MT4G

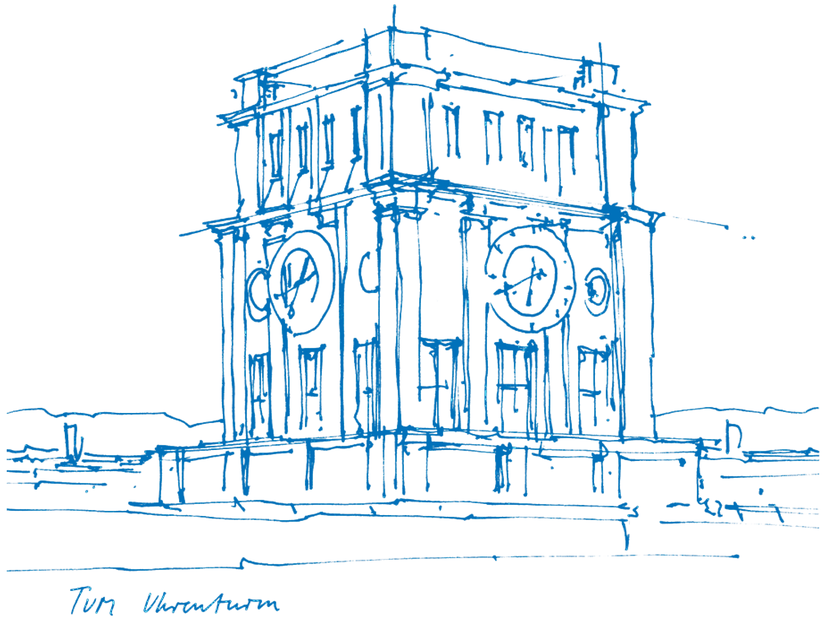
A Tool for Reliable Auto-Discovery of NVIDIA and AMD GPU Compute and Memory Topologies

Stepan Vanecek
stepan.vanecek@tum.de

Manuel Walter Mußbacher
Dominik Größler
Urvij Saroliya
Martin Schulz

Chair of Computer Architecture and Parallel Systems
Technical University of Munich

ProTools'25, 17th November 2025



Why MT4G? GPU Topologies are complex yet important

Why MT4G? GPU Topologies are complex yet important

1 GPU topologies are very complex

- Compute — massive parallelism and multiple compute blocks
- Memory — many different caches, specific for each vendor/microarchitecture

Why MT4G? GPU Topologies are complex yet important

1 GPU topologies are very complex

- Compute — massive parallelism and multiple compute blocks
- Memory — many different caches, specific for each vendor/microarchitecture

2 Massive throughput required

- Cache subsystem plays a crucial role
- Stalls potentially affect thousands of compute cores on a single GPU

Why MT4G? GPU Topologies are complex yet important

1 GPU topologies are very complex

- Compute — massive parallelism and multiple compute blocks
- Memory — many different caches, specific for each vendor/microarchitecture

2 Massive throughput required

- Cache subsystem plays a crucial role
- Stalls potentially affect thousands of compute cores on a single GPU

3 Limited control over scheduling

Why MT4G? GPU Topologies are complex yet important

Why MT4G? GPU Topologies are complex yet important

Trivial use-case

Repeat accesses to one array

? Array size

? Performance

? Performance Portability

Why MT4G? GPU Topologies are complex yet important

Trivial use-case

Repeat accesses to one array

- ? Array size
- ? Performance
- ? Performance Portability

Performance results

1. Different Array Sizes
2. Different GPUs

Why MT4G? GPU Topologies are complex yet important

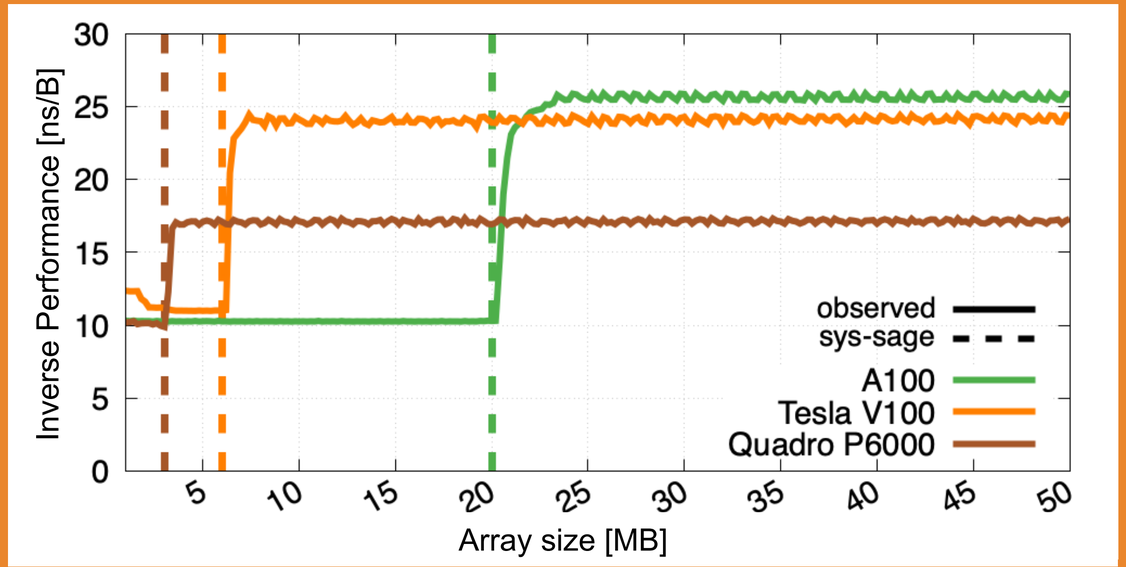
Trivial use-case

Repeat accesses to one array

? Array size
? Performance
? Performance Portability

Performance results

1. Different Array Sizes
2. Different GPUs



source: Vanecek et al., sys-sage: A Unified Representation of Dynamic Topologies & Attributes on HPC Systems

Why MT4G? GPU Topologies are complex yet important

Trivial use-case

Repeat accesses to one array

? Array size
? Performance
? Performance Portability

Observations

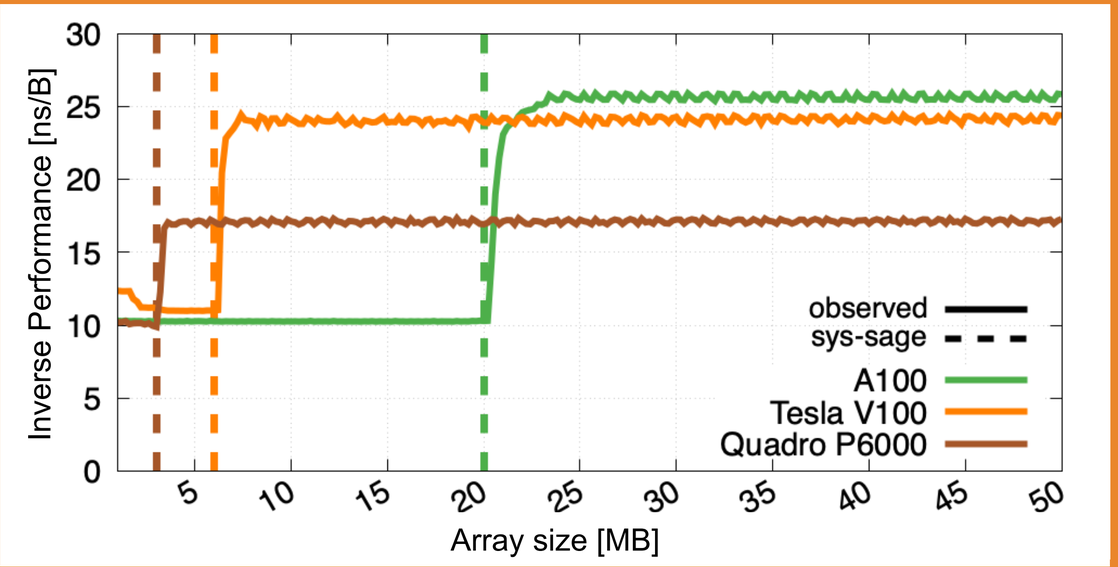
- Array size is performance-critical (up to 2.5x)
- Sharp changes, different for each GPU
- Predictable behaviour

Which array size to chose?

➡️ **Need for topological information!**

Performance results

1. Different Array Sizes
2. Different GPUs



source: Vanecek et al., sys-sage: A Unified Representation of Dynamic Topologies & Attributes on HPC Systems

Current Landscape

Current Landscape

CPUs

Tools & Interfaces for discovery:

- HWloc, LIKWID, lscpu, …

Providing valuable Information:

- Number of Cores, hyperthreading
- NUMAs, sockets, ..
- Cache hierarchy & properties (size, $lines, ..)
- Main memory (incl. sizes)

Current Landscape

CPUs

Tools & Interfaces for discovery:

- HWloc, LIKWID, lscpu, …

Providing valuable Information:

- Number of Cores, hyperthreading
- NUMAs, sockets, ..
- Cache hierarchy & properties (size, $lines, ..)
- Main memory (incl. sizes)

GPUs

Tools & Interfaces for discovery:

- —lack ot standardized ones—
- nvidia-smi, rocm-smi, hipdeviceproperties, ..

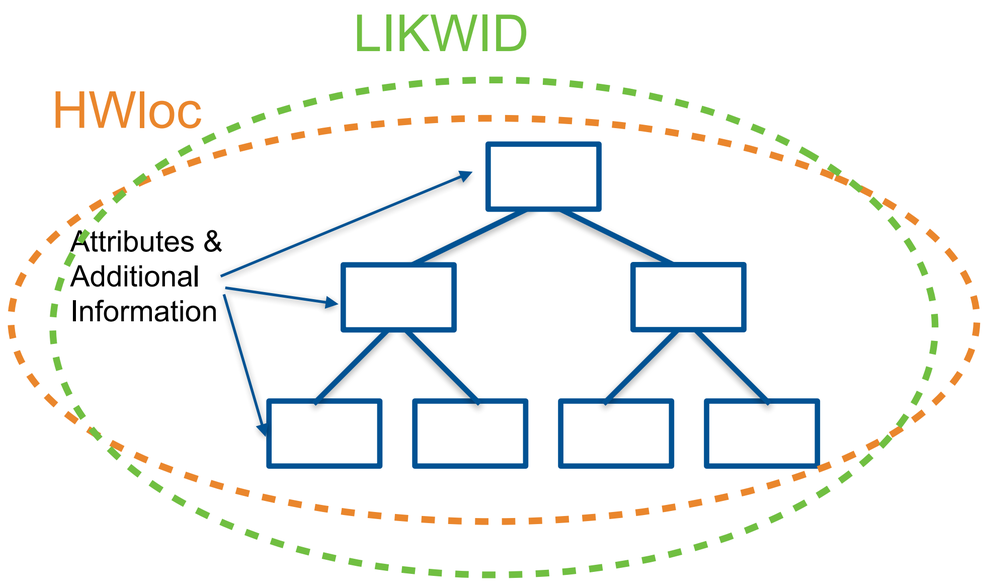
Scattered information:

- nvidia-smi: GPU name, memory size
- cuda/hipdeviceprop: L2 cache size
- nvml: GPU configuration

Current Landscape

CPUs

Complete, cross-vendor



GPUs

Tools & Interfaces for discovery:

- —lack ot standardized ones—
- nvidia-smi, rocm-smi, hipdeviceproperties, ..

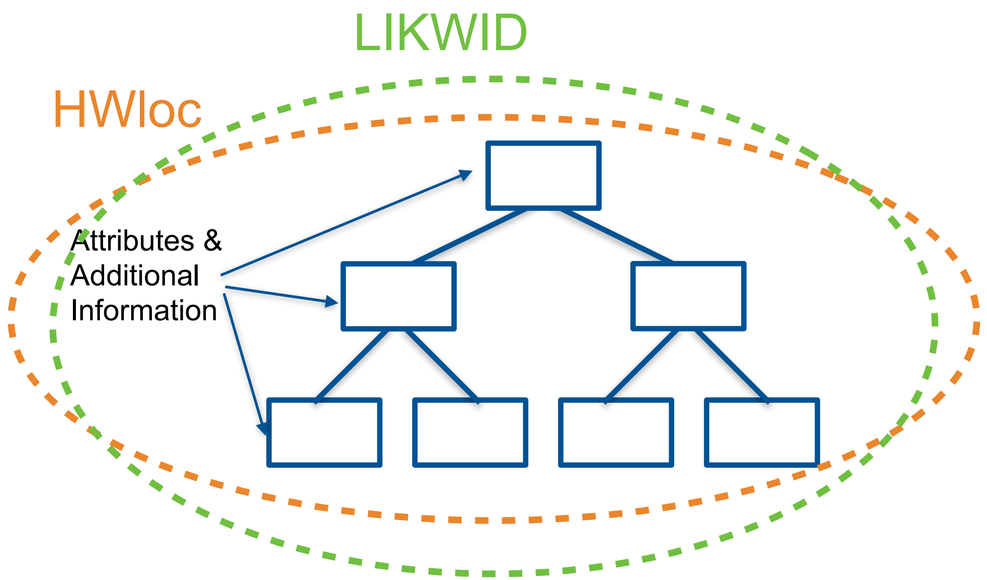
Scattered information:

- nvidia-smi: GPU name, memory size
- cuda/hipdeviceprop: L2 cache size
- nvml: GPU configuration

Current Landscape

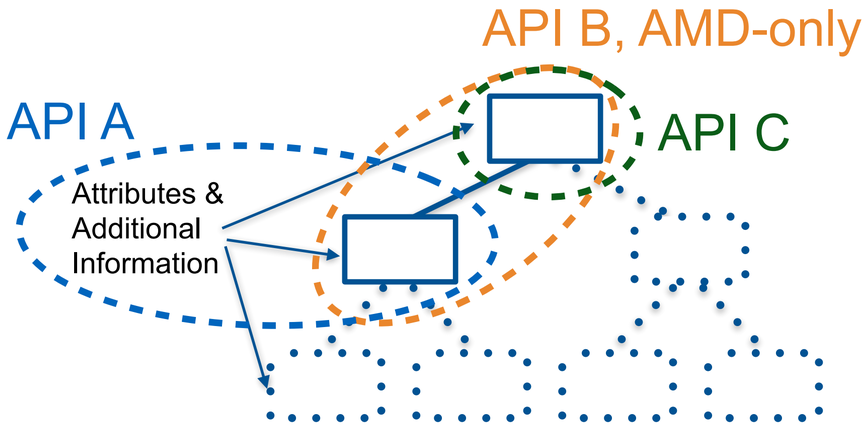
CPUs

Complete, cross-vendor



GPUs

Incomplete, scattered, vendor-specific



What MT4G Provides

What MT4G Provides

Report on GPU building blocks and their attributes and capabilities

- NVIDIA (Pascal and newer)
- AMD (CDNA)

What MT4G Provides

Report on GPU building blocks and their attributes and capabilities

- NVIDIA (Pascal and newer)
- AMD (CDNA)

Target Memory Elements

NVIDIA

- L1 cache
- L2 cache
- Texture & Readonly cache
- Constant L1, L1.5 cache
- Shared memory
- Device memory

AMD

- L1 cache
- L1s cache
- L2 cache
- L3 cache (partially)
- Local Data Share
- Device memory

What MT4G Provides

Report on GPU building blocks and their attributes and capabilities

- NVIDIA (Pascal and newer)
- AMD (CDNA)

Target Memory Elements

NVIDIA

- L1 cache
- L2 cache
- Texture & Readonly cache
- Constant L1, L1.5 cache
- Shared memory
- Device memory

AMD

- L1 cache
- L1s cache
- L2 cache
- L3 cache (partially)
- Local Data Share
- Device memory

Queried properties

- Size
- Cache Line Size ($-only)
- Fetch Granularity ($-only)
- Physical Layout
- Load Latency
- Read & Write Bandwidth (L2 cache and above)

\+ Compute Resource Topological Information

How MT4G Collects Data

How MT4G Collects Data

1. Integrates existing interfaces, where available
   - HIP runtime (hipDeviceProp_t), AMD KFD Driver files

How MT4G Collects Data

1. Integrates existing interfaces, where available
   - HIP runtime (hipDeviceProp_t), AMD KFD Driver files
2. Introduces a suite of (>50) microbenchmarks

How MT4G Collects Data

1. Integrates existing interfaces, where available
   - HIP runtime (hipDeviceProp_t), AMD KFD Driver files
2. Introduces a suite of (>50) microbenchmarks

   Automatic evaluation of the results
   - Kolmogorov-Smirnov test
   - Base statistical information
   - Custom heuristic

How MT4G Collects Data

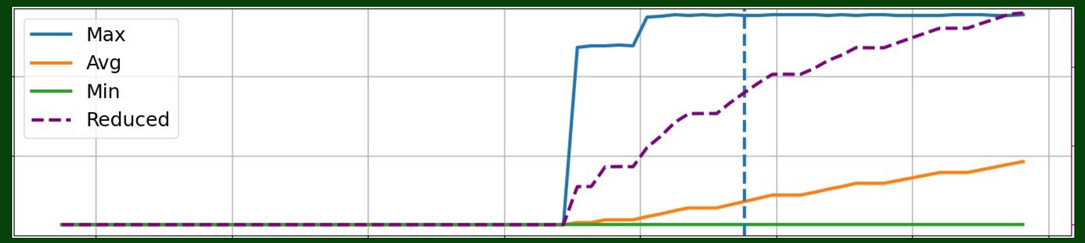
1 Integrates existing interfaces, where available

- HIP runtime (hipDeviceProp_t), AMD KFD Driver files

2 Introduces a suite of (>50) microbenchmarks

Automatic evaluation of the results

- Kolmogorov-Smirnov test
- Base statistical information
- Custom heuristic




```
"texture": {
    "size": {
        "confidence": 0.9704014799260037,
        "method": "p-chase",
        "randomized": false,
        "size": 241920,
        "unit": "bytes"
    }
}
```

How MT4G Collects Data

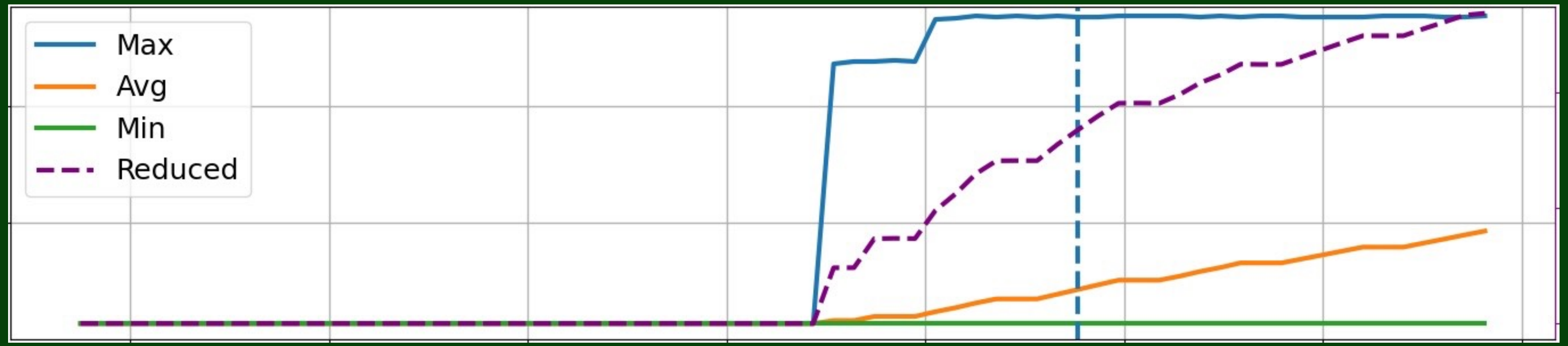
1 Integrates existing interfaces, where available

- HIP runtime (hipDeviceProp_t), AMD KFD Driver files

2 Introduces a suite of (>50) microbenchmarks

44,37,42,37,42,37,42,37,42,37,42,37,42,37,42,37,42,37,42,37,4
2,37,42,37,42,37,42,37,42,37,42,37,42,8,44,37,42,37,42,37,42,37,42,
37,42,37,42,37,42,37,42,37,42,37,42,37,42,37,...

esults



- Custom heuristic


```
"texture": {
    "latency": {
        "mean": 38.77647058823529,
        "measurements": 255,
        "method": "p-chase",
        "p50": 42.0,
        "p95": 42.0,
        "sampleSize": 256,
        "stdev": 5.776090693811043,
        "unit": "cycles"
    },
}
```

```
"texture": {
    "size": {
        "confidence": 0.9704014799260037,
        "method": "p-chase",
        "randomized": false,
        "size": 241920,
        "unit": "bytes"
    }
}
```

What MT4G Provides for **NVIDIA** GPUs

Memory Element	Size	LD lat	R&W BW	cache line size	fetch granularity	Amount per SM/GPU	Physical hardware sharing
L1 cache	✅	✅	➡ SOON	✅	✅	✅	✅
L2 cache	☑	✅	✅	✅	✅	✅	
Texture cache	✅	✅	➡ SOON	✅	✅	✅	✅
Readonly cache	✅	✅	➡ SOON	✅	✅	✅	✅
Constant L1 cache	✅	✅	➡ SOON	✅	✅	✅	✅
Constant L1.5 cache	✅ (up to 64kB)	✅	➡ SOON	❌	✅	❌	
Shared Memory	☑	✅	➡ SOON				
Device Memory	☑	✅	✅				

✅ = Available via microbenchmark; ☑ = Available via API; ➡ SOON = Bandwidth only for higher-level elements; ❌ = not available

What MT4G Provides for **AMD** GPUs

Memory Element	Size	LD lat	R&W BW	cache line size	fetch granularity	Amount per SM/GPU	Physical hardware sharing
(v)L1 cache	✅	✅	➡ SOON	✅	✅	✅	
sL1 cache	✅	✅	➡ SOON	✅	✅		✅ (CU id)
L2 cache	☑	✅	✅	☑	✅	☑	
L3 cache	☑	❌	✅	☑	❌	☑	
Local Data Store	☑	✅	➡ SOON				
Device Memory	☑	✅	✅				

✅ = Available via microbenchmark; ☑ = Available via API; ➡ SOON = Bandwidth only for higher-level elements; ❌ = not available

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

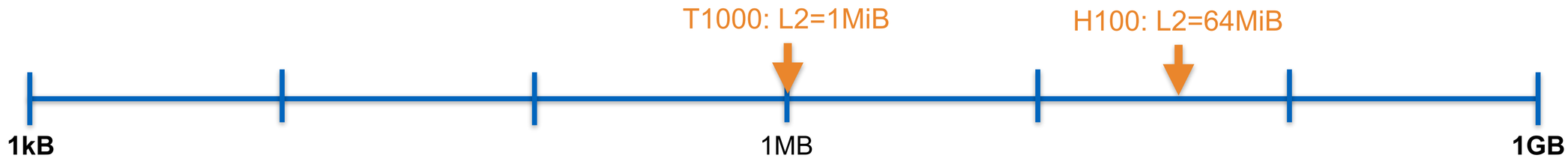
1 Narrow down the search window

2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

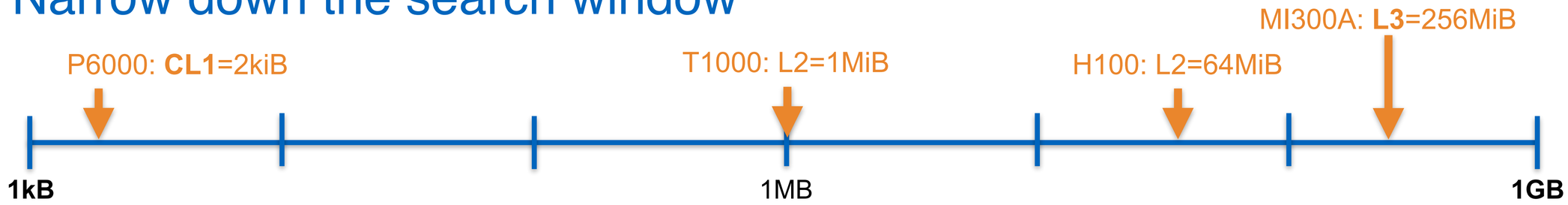


2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

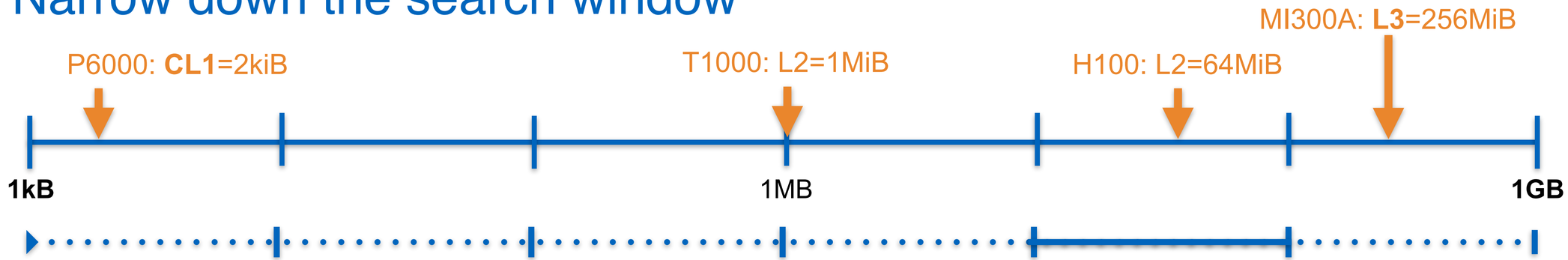


2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window



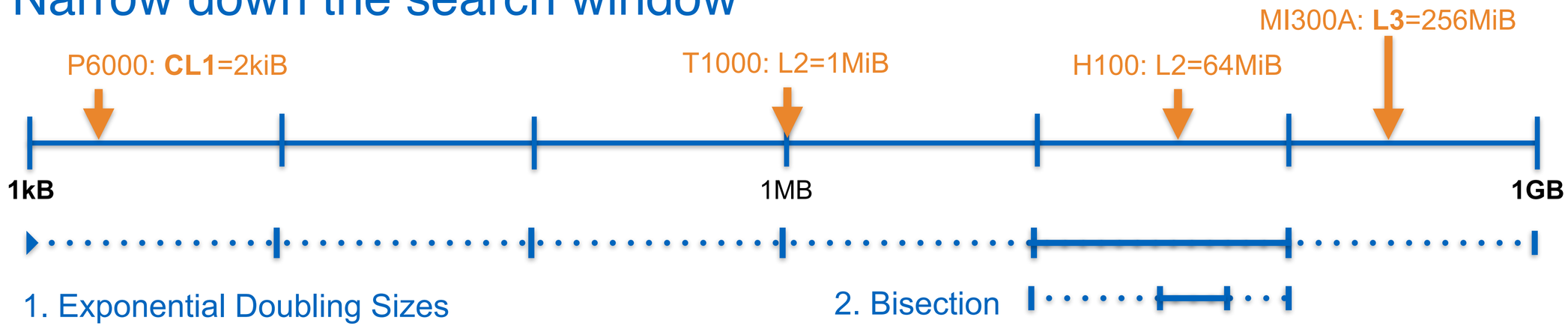
1. Exponential Doubling Sizes

2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

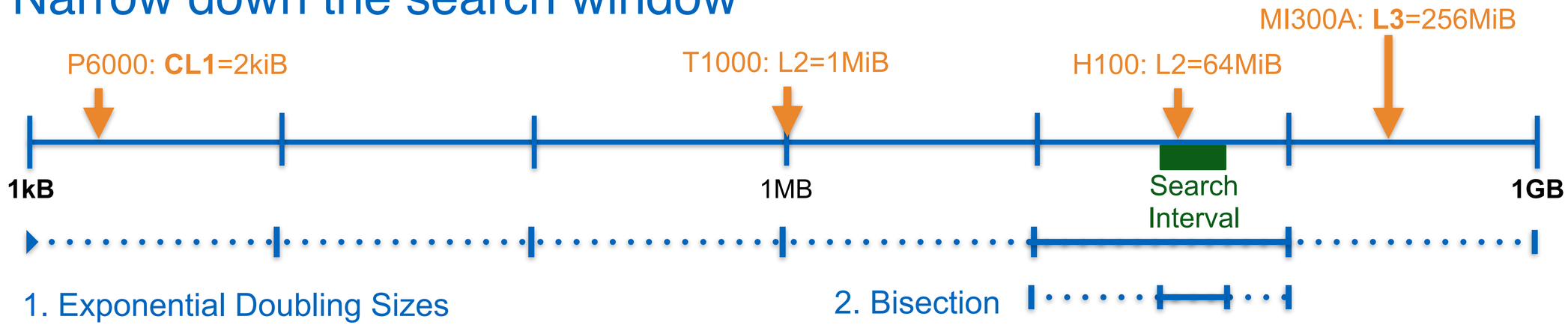


2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window



2 Run p-chase

3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

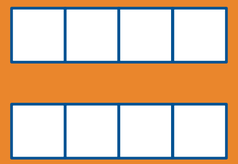
1. Narrow down the search window

2. Run p-chase

   Increase array sizes for p-chase

   array size = 3

   array size = 6



3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window

2. Run p-chase

   Increase array sizes for p-chase

   1. Warm-up (populate the cache)

   array size = 3

   array size = 6

3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window

2. Run p-chase

   Increase array sizes for p-chase

   1. Warm-up (populate the cache)

   array size = 3

   array size = 6

3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window

2. Run p-chase

   Increase array sizes for p-chase

   1. Warm-up (populate the cache)

   array size = 3

   array size = 6

3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

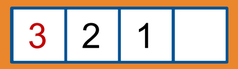
1. Narrow down the search window

2. Run p-chase

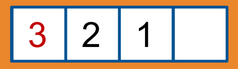
   Increase array sizes for p-chase

   1. Warm-up (populate the cache)

   array size = 3

   array size = 6

3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

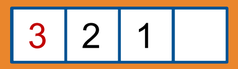
1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase

1. Warm-up (populate the cache)

array size = 3



array size = 6

3 Reduction + K-S test

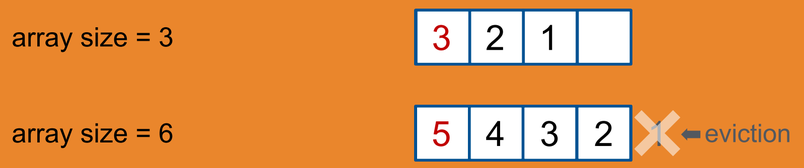
Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase

1. Warm-up (populate the cache)



3 Reduction + K-S test

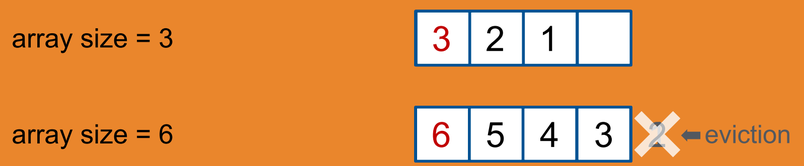
Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase

1. Warm-up (populate the cache)



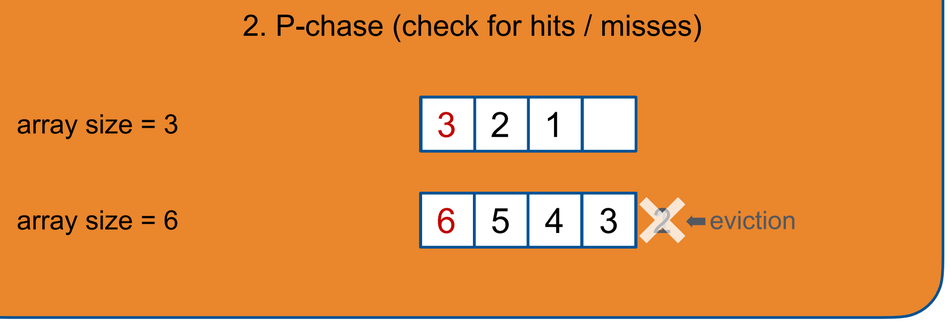
3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase



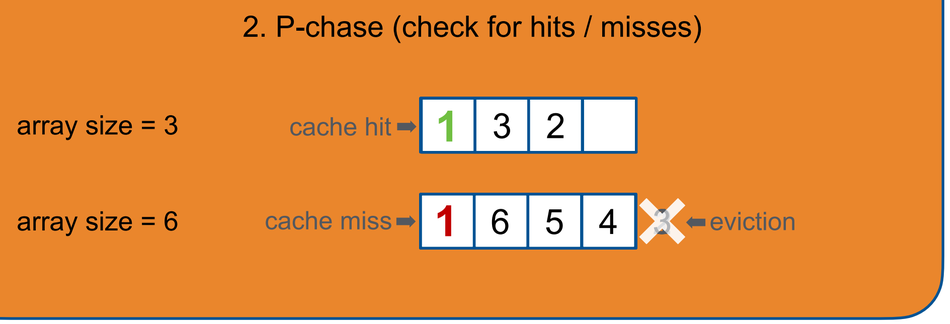
3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase



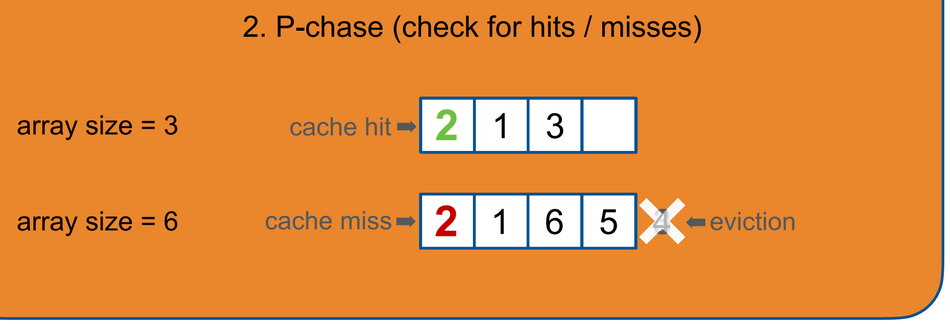
3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase



3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1 Narrow down the search window

2 Run p-chase

Increase array sizes for p-chase

2. P-chase (check for hits / misses)

array size = 3 — cache hit ➡ | 2 | 1 | 3 | |

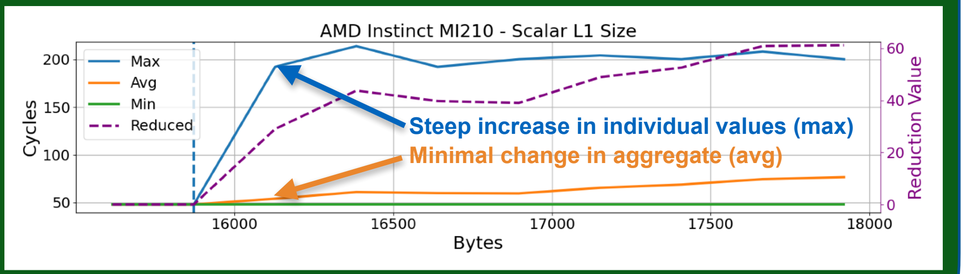
array size = 6 — cache miss ➡ | 2 | 1 | 6 | 5 | ✕ ⬅ eviction

Fine-grained p-chase

```
for(int i=0; i<size; i++)
{
  ts_start[i] = clock();

  index = arr[index];

  ts_end[i] = clock();
}
```



3 Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

Array size: observed latencies

```
254208: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254464: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254720: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254976: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
255232: 38,38,38,38,261,266,263,263,38,38,38,38,38,38,38,38
255488: 38,38,38,38,300,295,304,297,302,261,266,265,38,38,38,38
255744: 304,293,306,298,38,38,38,38,294,305,295,260,267,263,262,261
256000: 292,298,295,304,297,302,305,295,260,264,264,267,260,267,263,262
```

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

$$S_i = \sqrt{\sum_{j=0}^{N} \left(r_{ij} - \min(r) \right)^2}$$

Array size: observed latencies

```
254208: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254464: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254720: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254976: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
255232: 38,38,38,38,261,266,263,263,38,38,38,38,38,38,38,38
255488: 38,38,38,38,300,295,304,297,302,261,266,265,38,38,38,38
255744: 304,293,306,298,38,38,38,38,294,305,295,260,267,263,262,261
256000: 292,298,295,304,297,302,305,295,260,264,264,267,260,267,263,262
```

Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

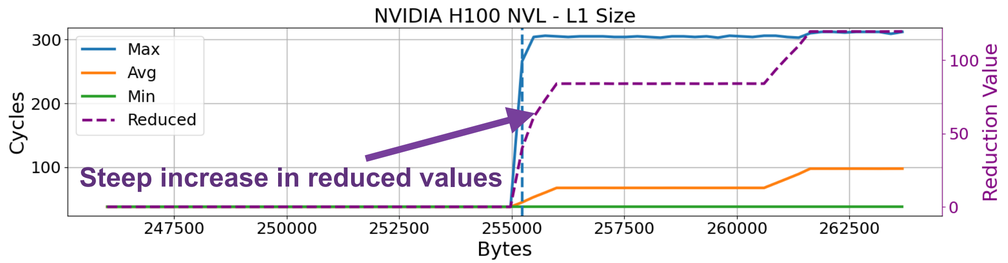
$$S_i = \sqrt{\sum_{j=0}^{N} \left(r_{ij} - \min(r) \right)^2}$$

Array size: observed latencies

```
254208: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254464: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254720: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254976: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
255232: 38,38,38,38,261,266,263,263,38,38,38,38,38,38,38,38
255488: 38,38,38,38,300,295,304,297,302,261,266,265,38,38,38,38
255744: 304,293,306,298,38,38,38,38,294,305,295,260,267,263,262,261
256000: 292,298,295,304,297,302,305,295,260,264,264,267,260,267,263,262
```

2D → 1D

Reduction



Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

$$S_i = \sqrt{\sum_{j=0}^{N} \left(r_{ij} - \min(r) \right)^2}$$

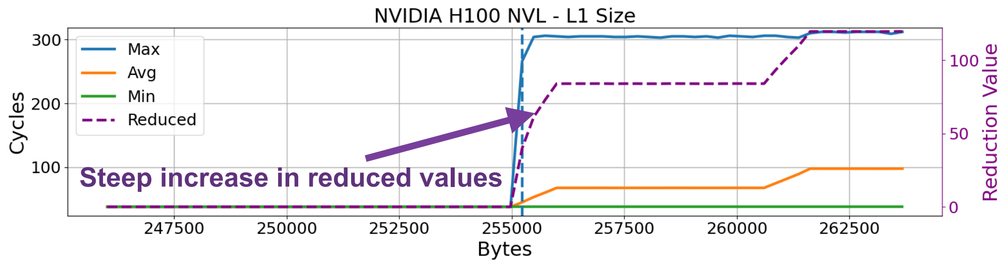
Array size: observed latencies

```
254208: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254464: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254720: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254976: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
255232: 38,38,38,38,261,266,263,263,38,38,38,38,38,38,38,38
255488: 38,38,38,38,300,295,304,297,302,261,266,265,38,38,38,38
255744: 304,293,306,298,38,38,38,38,294,305,295,260,267,263,262,261
256000: 292,298,295,304,297,302,305,295,260,264,264,267,260,267,263,262
```

2D → 1D

Reduction

```
254208: 0
254464: 2
254720: 0
254976: 0
255232: 28
255488: 44
255744: 50
256000: 62
```



Microbenchmark deep-dive: Cache Size

1. Narrow down the search window
2. Run p-chase
3. Reduction + K-S test

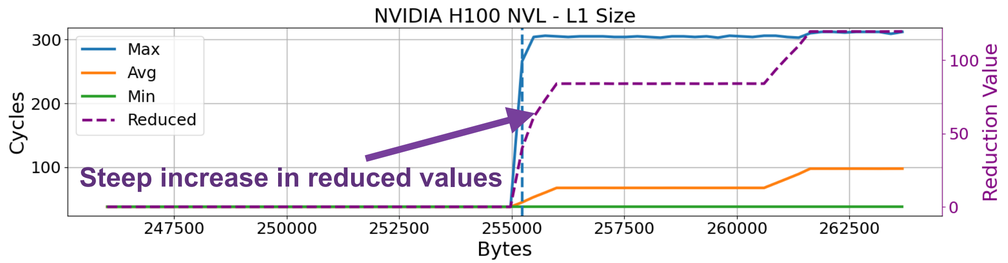
$$S_i = \sqrt{\sum_{j=0}^{N} \left(r_{ij} - \min(r) \right)^2}$$

Array size: observed latencies

```
254208: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254464: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254720: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
254976: 38,38,38,38,38,38,38,38,38,38,38,38,38,38,38,38
255232: 38,38,38,38,261,266,263,263,38,38,38,38,38,38,38,38
255488: 38,38,38,38,300,295,304,297,302,261,266,265,38,38,38,38
255744: 304,293,306,298,38,38,38,38,294,305,295,260,267,263,262,261
256000: 292,298,295,304,297,302,305,295,260,264,264,267,260,267,263,262
```

2D → 1D Reduction

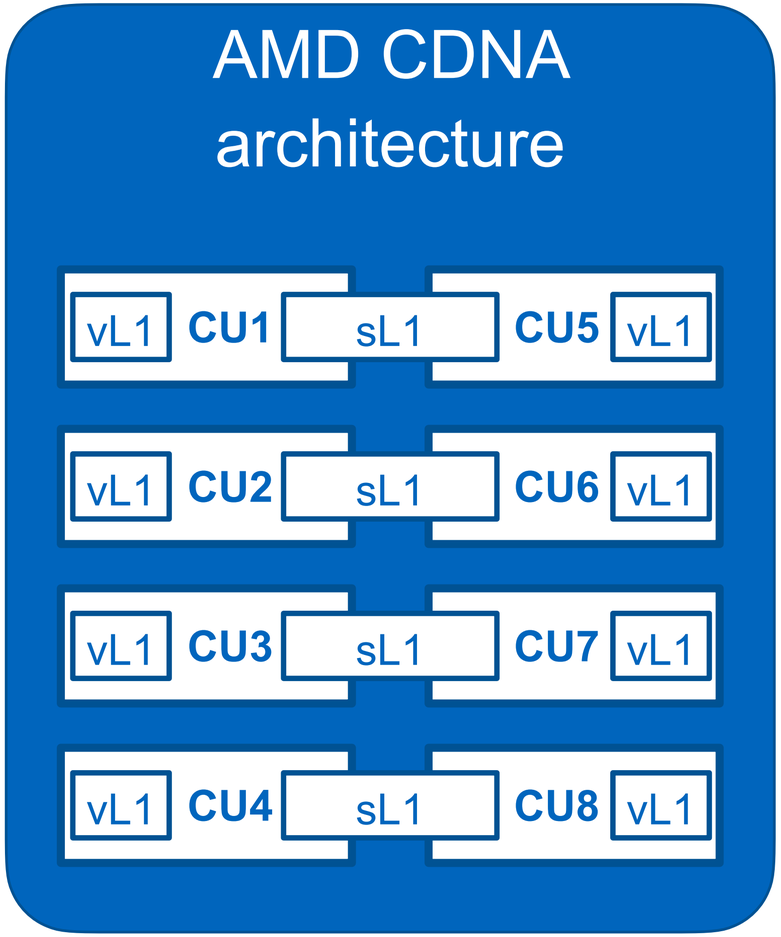
```
254208: 0
254464: 2
254720: 0
254976: 0
255232: 28   ← K-S: CHANGE POINT FOUND
255488: 44
255744: 50
256000: 62
```



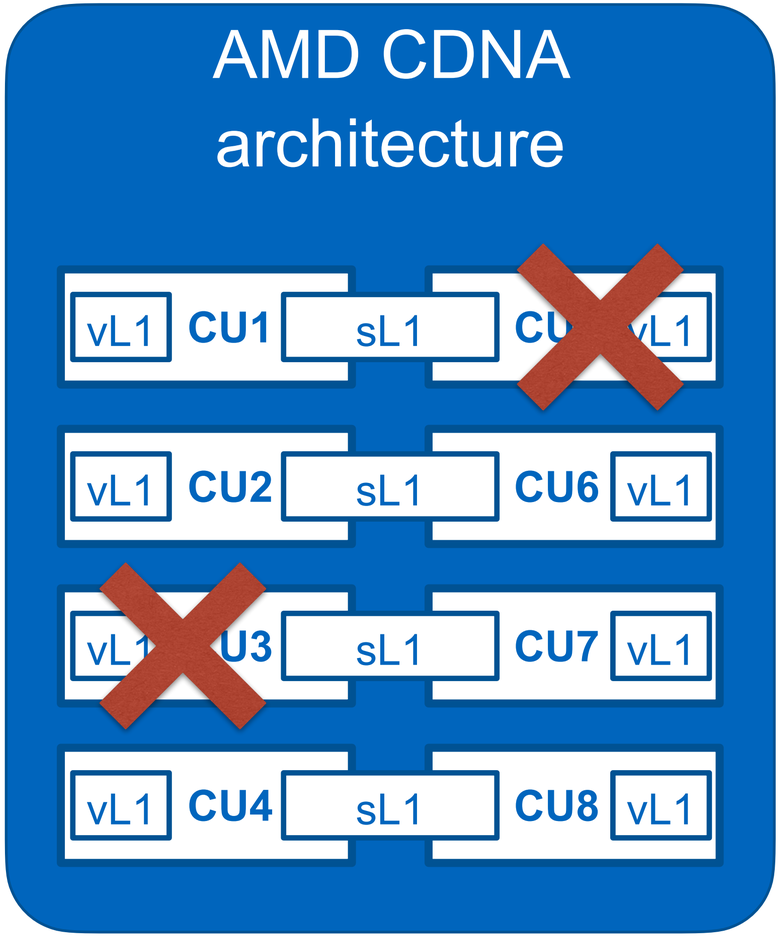
Microbenchmark deep-dive: AMD scalar L1 cache sharing

AMD CDNA architecture

Microbenchmark deep-dive: AMD scalar L1 cache sharing

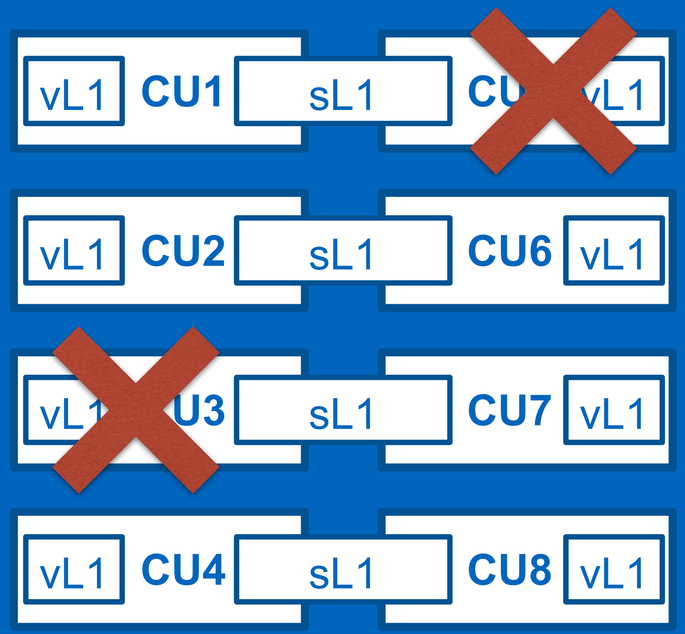


Microbenchmark deep-dive: AMD scalar L1 cache sharing



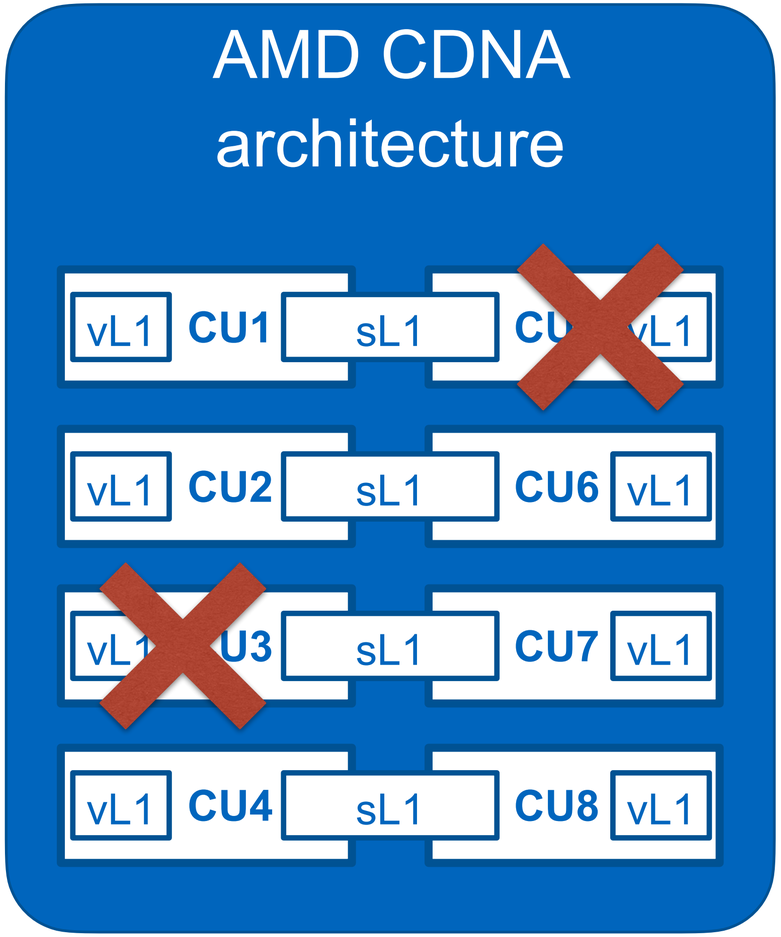
Microbenchmark deep-dive: AMD scalar L1 cache sharing

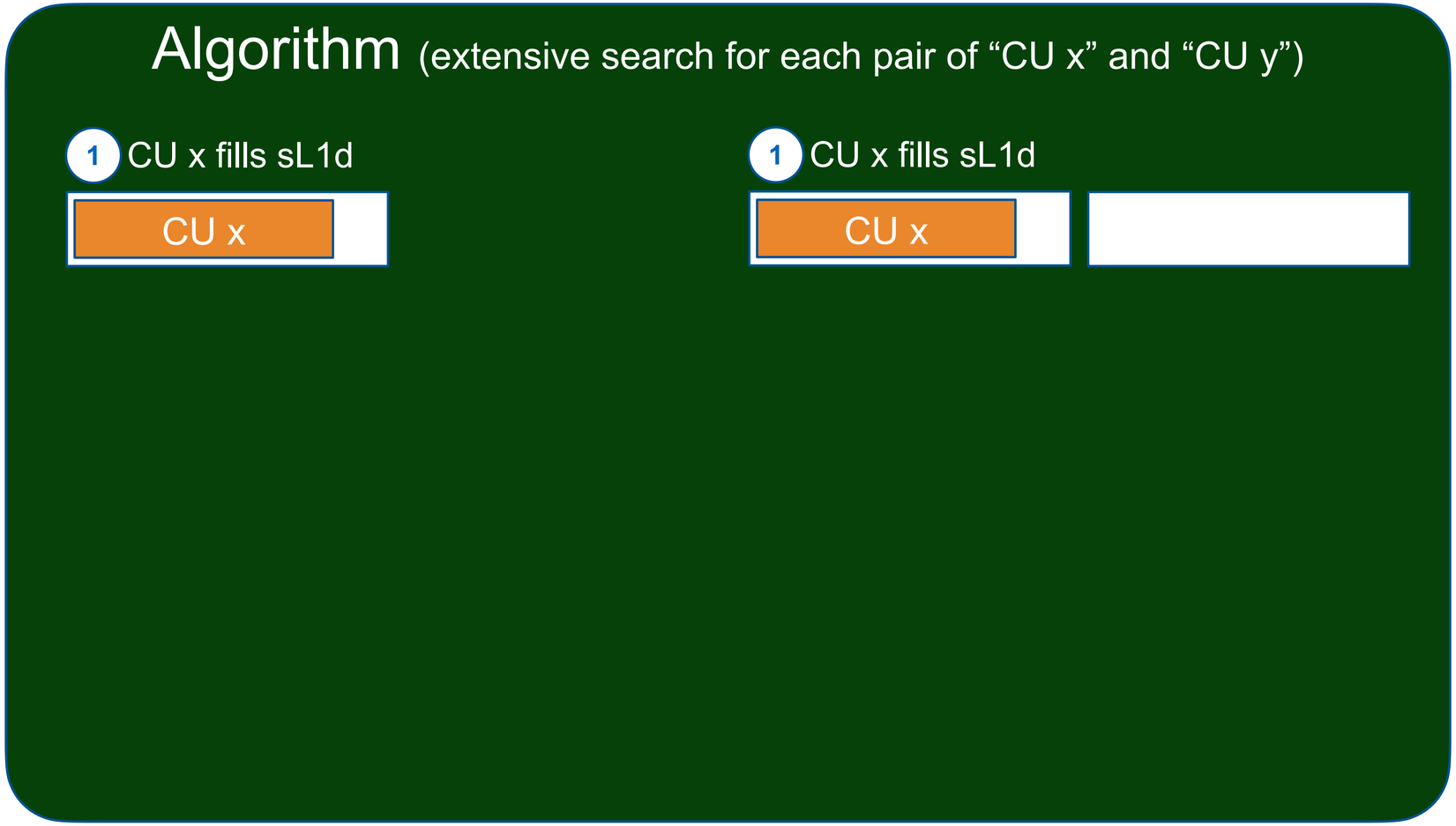
AMD CDNA architecture



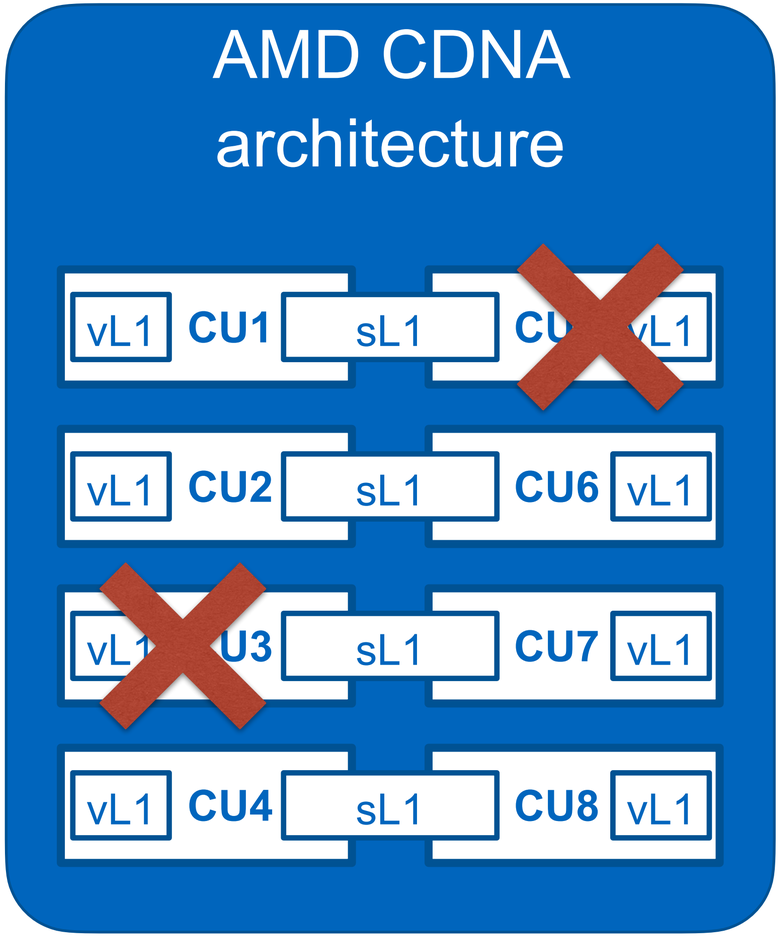
Algorithm (extensive search for each pair of “CU x” and “CU y”)

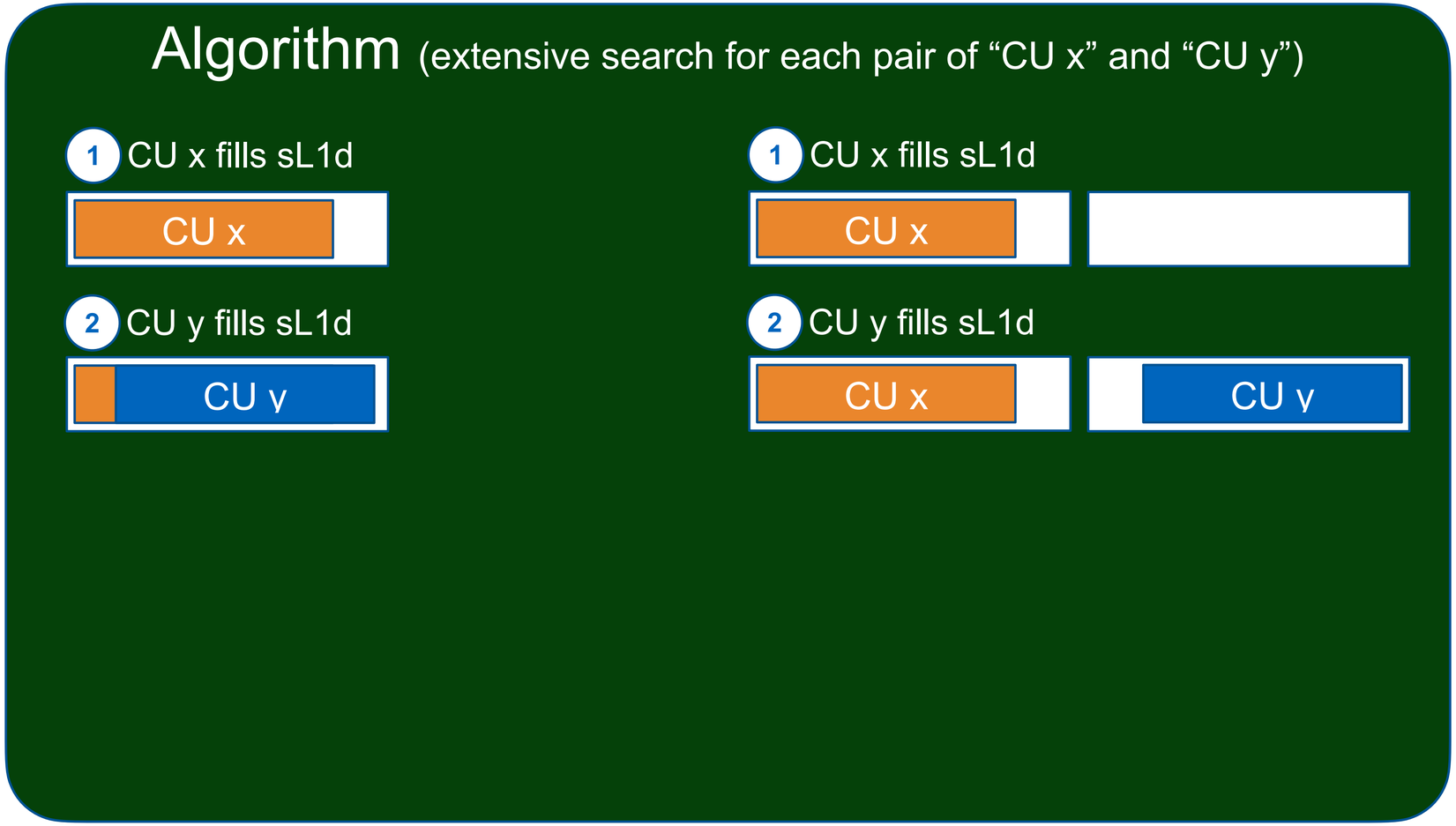
Microbenchmark deep-dive: AMD scalar L1 cache sharing





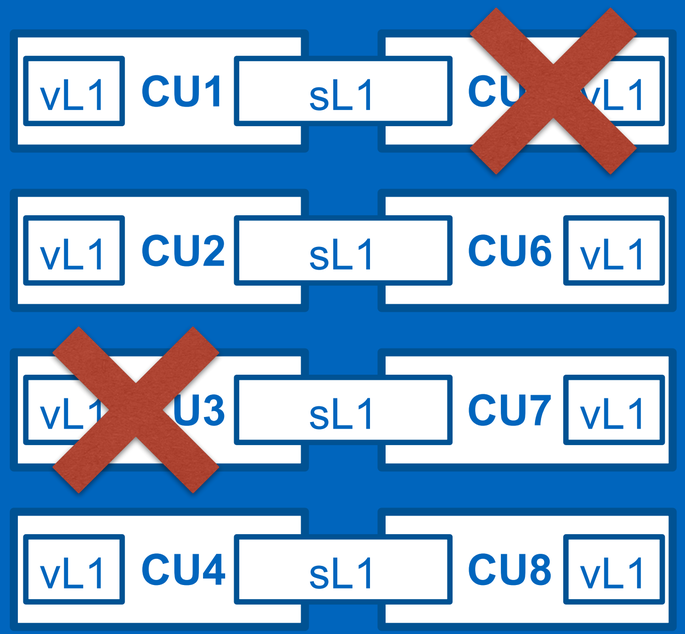
Microbenchmark deep-dive: AMD scalar L1 cache sharing





Microbenchmark deep-dive: AMD scalar L1 cache sharing

AMD CDNA architecture



Algorithm (extensive search for each pair of “CU x” and “CU y”)

1. CU x fills sL1d

2. CU y fills sL1d

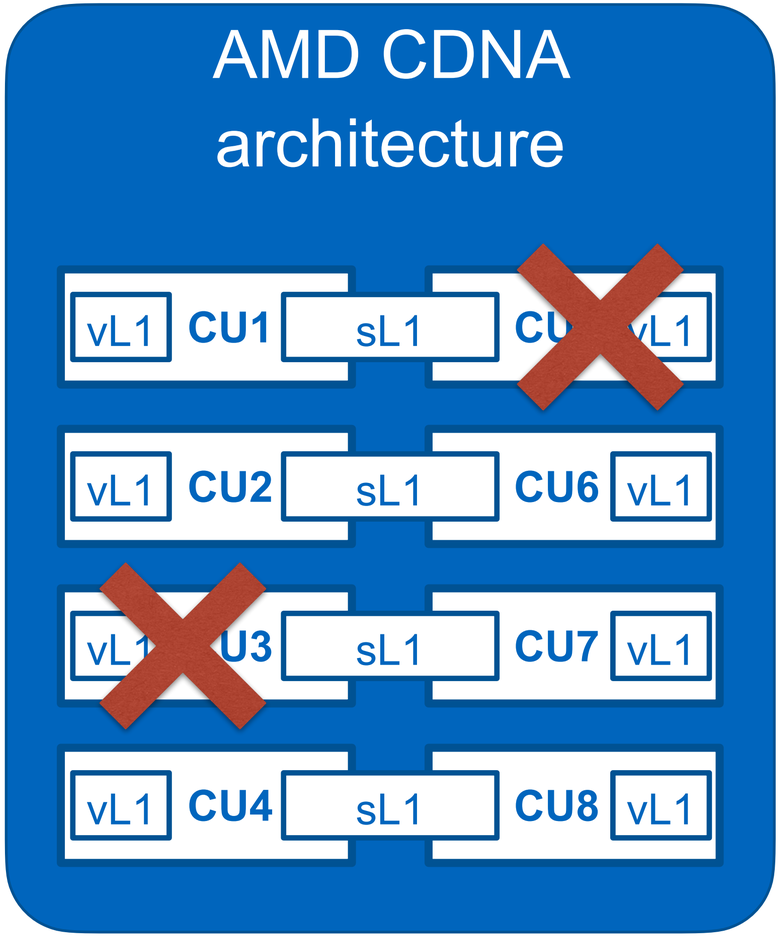
3. CU x misses on repeated access

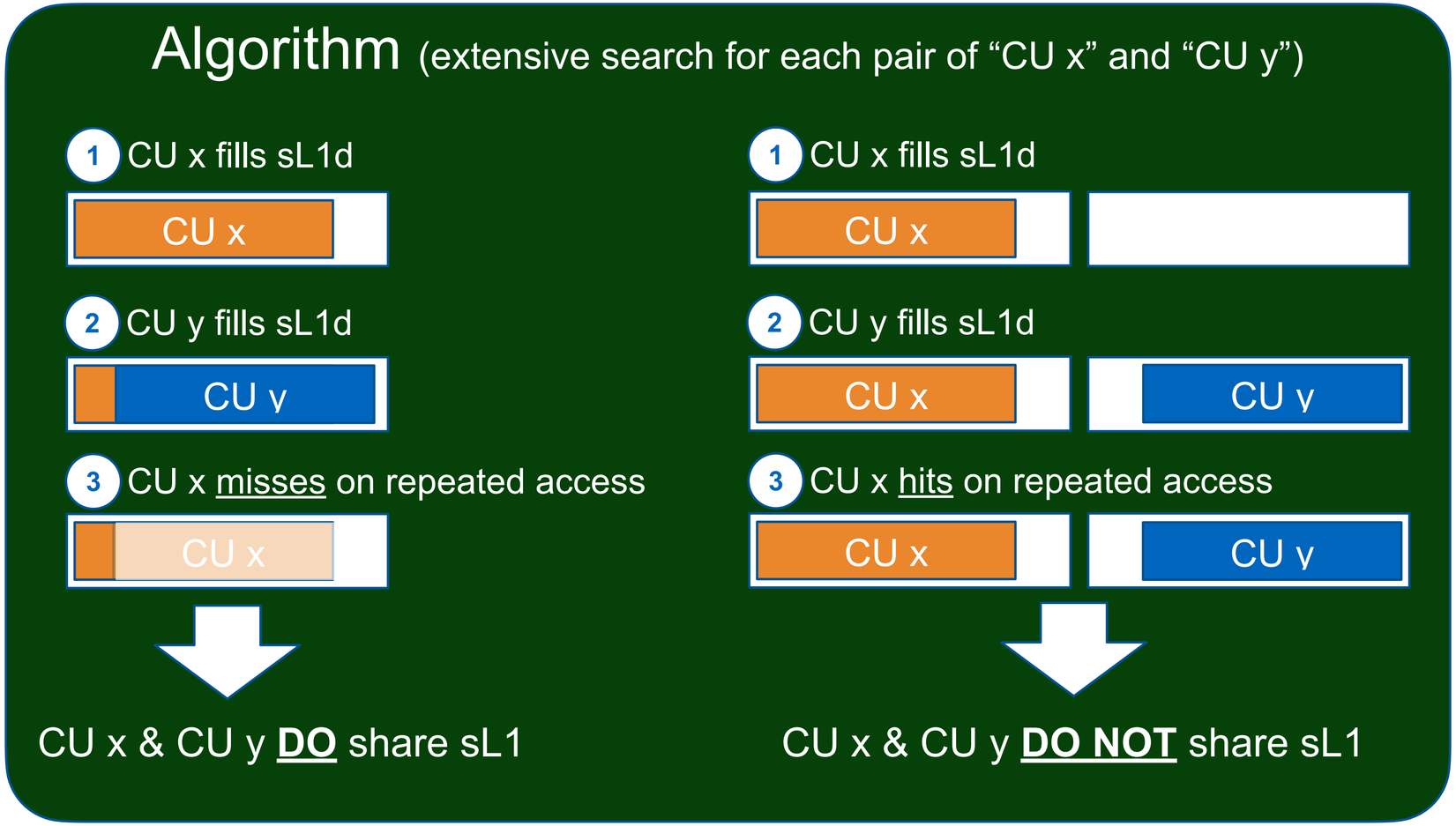
1. CU x fills sL1d

2. CU y fills sL1d

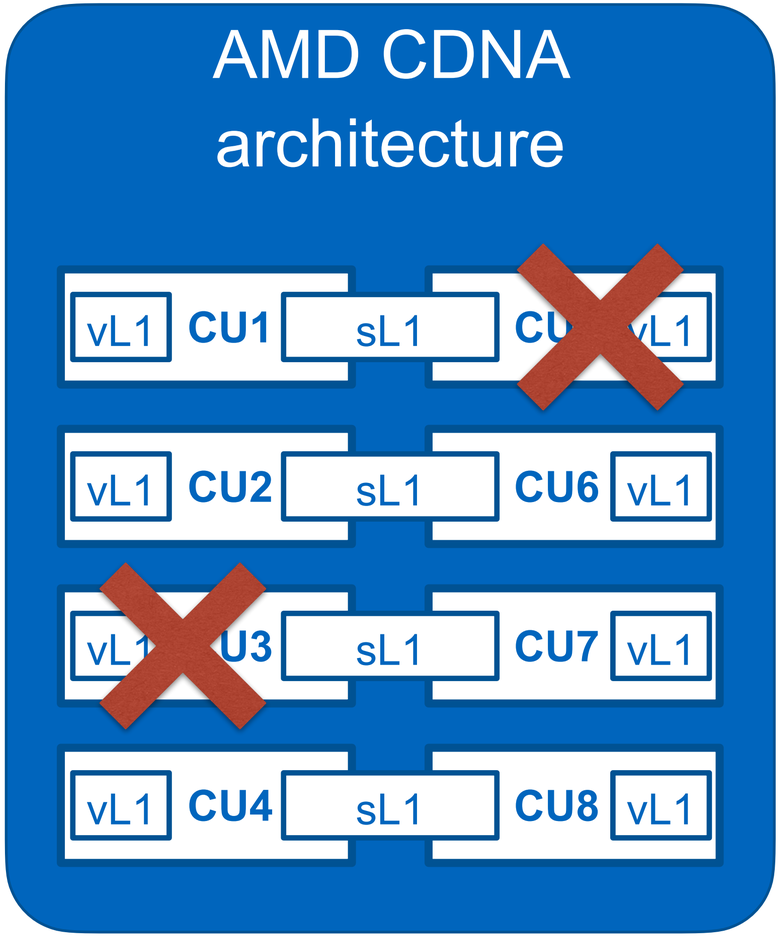
3. CU x hits on repeated access

Microbenchmark deep-dive: AMD scalar L1 cache sharing



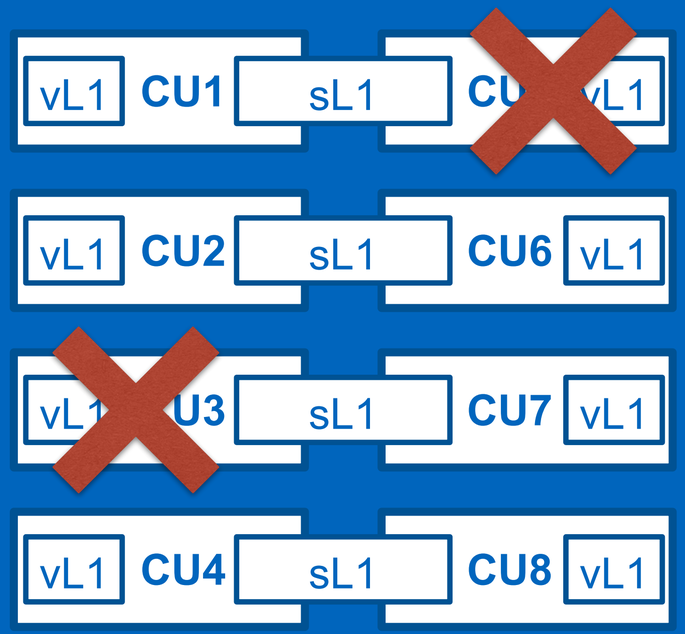


Microbenchmark deep-dive: AMD scalar L1 cache sharing



Microbenchmark deep-dive: AMD scalar L1 cache sharing

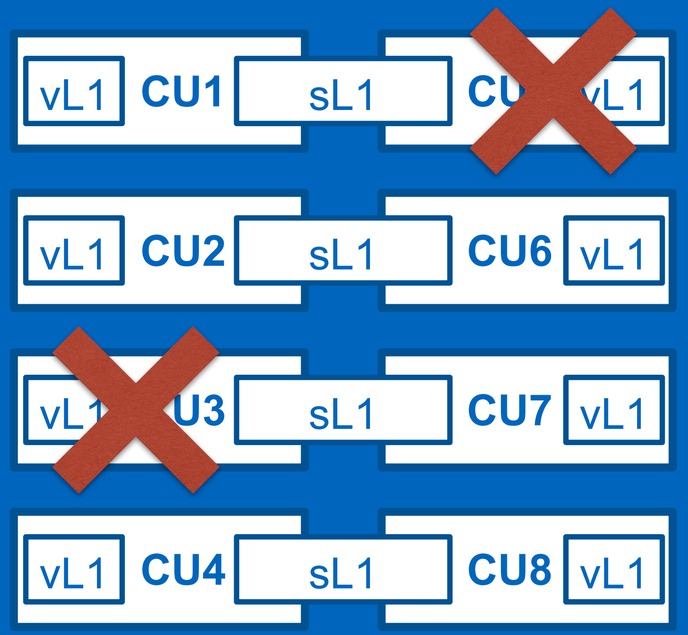
AMD CDNA architecture



Why is it interesting?

Microbenchmark deep-dive: AMD scalar L1 cache sharing

AMD CDNA architecture

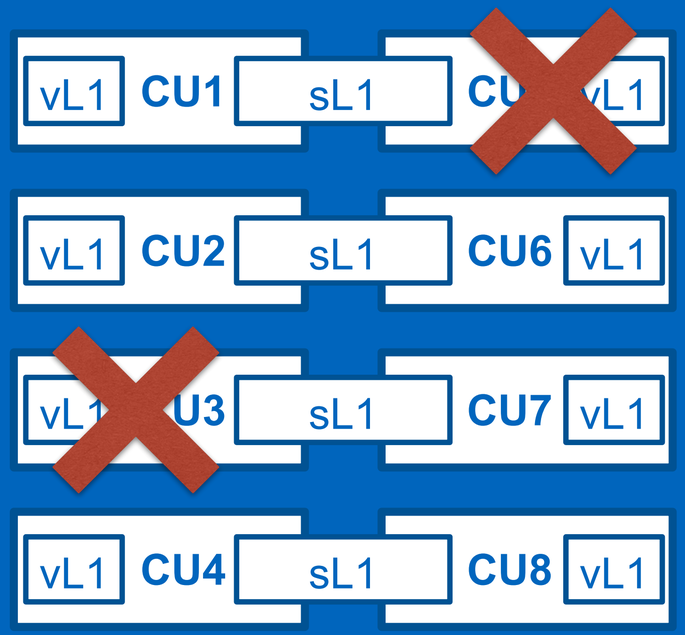


Why is it interesting?

1. How many bytes per CU available?

Microbenchmark deep-dive: AMD scalar L1 cache sharing

AMD CDNA architecture

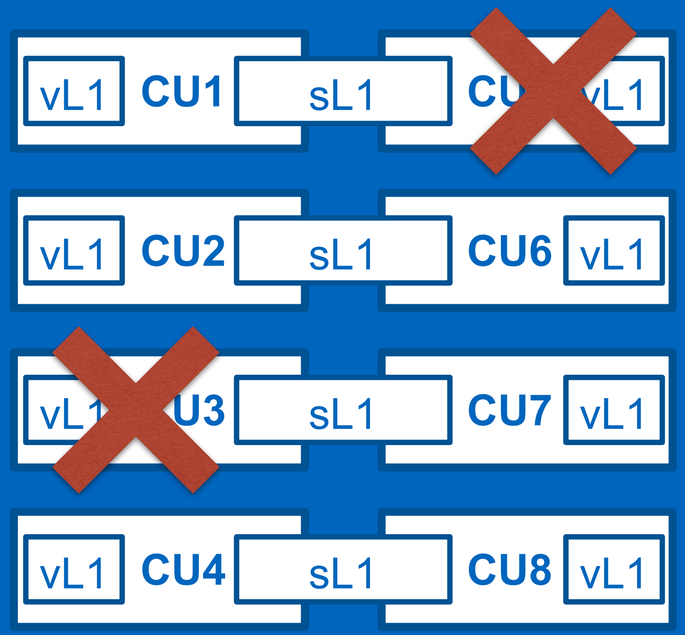


Why is it interesting?

1. How many bytes per CU available?
2. Schedule on CUs with large sL1 cache

Microbenchmark deep-dive: AMD scalar L1 cache sharing

AMD CDNA architecture



Why is it interesting?

1. How many bytes per CU available?
2. Schedule on CUs with large sL1 cache
3. Cross-CU data exchange

Validation

Tested and verified results on 10 GPUs

1. NVIDIA **Pascal** Quadro P6000
2. NVIDIA **Volta** V100
3. NVIDIA **Turing** T1000
   NVIDIA **Turing** RTX 2080 Ti
4. NVIDIA **Ampere** A100
5. NVIDIA **Hopper** H100 80 GB
   NVIDIA **Hopper** H100 96 GB

1. AMD **CDNA** Instinct MI-100
2. AMD **CDNA2** Instinct MI-210
3. AMD **CDNA3** Instinct MI-300X VF

Validation

NVIDIA H100

Memory Element	Size	LD lat	R&W BW	cache line size	fetch granularity	Amount per SM/GPU	Physical hardware sharing
L1 cache	**238 KiB**	38 ns		**128 B**	**32 B**	1	RO,TX,L1
L2 cache	*50 MB*	220 ns	4.4/3.4 TiB/s	**128 B**	**32 B**	2	
Texture cache	**238 KiB**	**39 ns**		**128 B**	**32 B**	**1**	RO,TX,L1
Readonly cache	**238 KiB**	**35 ns**		**128 B**	**32 B**	**1**	RO,TX,L1
Constant L1 cache	**2 KiB**	**21 ns**		**64 B**	**64 B**	**1**	**no**
Constant L1.5 cache	**> 64 KiB**	**105 ns**		❌	**256 B**		
Shared Memory	*228 KiB*	**30 ns**					
Device Memory	*80 GB*	843 ns	2.5/2.7 TiB/s				

Bold = new for this GPU; *Italic* = from an API; **Underlined** = new for this vendor; ❌ = not available

Validation

AMD MI210

Memory Element	Size	LD lat	R&W BW	cache line size	fetch granularity	Amount per SM/GPU	Physical hardware sharing
(v)L1 cache	16 KiB	125 ns		64 B	**64 B**	1	
sL1 cache	15.5 KiB	50 ns		**64 B**	**64 B**		**CU id**
L2 cache	*8 MB*	**310 ns**	4.2/2.4 TiB/s	128 B	**64 B**	1	
Local Data Store	*64 KiB*	55 ns					
Device Memory	64 GiB	**738 ns**	1.0/0.9 TiB/s				

Bold = new for this GPU; *Italic* = from an API; **Underlined** = new for this vendor;

Use-case: Dynamic GPU Topologies with sys-sage

Use-case: Dynamic GPU Topologies with sys-sage

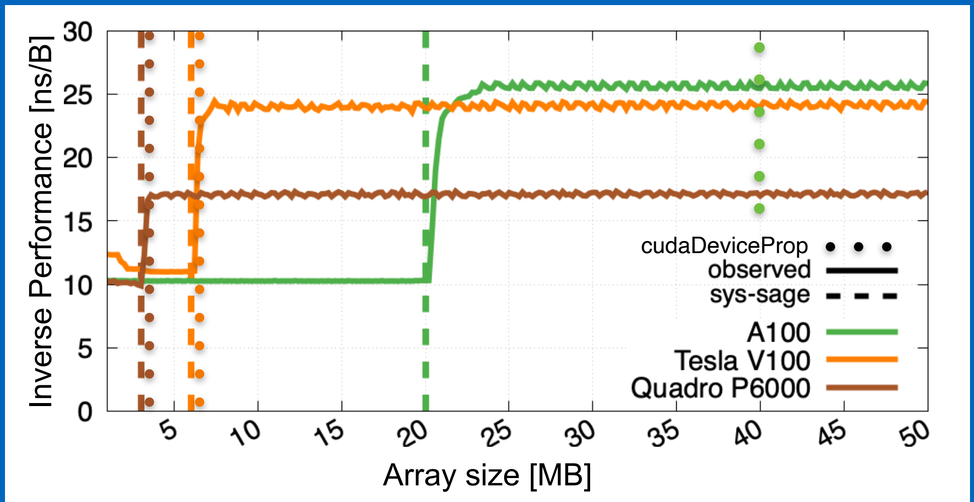
sys-sage integrates static and dynamic GPU topology information

- MT4G static topology report
- NVIDIA `nvml` API to read current MIG settings

Use-case: Dynamic GPU Topologies with sys-sage

sys-sage integrates static and dynamic GPU topology information

- MT4G static topology report
- NVIDIA `nvml` API to read current MIG settings

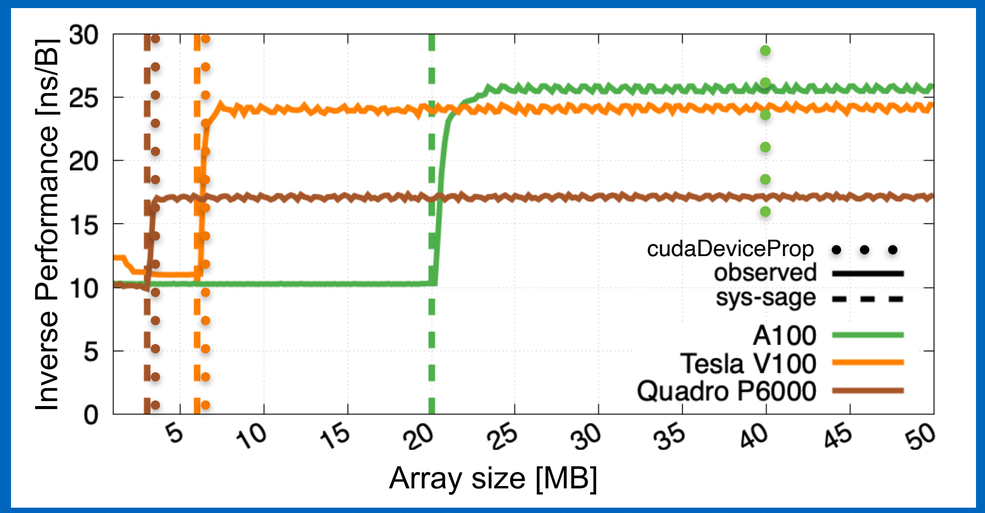


source: Vanecek et al., sys-sage: A Unified Representation of Dynamic Topologies & Attributes

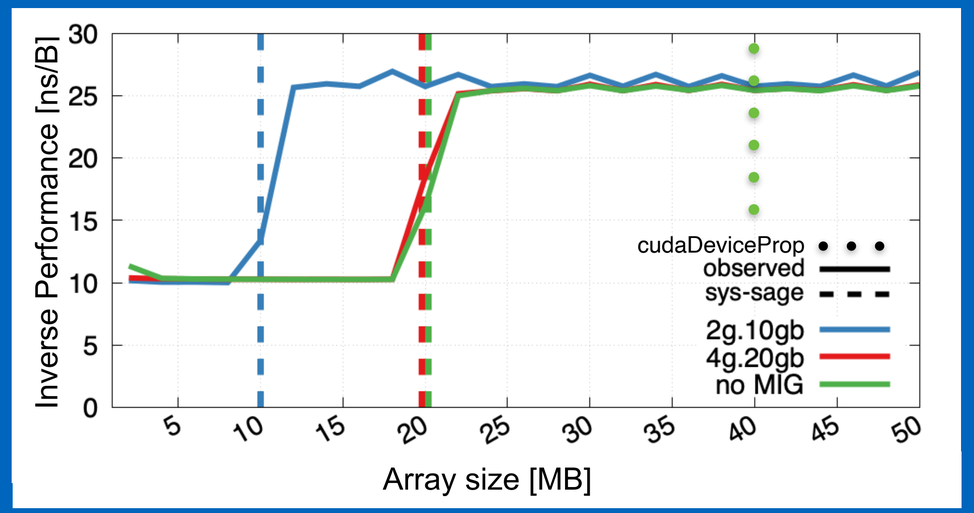
Use-case: Dynamic GPU Topologies with sys-sage

sys-sage integrates static and dynamic GPU topology information

- MT4G static topology report
- NVIDIA `nvml` API to read current MIG settings



source: Vanecek et al., sys-sage: A Unified Representation of Dynamic Topologies & Attributes



source: Vanecek et al., sys-sage: A Unified Representation of Dynamic Topologies & Attributes

MT4G: GPU Topology Information for NVIDIA and AMD GPUs

MT4G

- Open source
- Works with recent NVIDIA and AMD(CDNA) GPUs
- Provides rich topological information and attributes
- Microbenchmarking at the core
- Automatic evaluation (K-S test)
- Verified on 10 GPUs and 3 use-cases
- Machine+Human readable JSON output

Provides information on

- General GPU information
- Compute Topologies
- Cache and Memory subsystem
  - Size, cache line size, HW layout, …
  - Load latency, R&W bandwidth

Try MT4G out and reach out to us!

https://github.com/caps-tum/mt4g

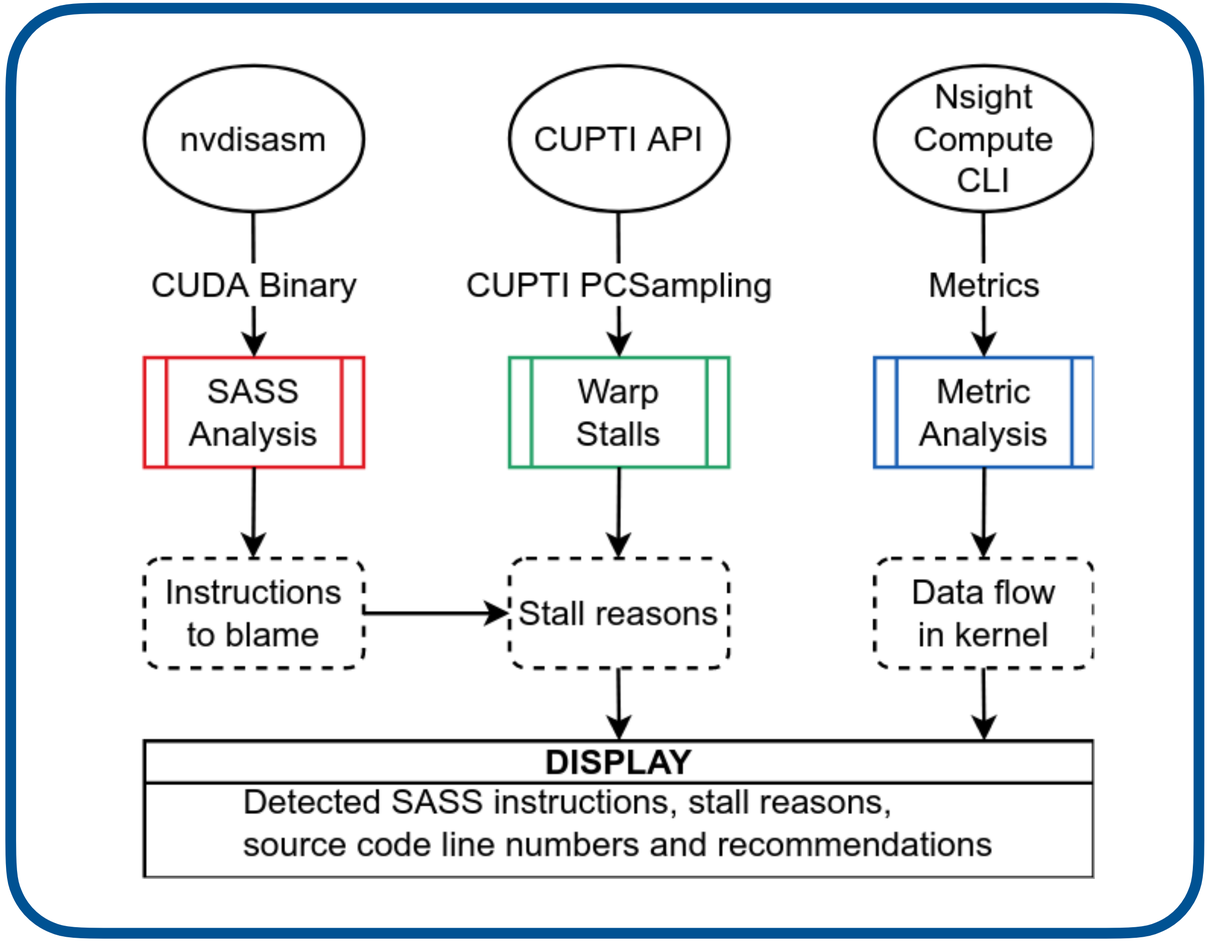
stepan.vanecek@tum.de

Acknowledgements

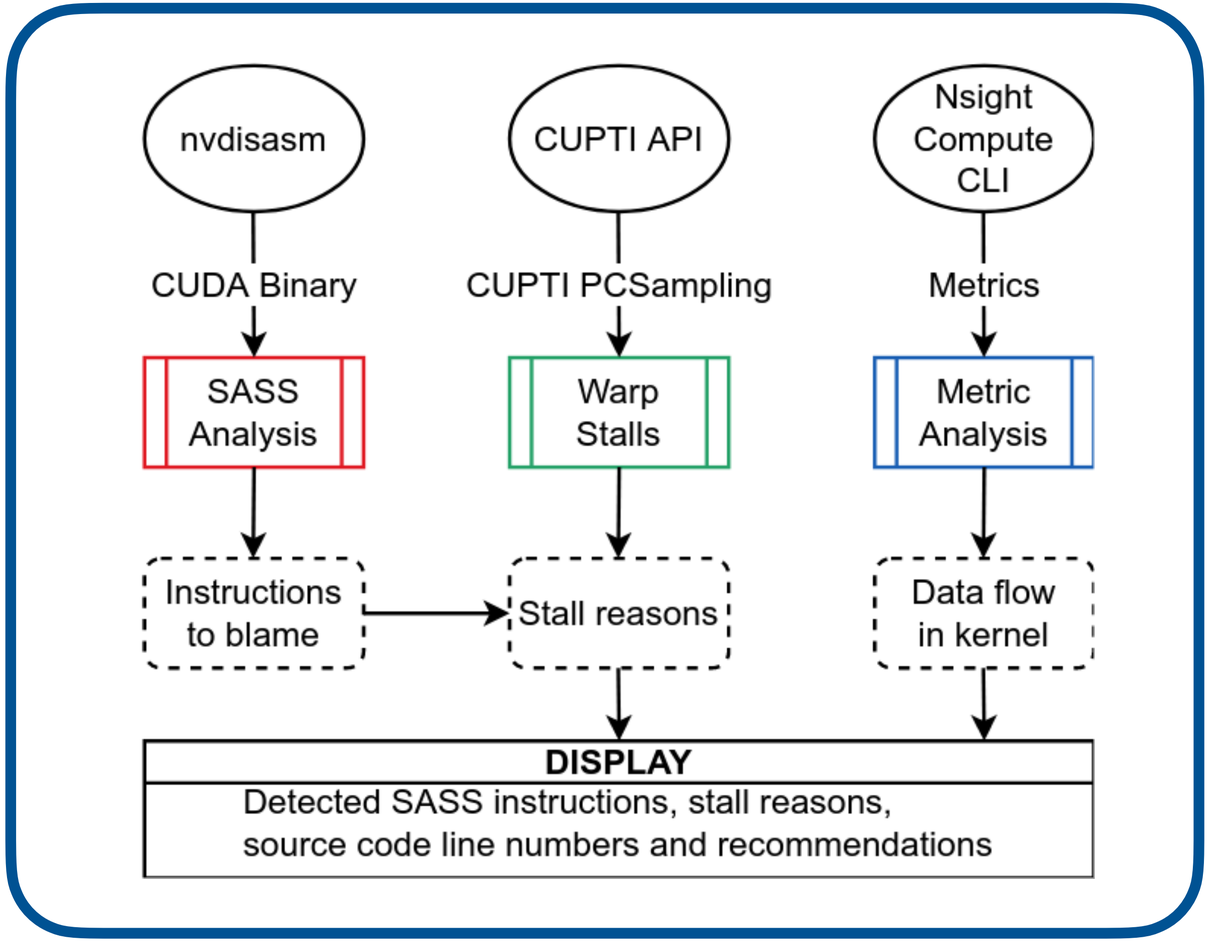
SEANERGYS Energy Efficient Exascale
PDExa
PLASMA PEPSC
DEEP-SEA
SPONSORED BY THE
Federal Ministry of Education and Research

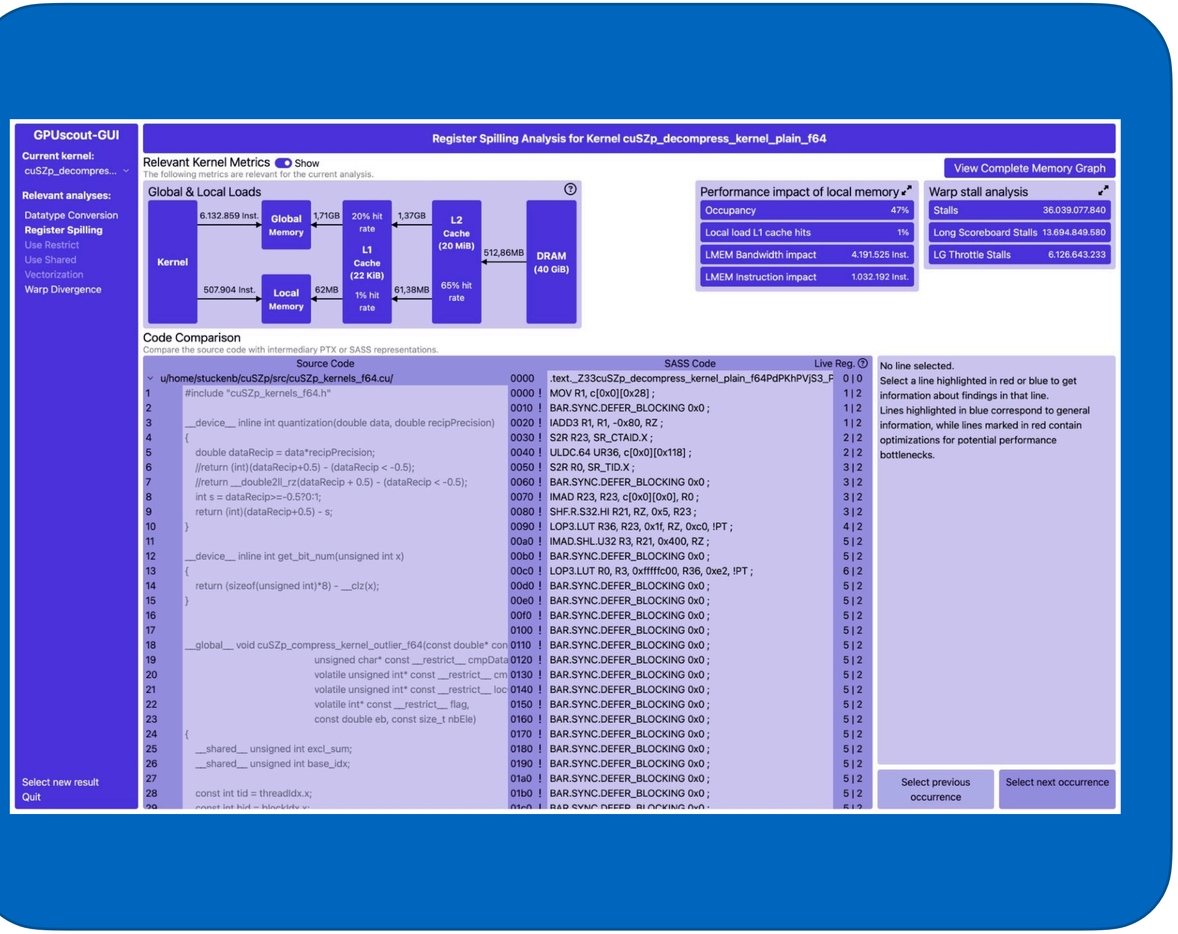
Use-case: Adding HW context to GPUscout

Use-case: Adding HW context to GPUscout

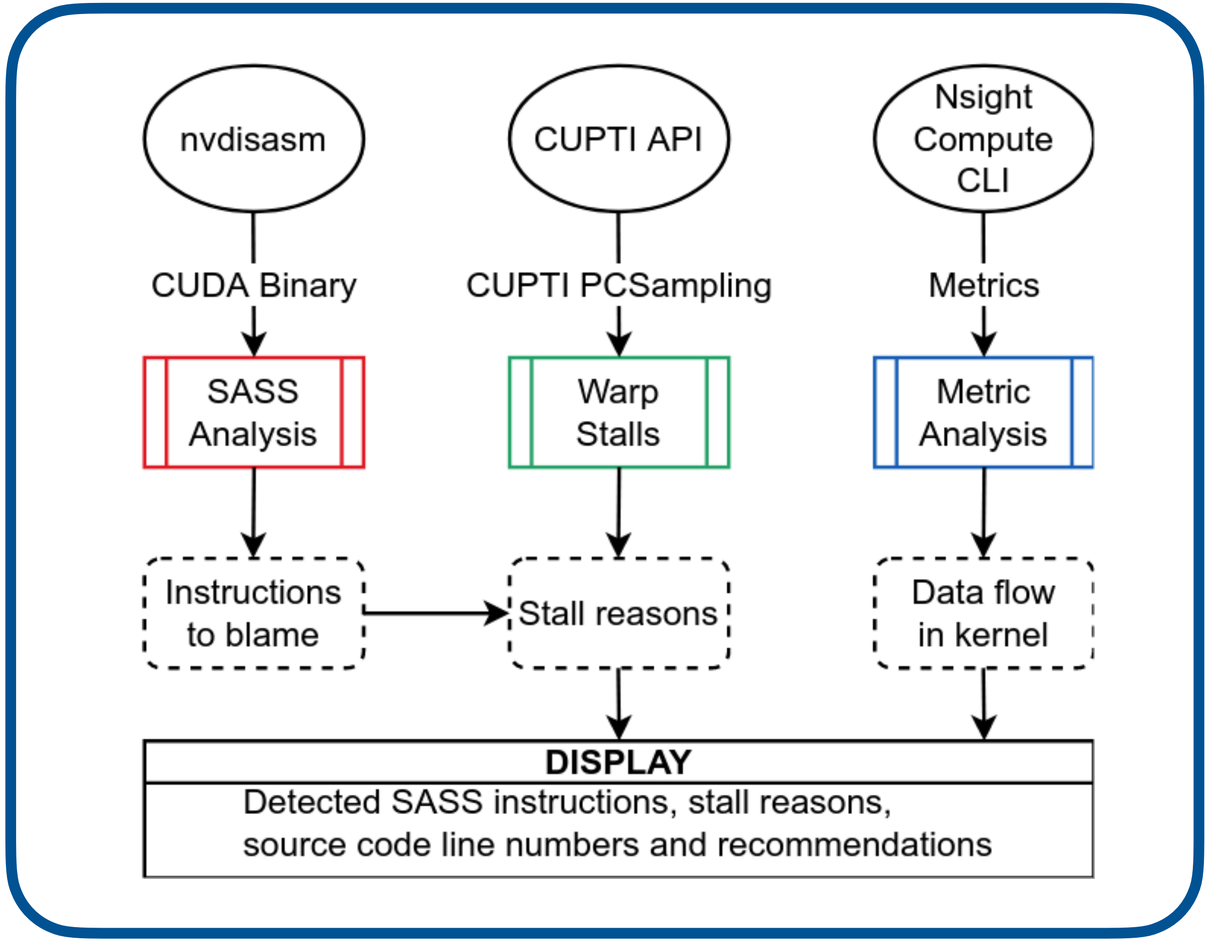


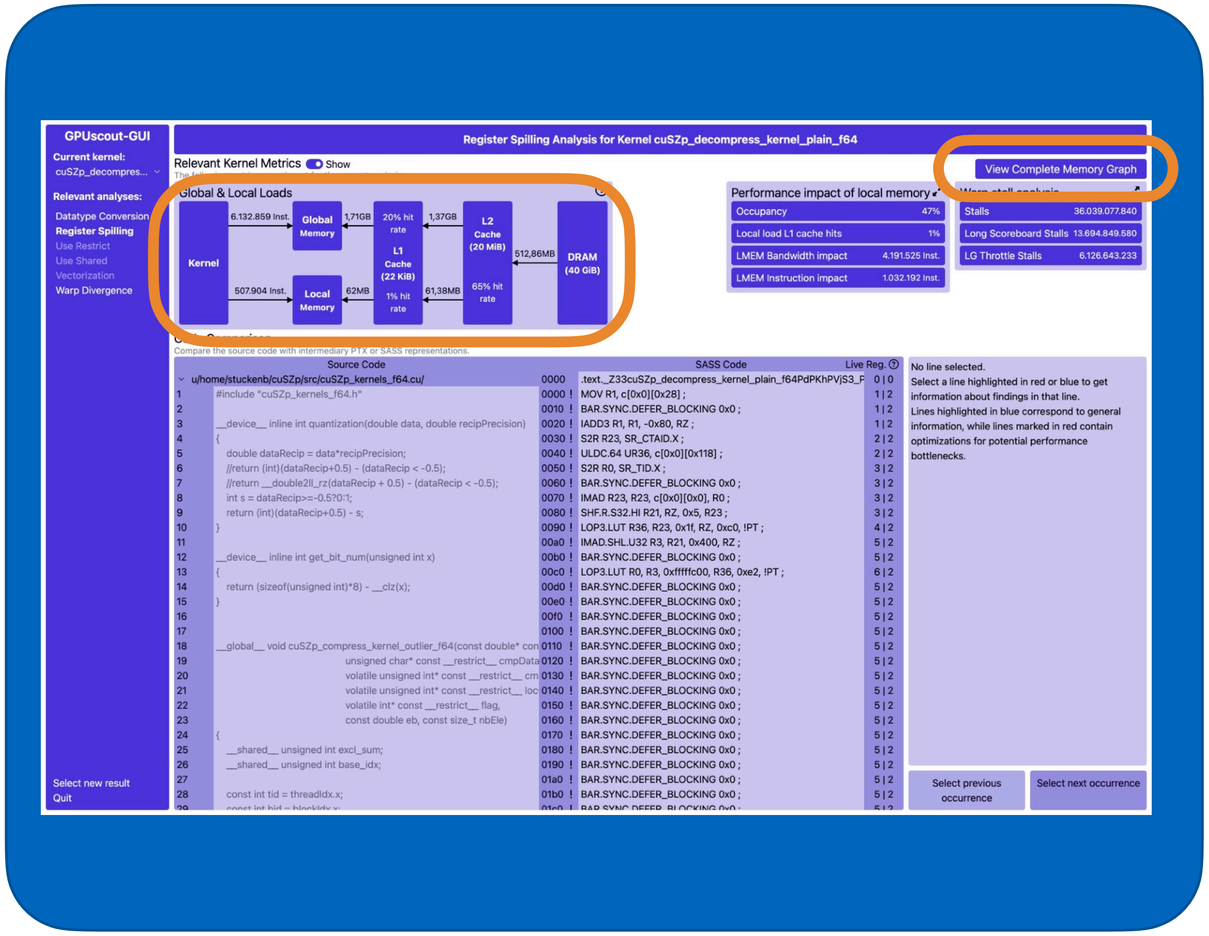
Use-case: Adding HW context to GPUscout



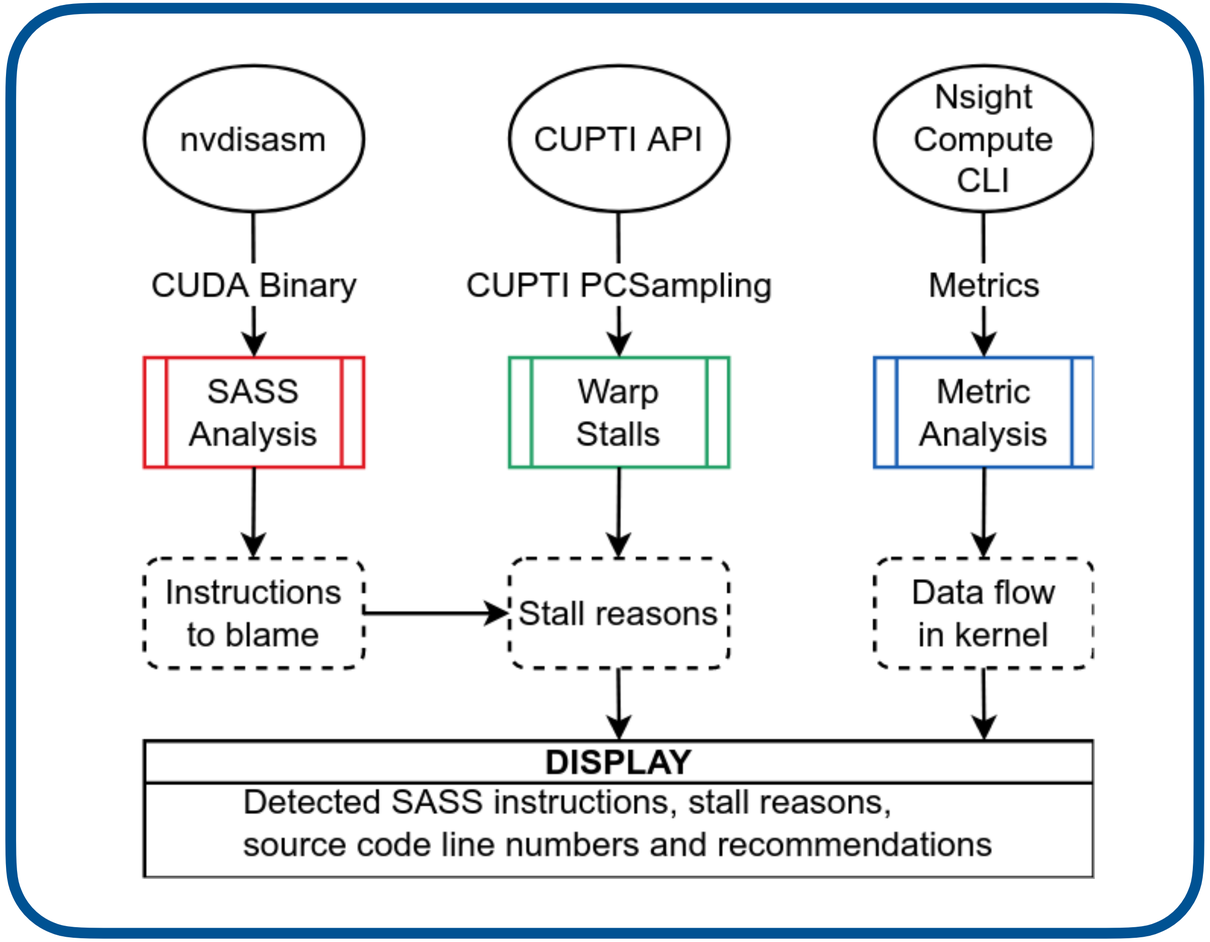


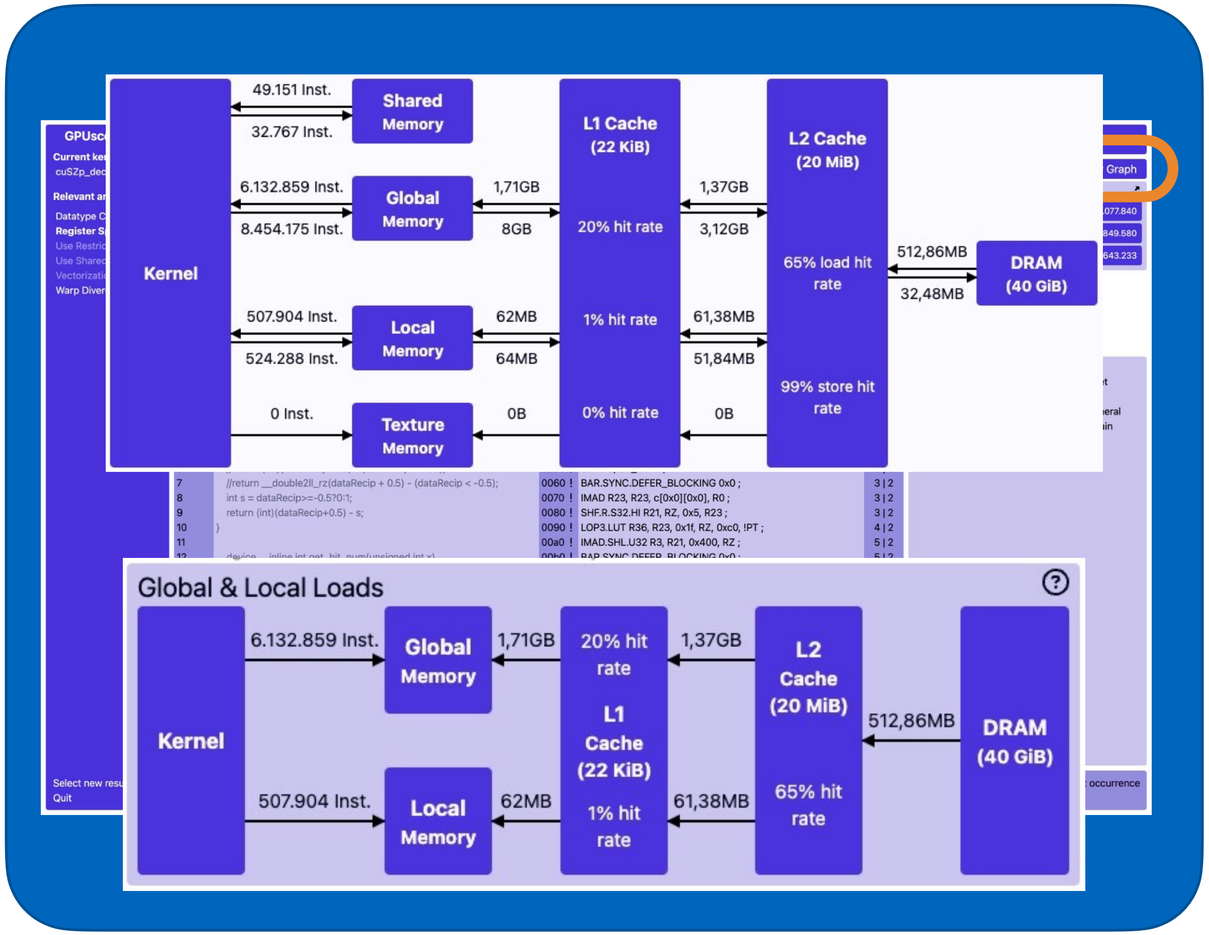
Use-case: Adding HW context to GPUscout





Use-case: Adding HW context to GPUscout





Use-case: Adding HW context to GPUscout