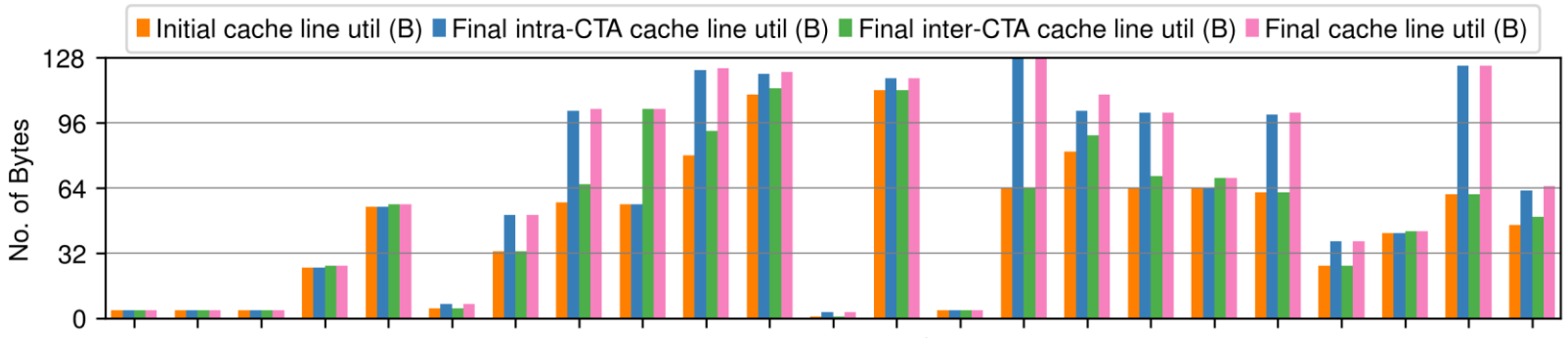
- Observation: Intra-warp and inter-warp accesses contribute equally to spatial locality
- Do these warps belong to the same or different CTAs?

Cache size: Infinite



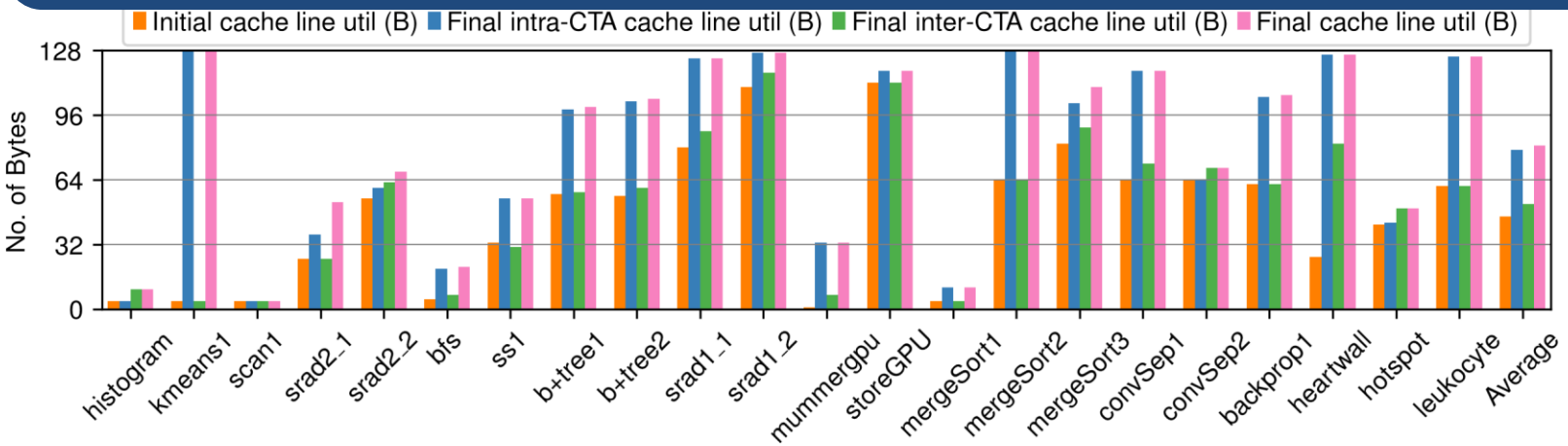
L1 Spatial Utilization at CTA Level

Cache size: 16 KB



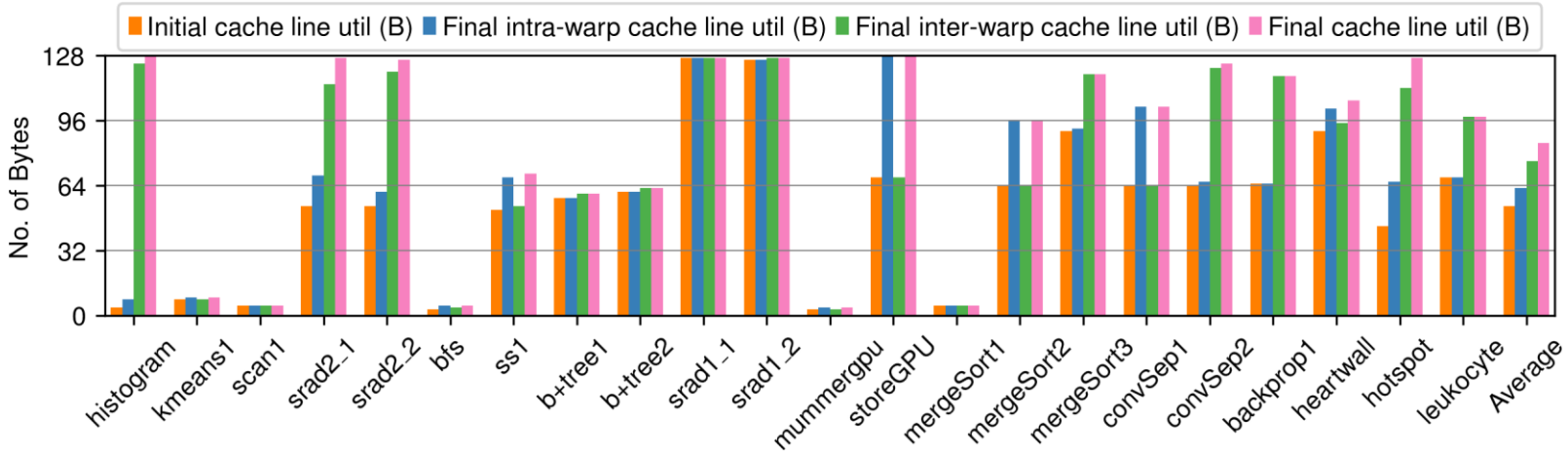
➢Observation: Most of the inter-warp locality belongs to the warps from intra-CTAs rather than inter-CTAs

Cache size: Infinite

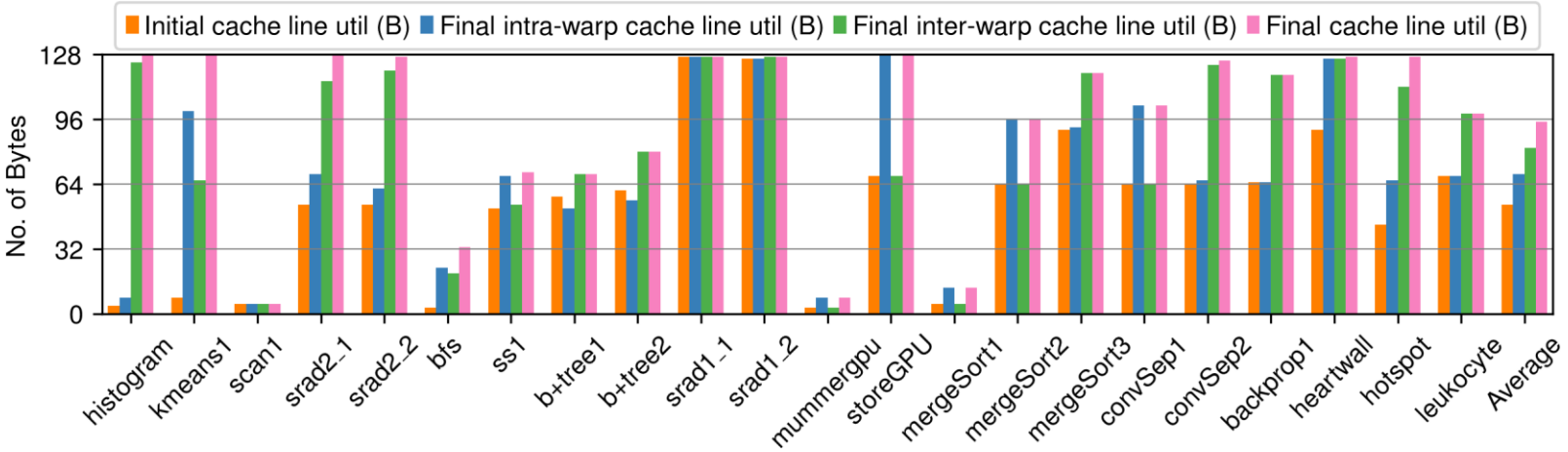


L2 Spatial Utilization at Warp Level

Cache size: 128 KB

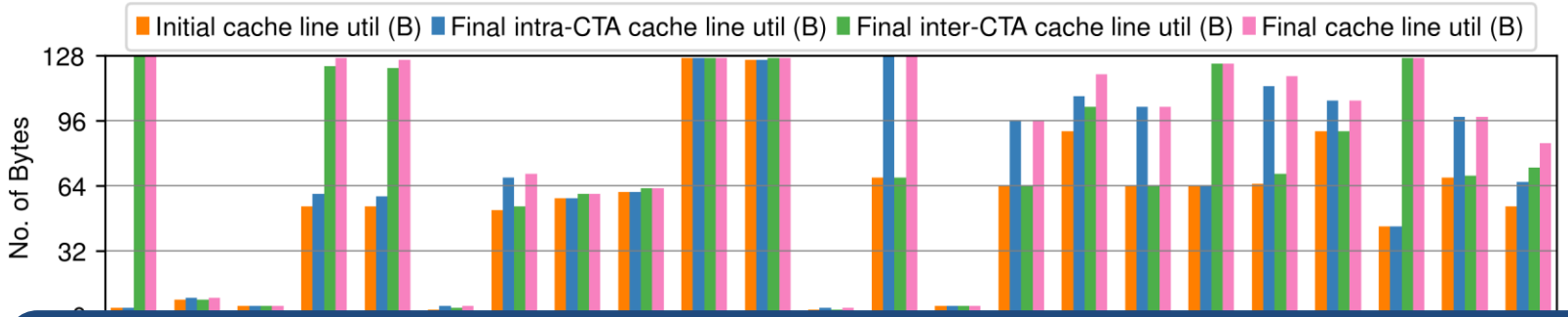


Cache size: Infinite



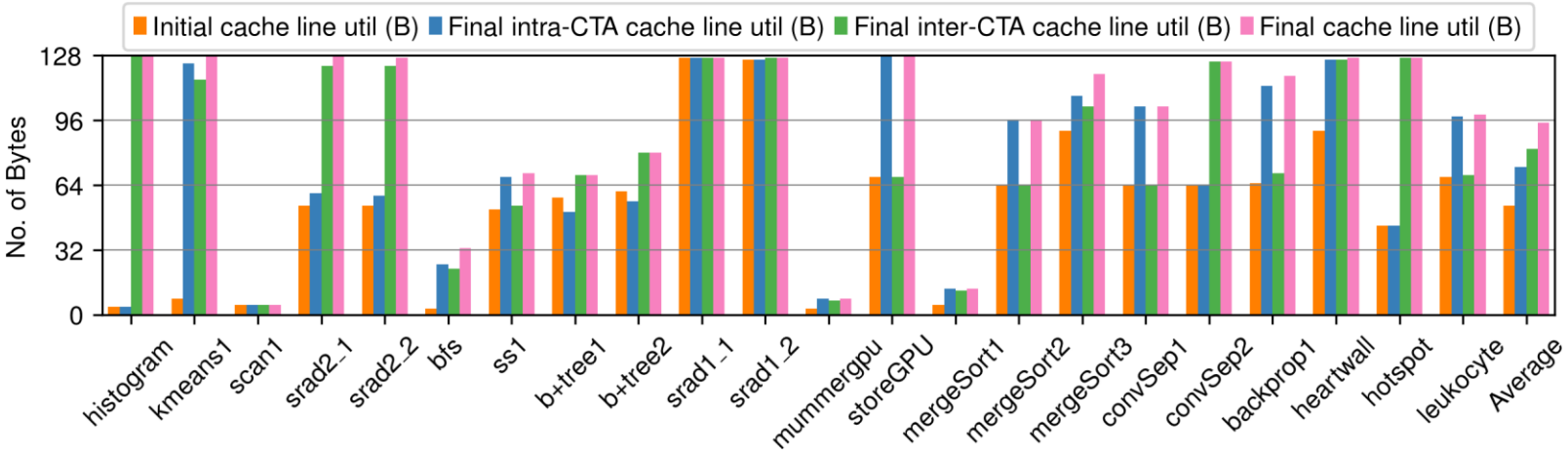
L2 Spatial Utilization at CTA Level

Cache size: 128 KB



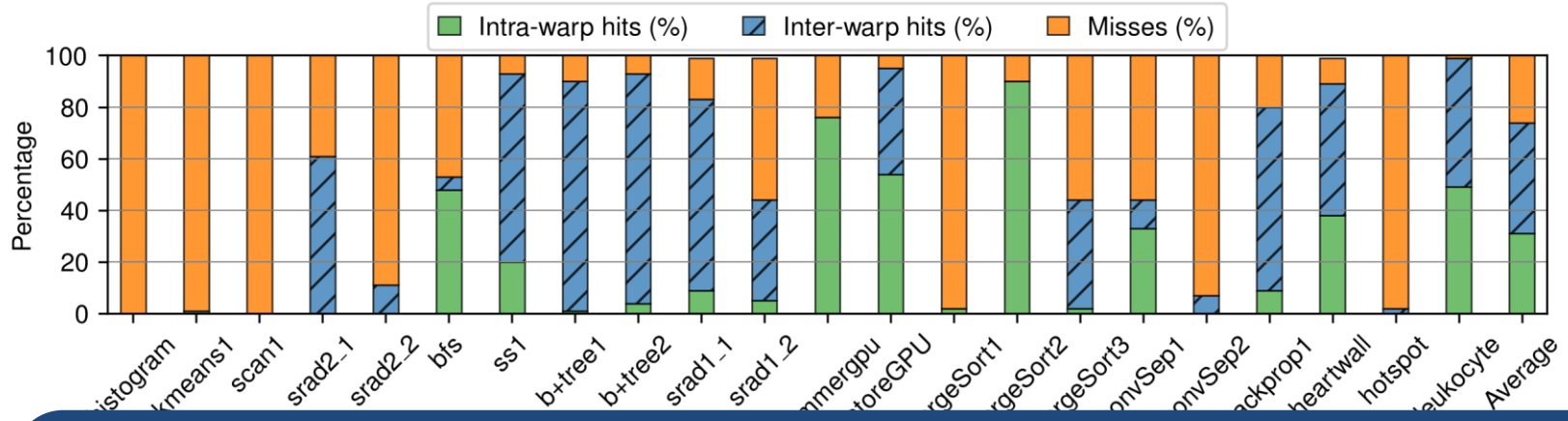
➢Observation: Most of the inter-warp locality belongs to the warps from inter-CTAs rather than intra-CTAs (82 B vs. 73 B)

Cache size: Infinite



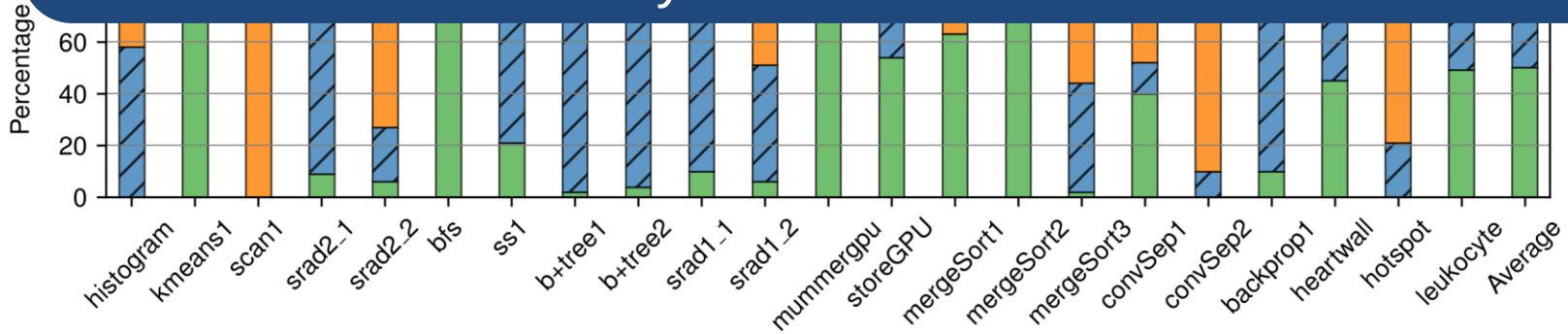
Warp-Level Hit and Miss Rate: L1

Cache size: 16 KB



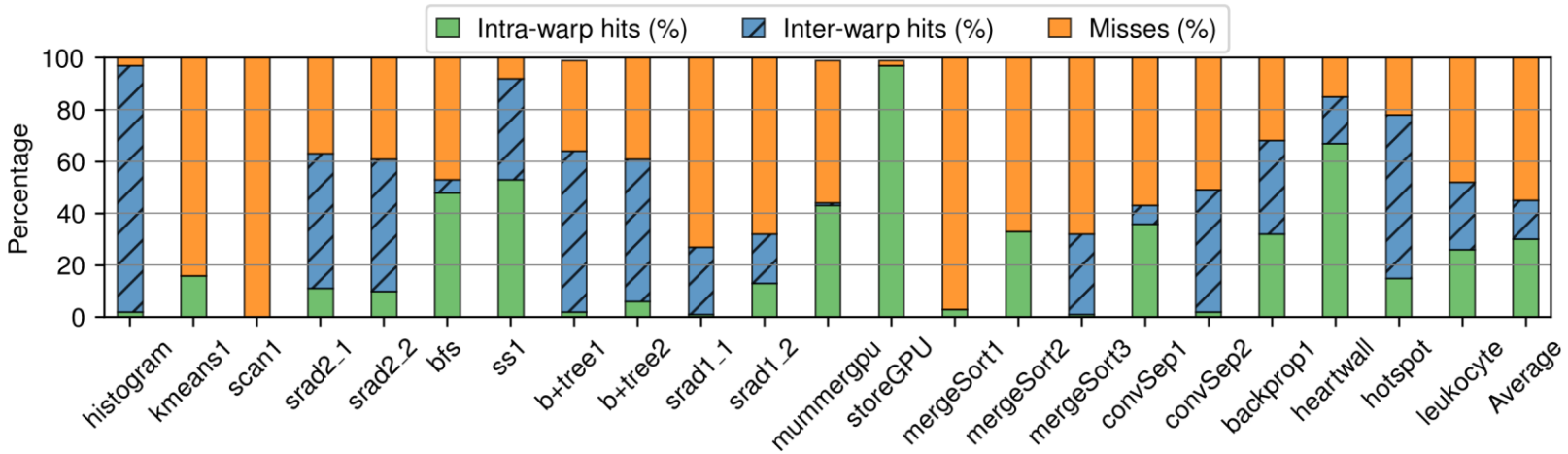
- Observation: significantly higher inter-warp hits (46%) at the L1 cache in contrast to the previous study by
- This is possible for memory divergent workloads as they have scattered memory accesses

Cache size: Infinite

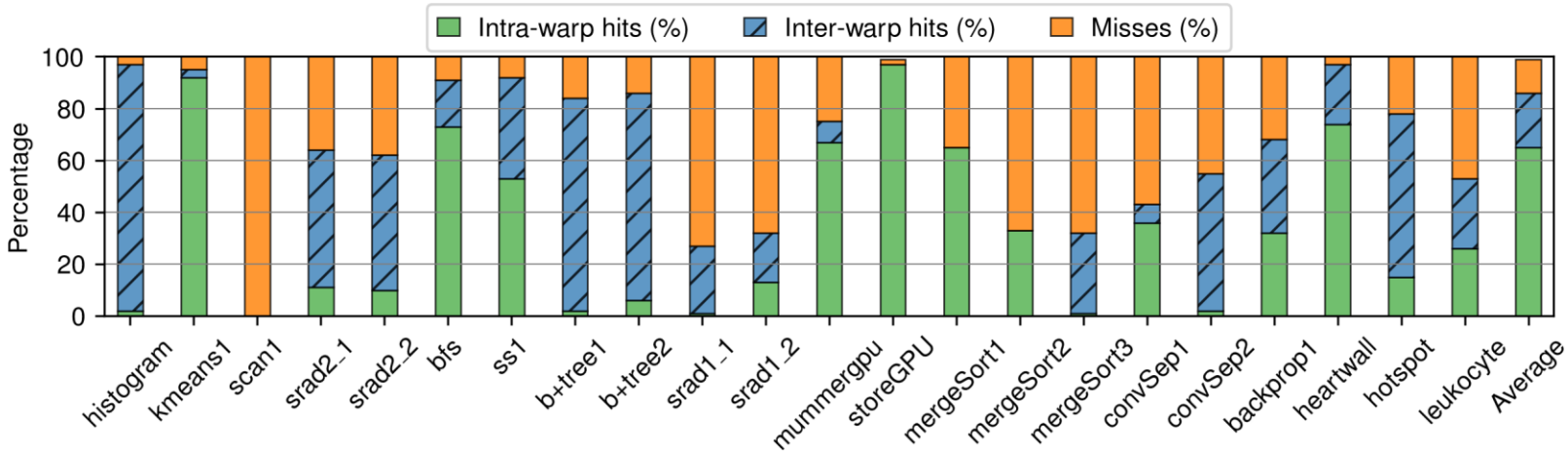


Warp-Level Hit and Miss Rate: L2

Cache size: 128 KB

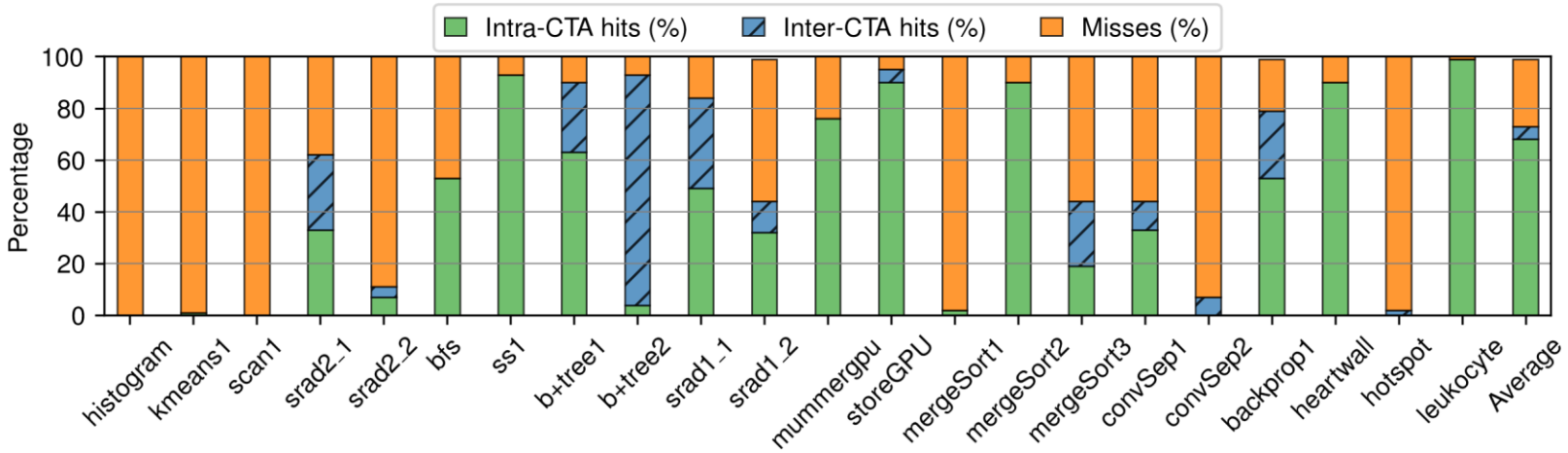


Cache size: Infinite

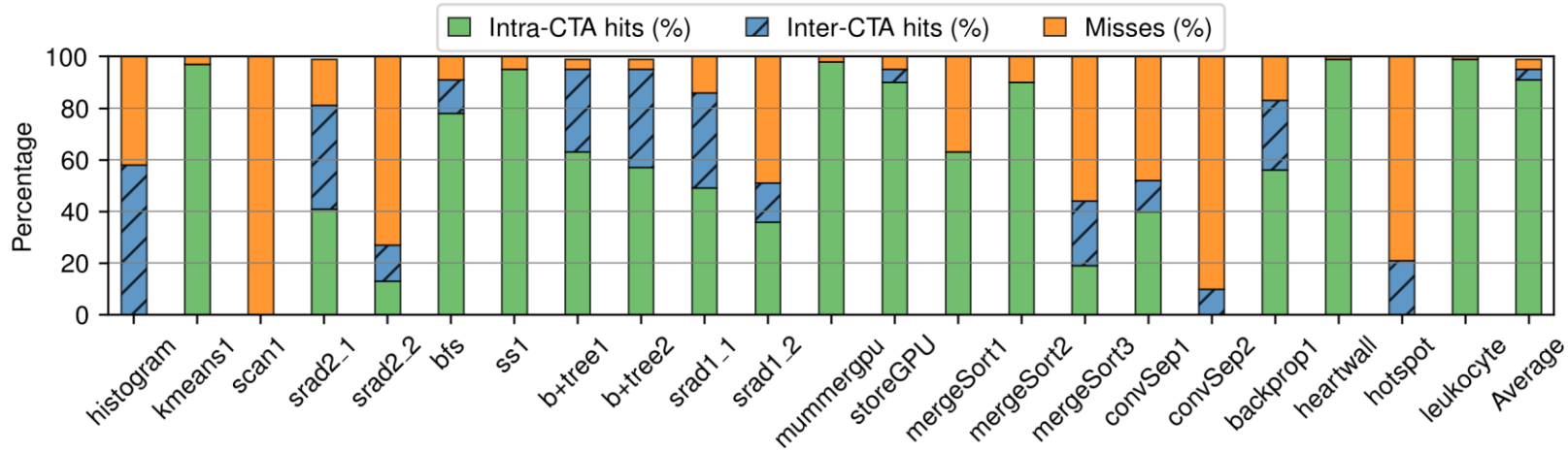


CTA-Level Hit and Miss Rate: L1

Cache size: 16 KB

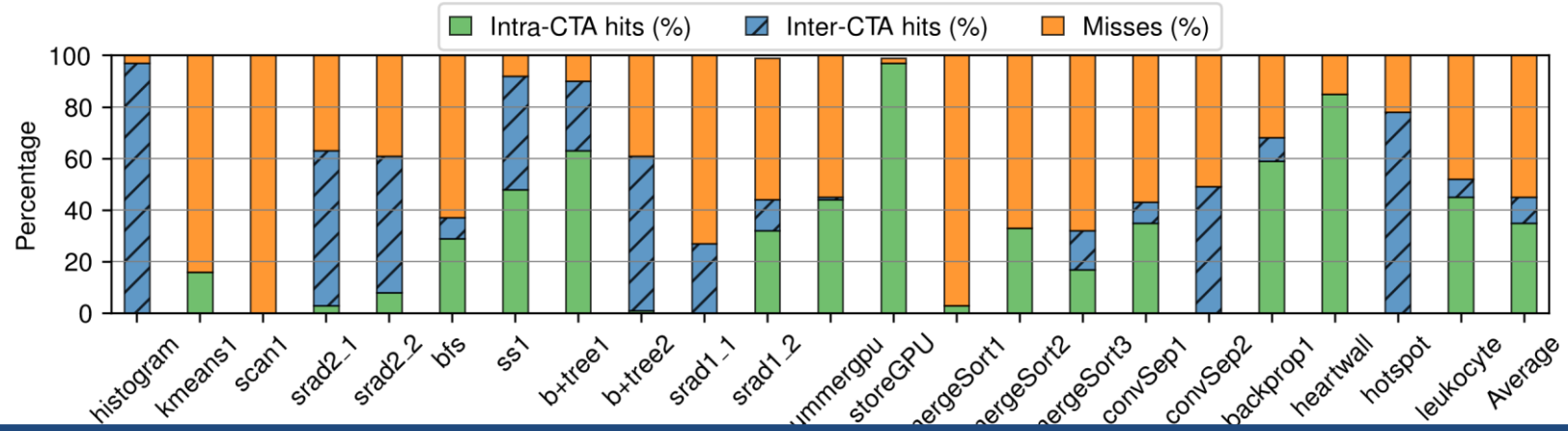


Cache size: Infinite



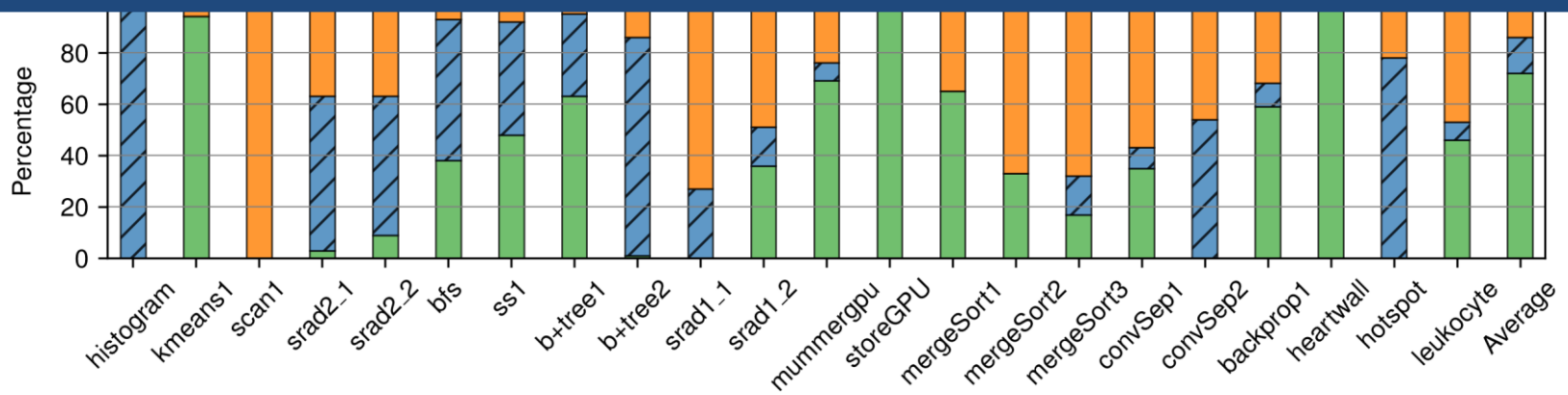
CTA-Level Hit and Miss Rate: L2

Cache size: 128 KB

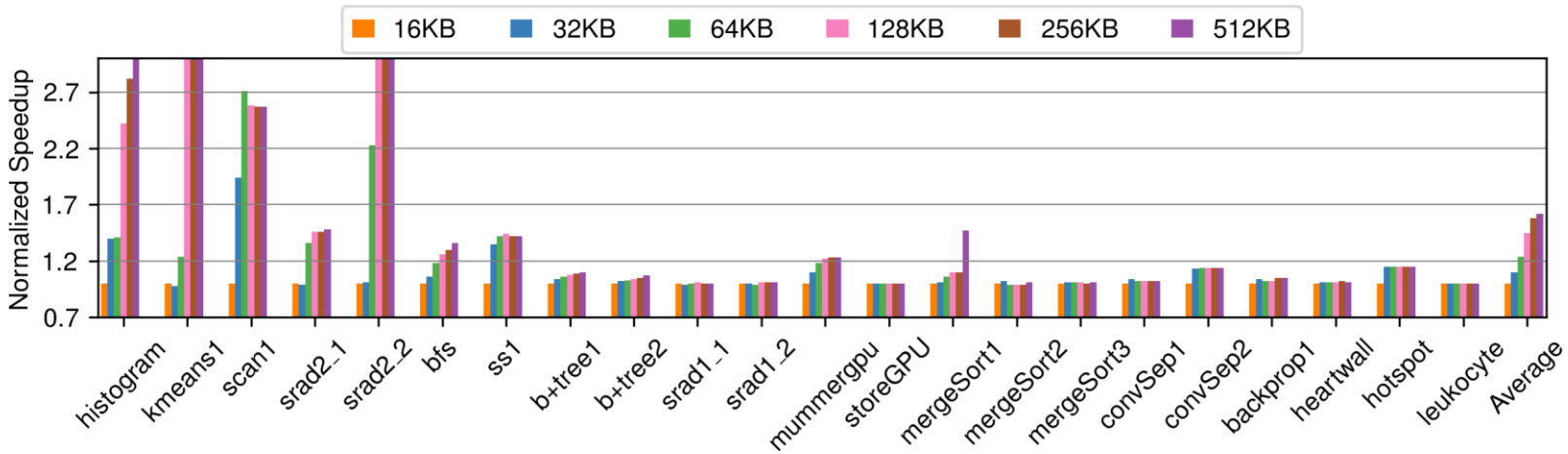


➢Observation: The CTA level hit rate confirms a higher intra-CTA locality at the L1 cache and inter-CTA locality at the L2 cache.

Cache size: Infinite



Cache Sensitivity



- Observations:
  - Performance increases up to 6.3x and on average 1.62x
  - Performance does not decrease
  - Also useful to save power consumption

Potential Use Cases

- Augment Sector Cache with spatial locality predictor
- Warp scheduler improvements

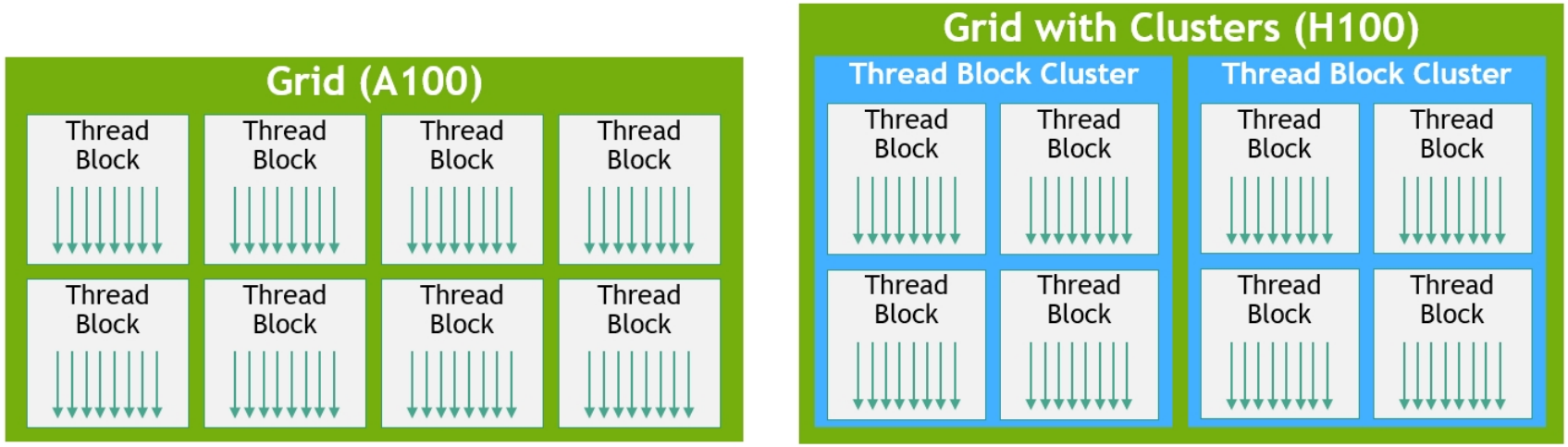
Quantitative Analysis Summary

TUHH
Technische Universität Hamburg

- Caches play an important role to bride the processor and memory gap
- Caches are direct mapped, set associative, or fully-associative
- GPU caches play an important role in higher performance
- Per-thread share of GPU caches is much smaller than CPU
- Caches suffer from thrashing, in particular, for **memory-divergent apps**
- Conducted **quantitative study** of locality and its limit in GPU caches
  - Significant higher locality than exploited by GPUs
  - **57%** cache lines **never re-referenced**, but the limit study shows only 30% never re-referenced
  - **50% L1 cache capacity wasted** due to data over-fetch
  - Initial low spatial utilization justifies use of **demand-fetched caches**
  - Much higher **actual spatial util**. presents **opportunities for optimizations**

Distributed Shared Memory in Hopper Architecture

H100 GPU Hierarchy

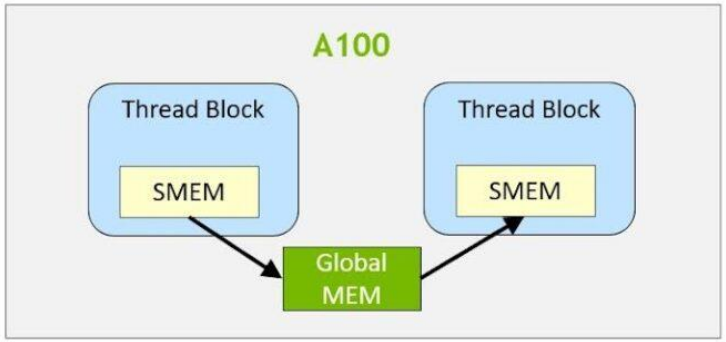


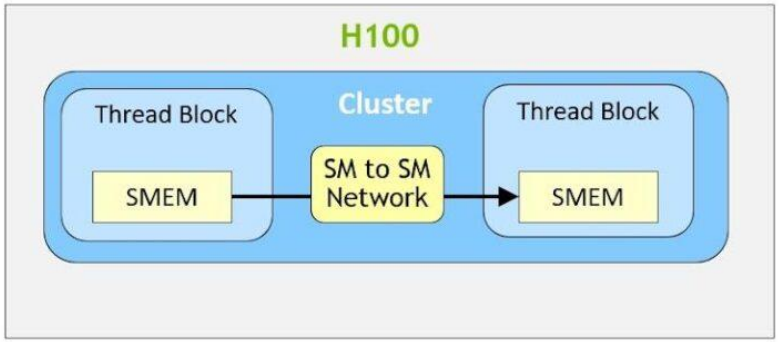
Grid vs. Grid with Clusters (A100 vs. H100 with clusters)

Src:Nvidia

- Thread blocks in a grid can **optionally** be grouped at kernel launch into **thread block clusters**
- **Cooperative_groups** API in CUDA to leverage cluster capabilities

Distributed Shared Memory

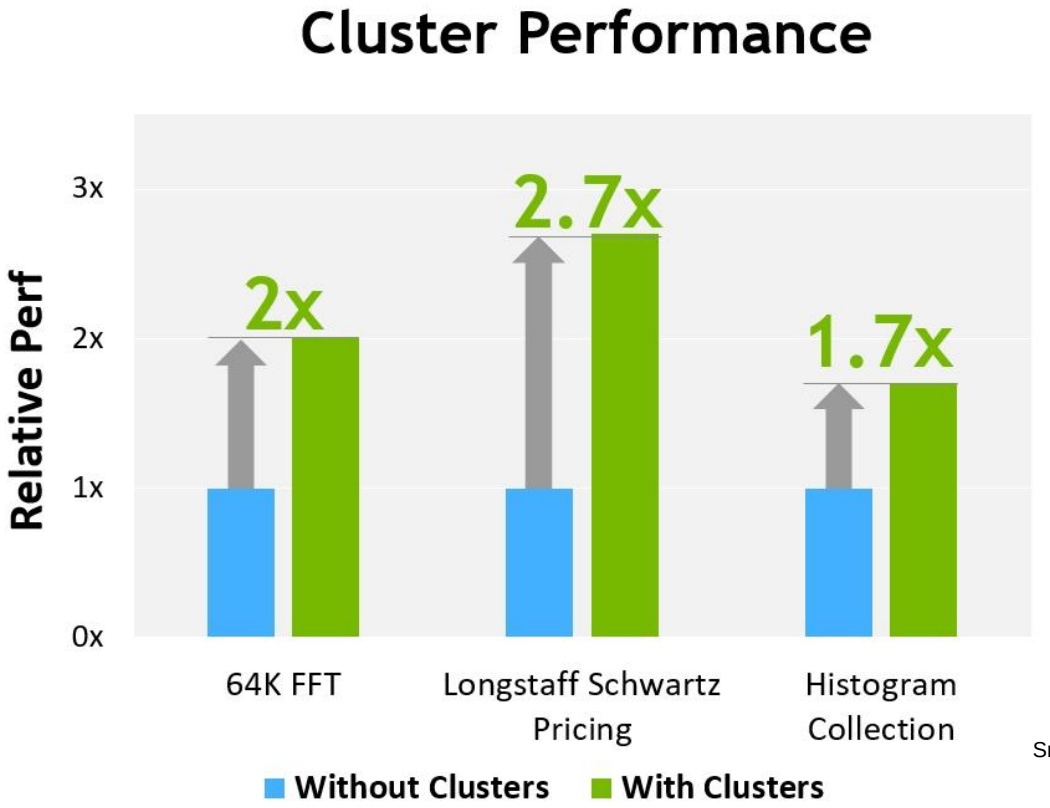




Thread-block-to-thread-block data exchange (A100 vs. H100 with clusters)

Src:Nvidia

Cluster Performance



Src:Nvidia

Cluster vs. non-cluster performance comparisons