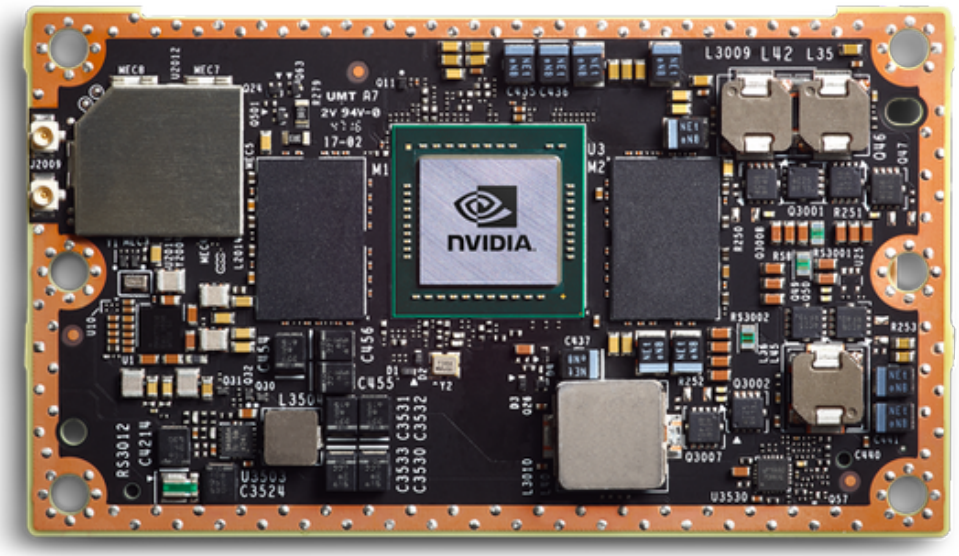
Experiments for Time-Predictable Execution of GPU Kernels

Flavio Kreiliger, Joel Matějka, **Michal Sojka**
and Zdeněk Hanzálek

OSPERT 2019
July 9, 2019,
Stuttgart, Germany

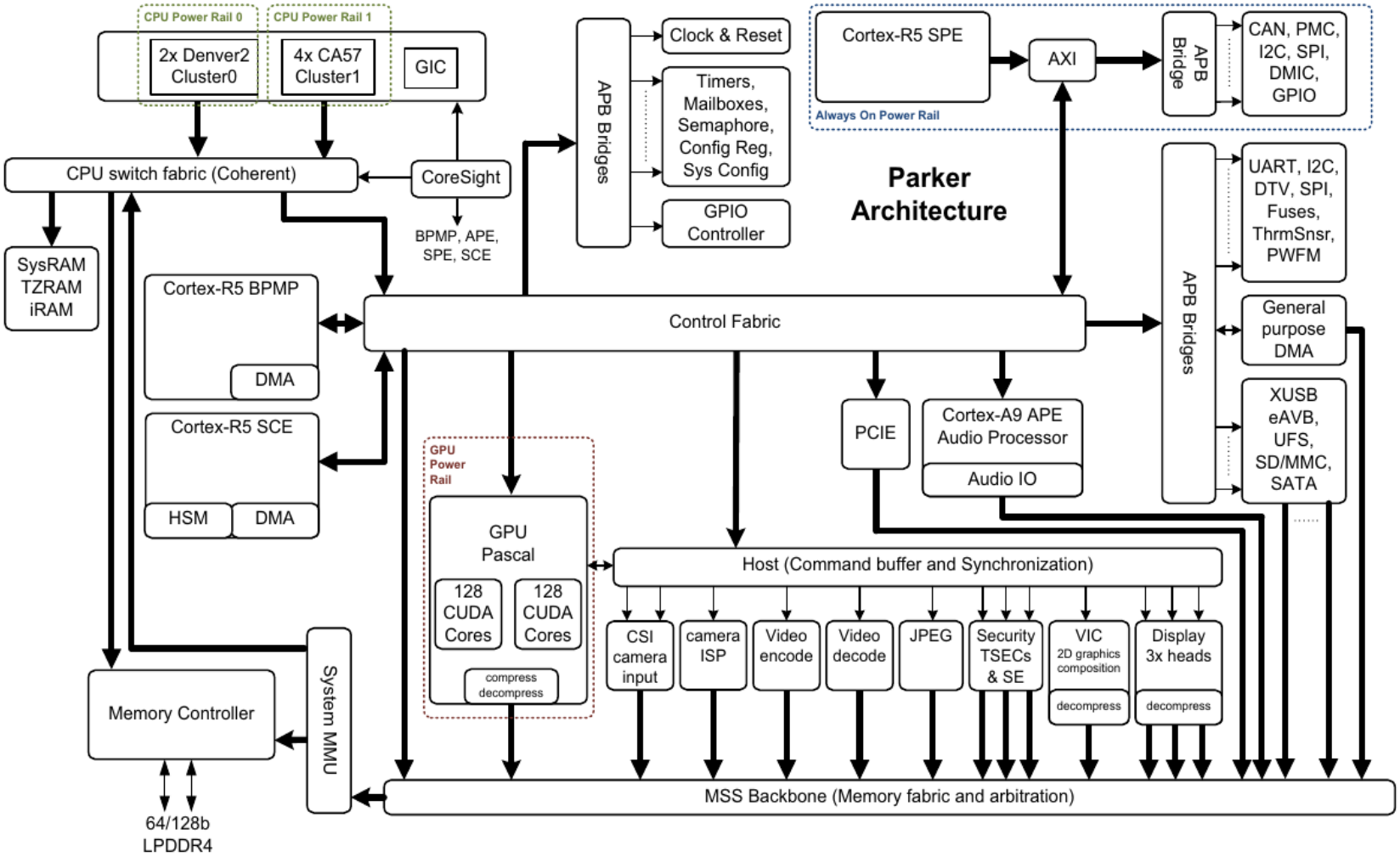
NVIDIA Tegra X2



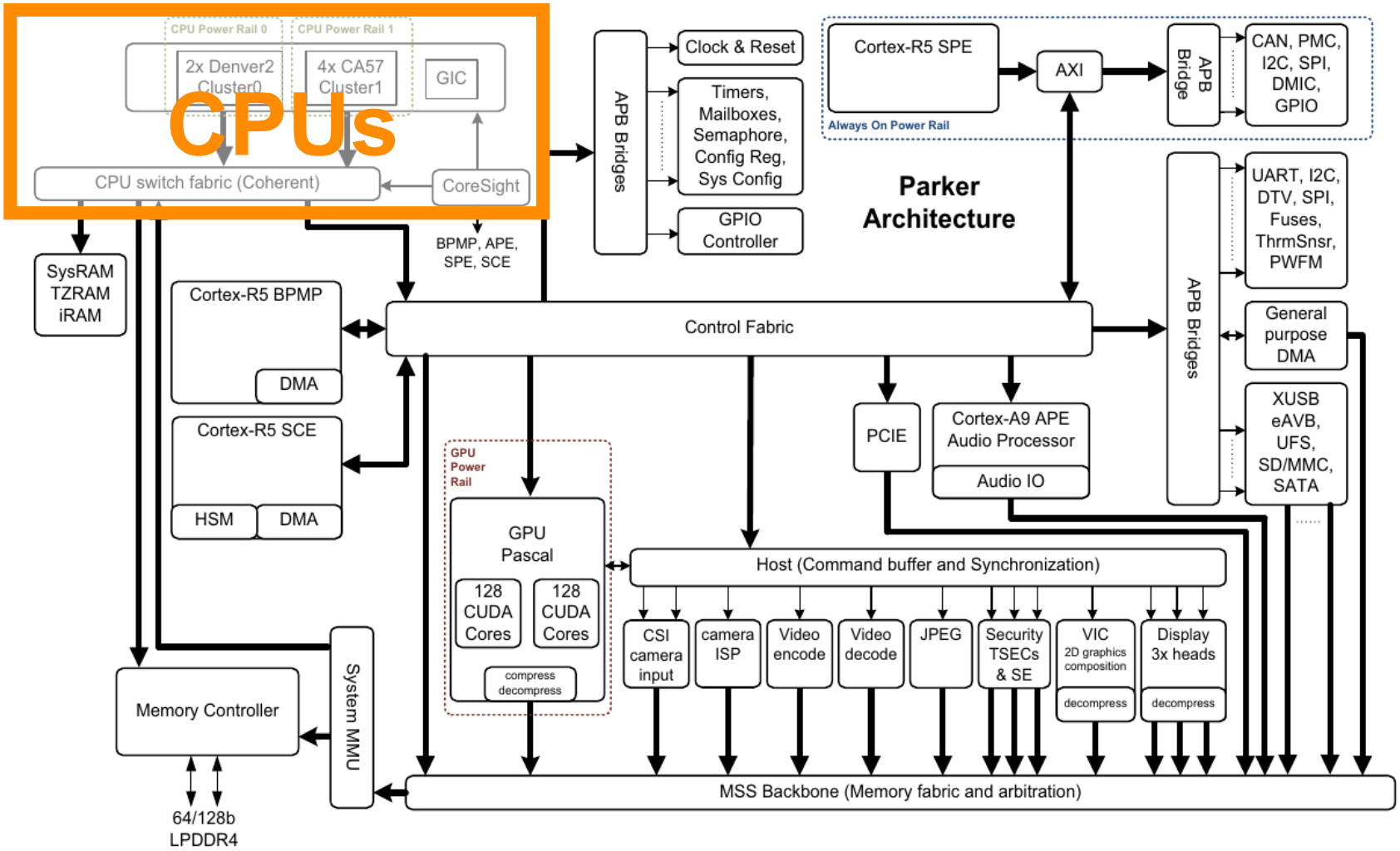
- CPUs: 4× ARM Cortex A57, 2× Denver (ARM/NVIDIA)
- GPU: 256 CUDA cores in 2 streaming multiprocessors (SM)

Outline

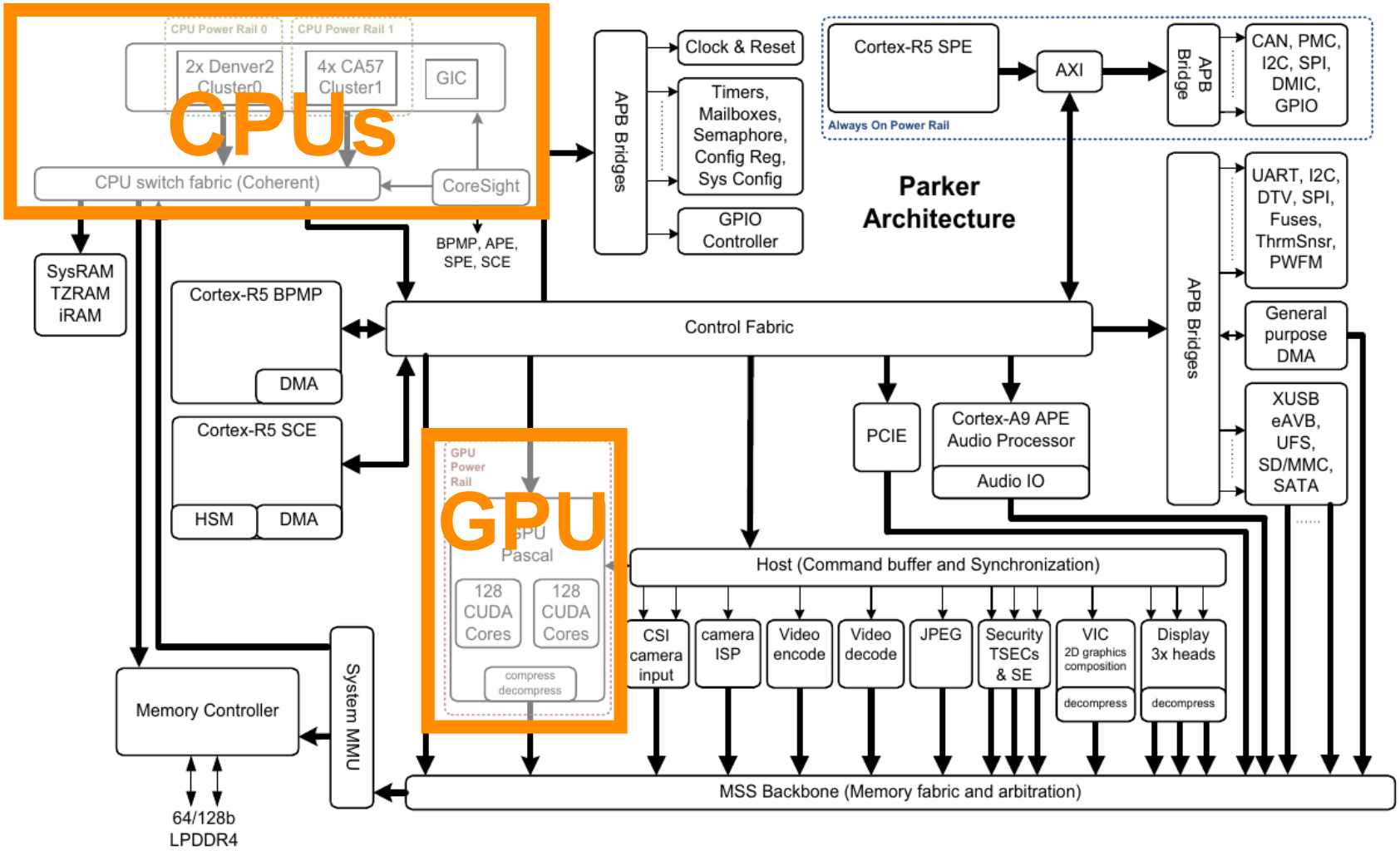
NVIDIA Tegra X2 block diagram



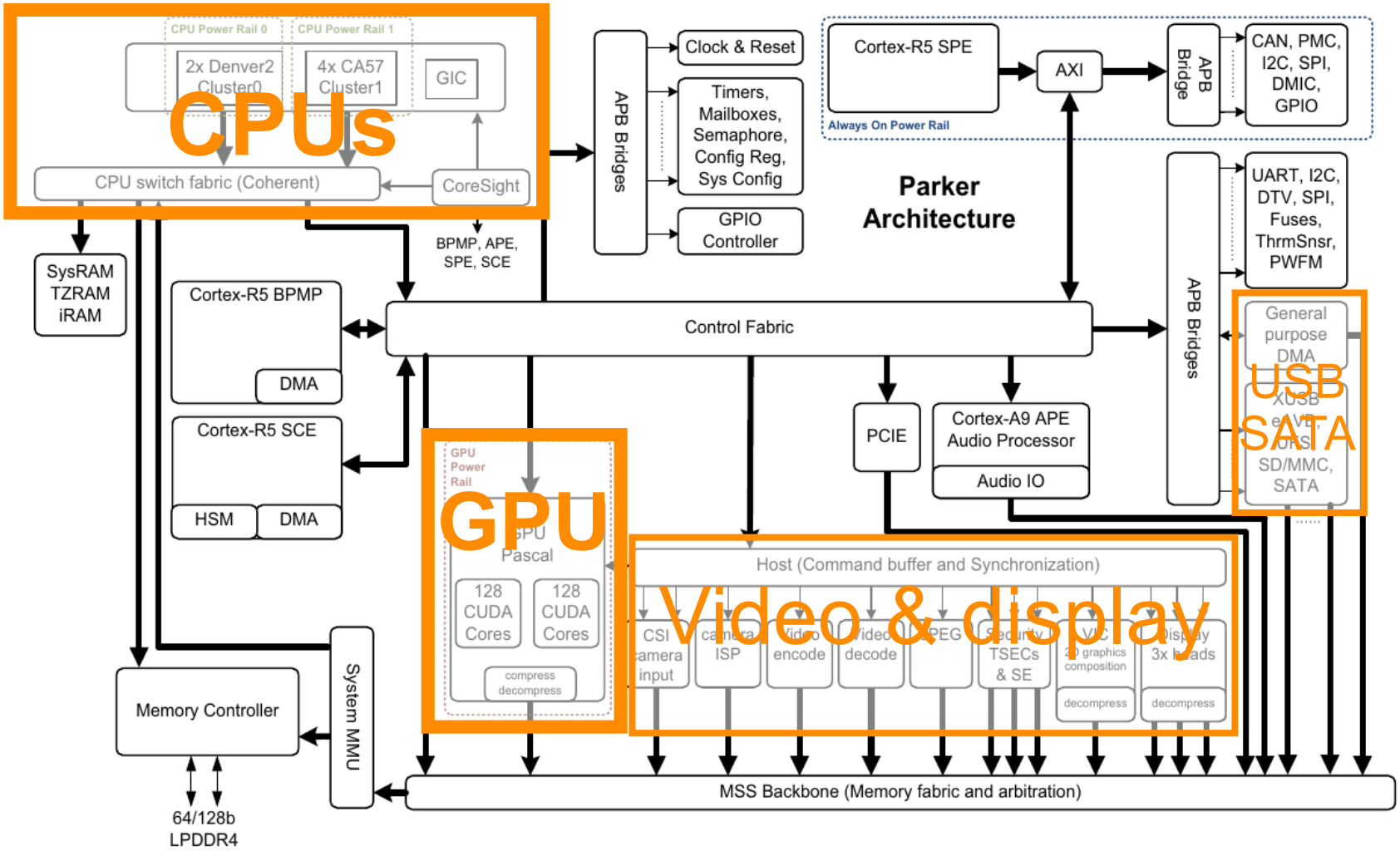
NVIDIA Tegra X2 block diagram



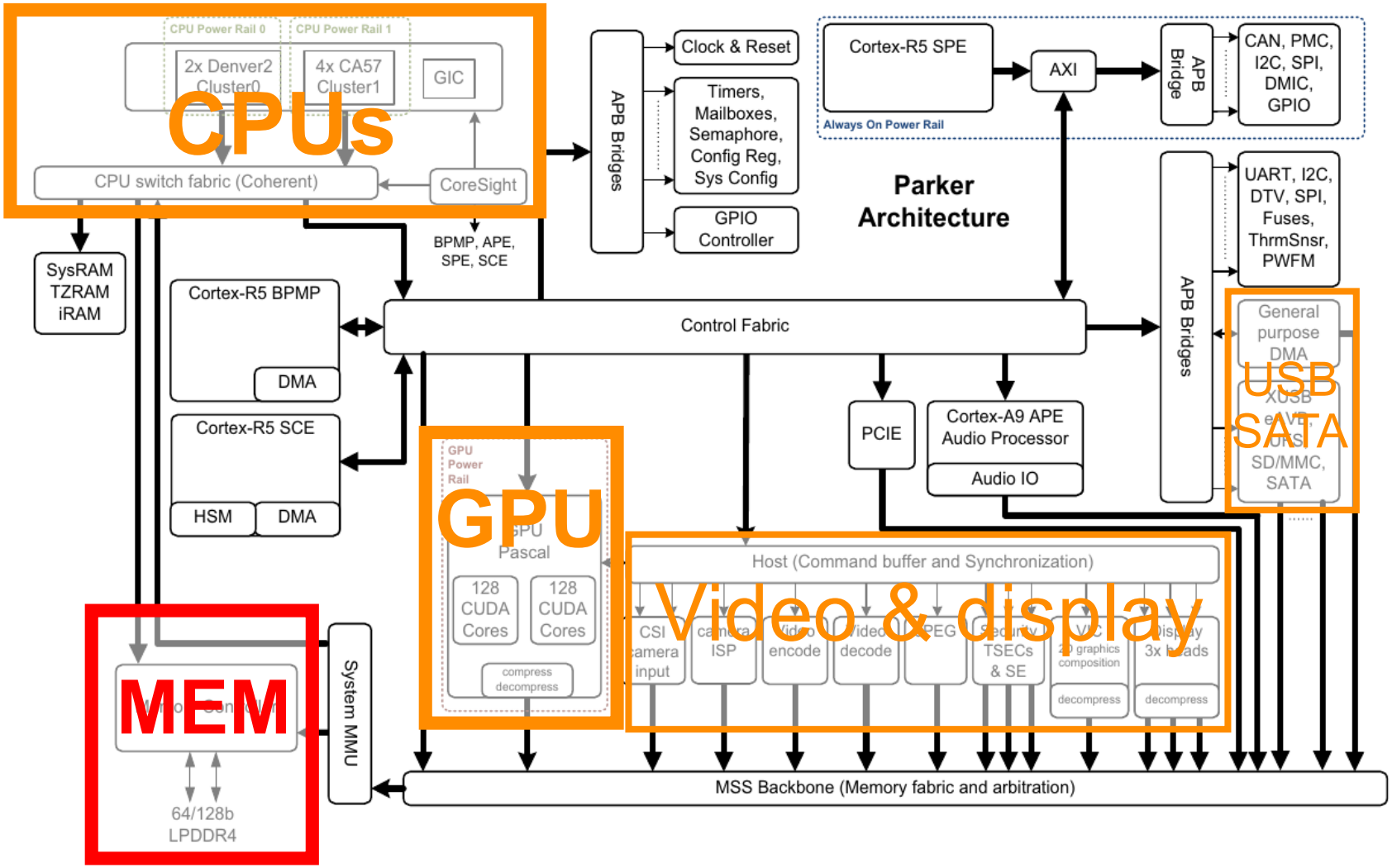
NVIDIA Tegra X2 block diagram



NVIDIA Tegra X2 block diagram

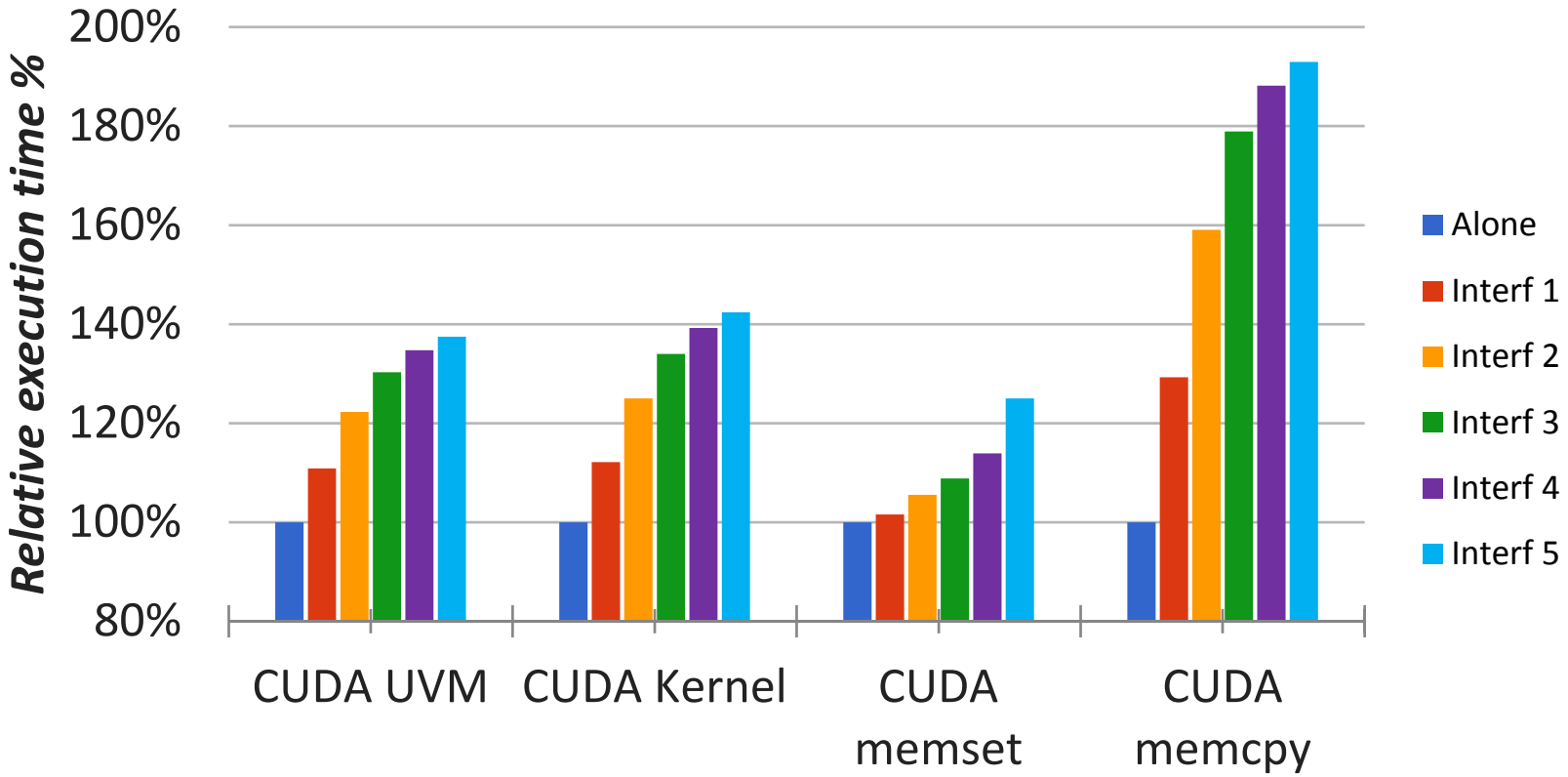


NVIDIA Tegra X2 block diagram



GPU execution times under CPU interference

Tegra X2, CPUs performing sequential memory accesses



Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

Safety-Critical applications

E.g. autonomous driving

- Future application will need to combine safety and high performance
- Typically, only some parts of the system are safety-critical
- **Goal:** isolate critical parts from non-critical ones
  - Failure in non-critical component should not propagate to a critical one

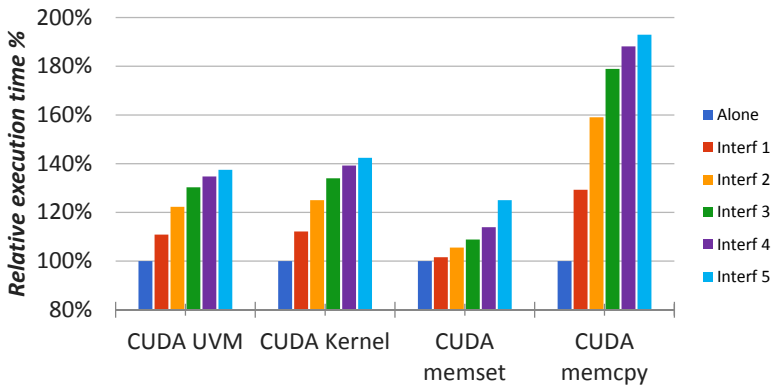
Safety-Critical applications

E.g. autonomous driving

- Future application will need to combine safety and high performance
- Typically, only some parts of the system are safety-critical
- **Goal:** isolate critical parts from non-critical ones
  - Failure in non-critical component should not propagate to a critical one
- ISO26262: **Freedom from interference**

Interference on TX2

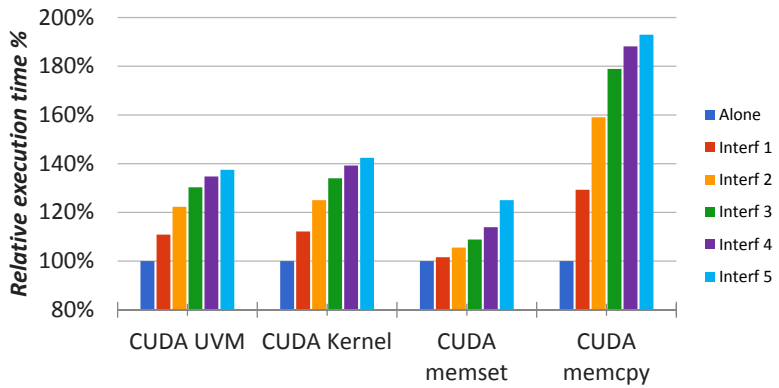
1. CPU-to-GPU



Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

Interference on TX2

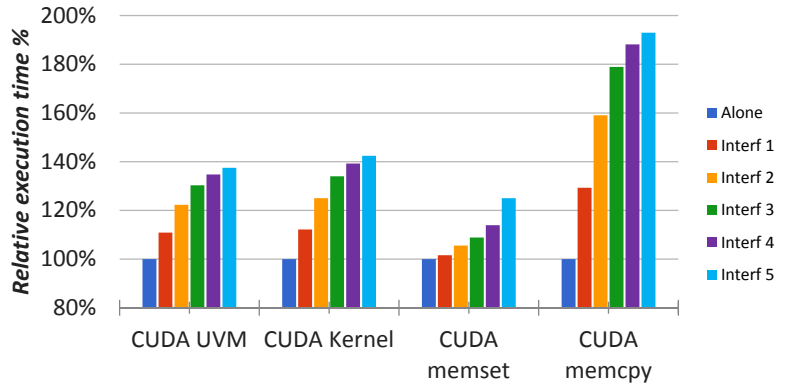
1. CPU-to-GPU
2. GPU-to-CPU

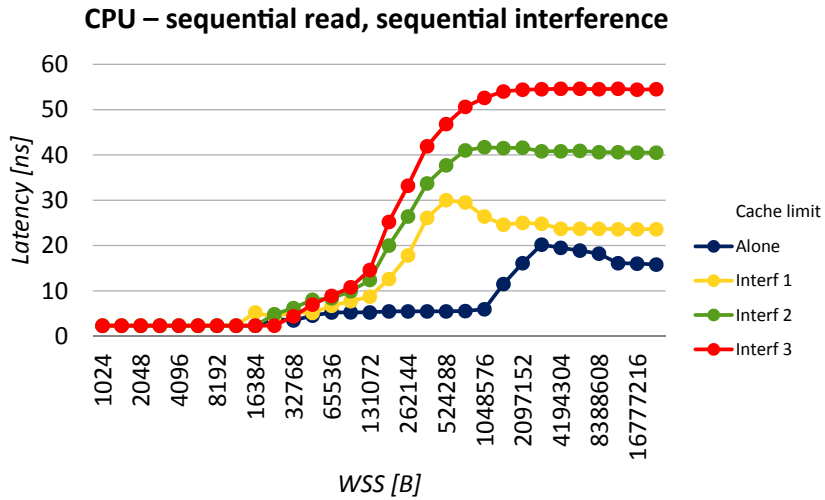


Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

Interference on TX2

1. CPU-to-GPU
2. GPU-to-CPU
3. CPU-to-CPU

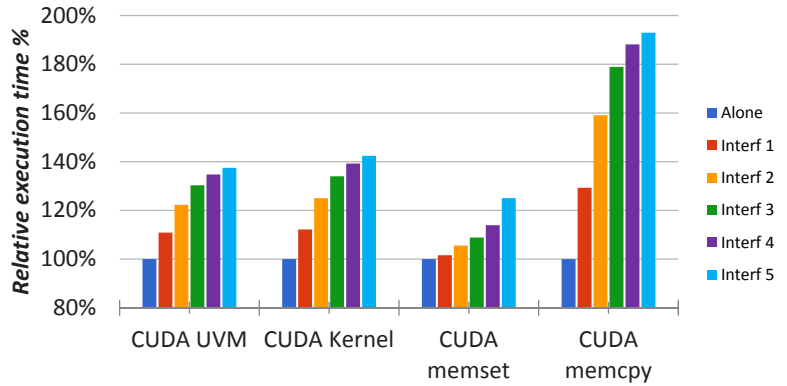


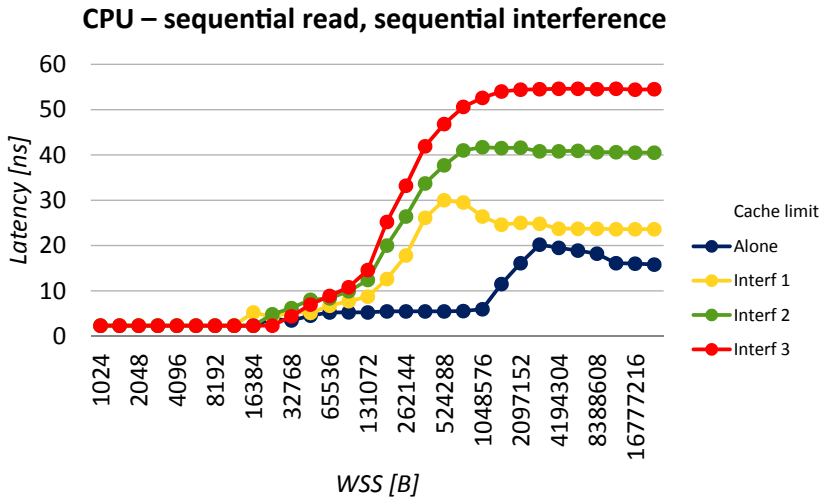


Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

Interference on TX2

1. CPU-to-GPU
2. GPU-to-CPU
3. CPU-to-CPU
4. GPU-to-GPU

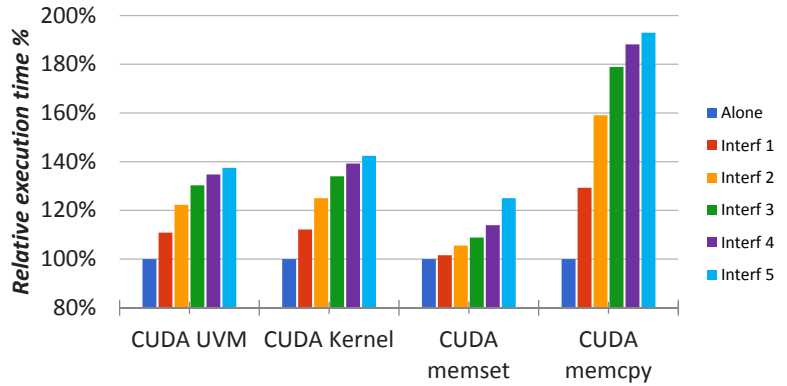


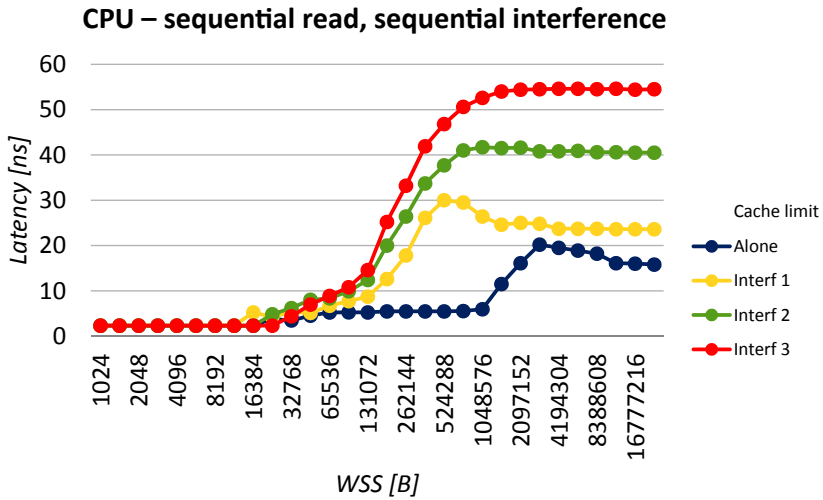


Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

Interference on TX2

1. CPU-to-GPU
2. GPU-to-CPU
3. CPU-to-CPU
4. GPU-to-GPU

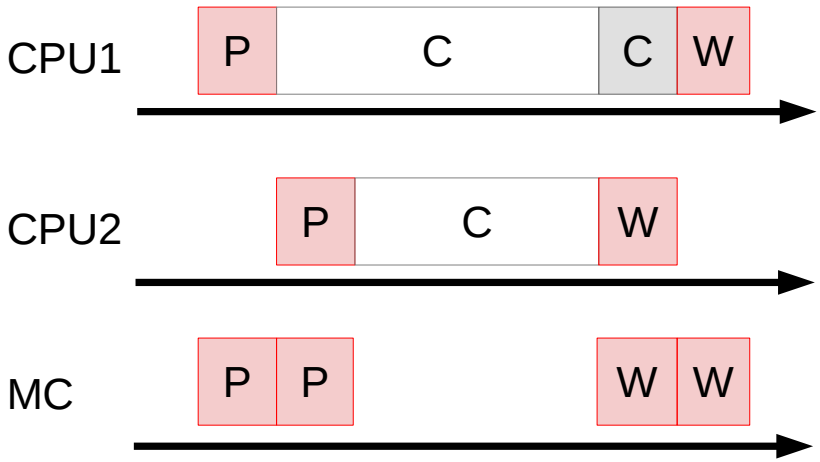




Source: Capodieci et al., *Detailed characterization of platforms*, Deliverable D2.2, H2020 project HERCULES, 2017.

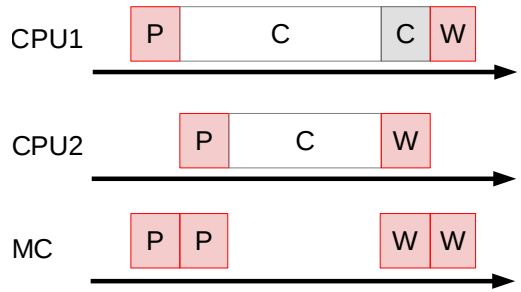
CPU-to-CPU interference

- Possible solution (a part of): **PRedictable Execution Model** (PREM)
- Tasks prefetch batches of data to CPU-local memory (cache/scratchpad) and synchronize on access to main memory
- Well applicable to number-crunching applications:
  - Image processing
  - Neural networks
- GPUs are better suited for these

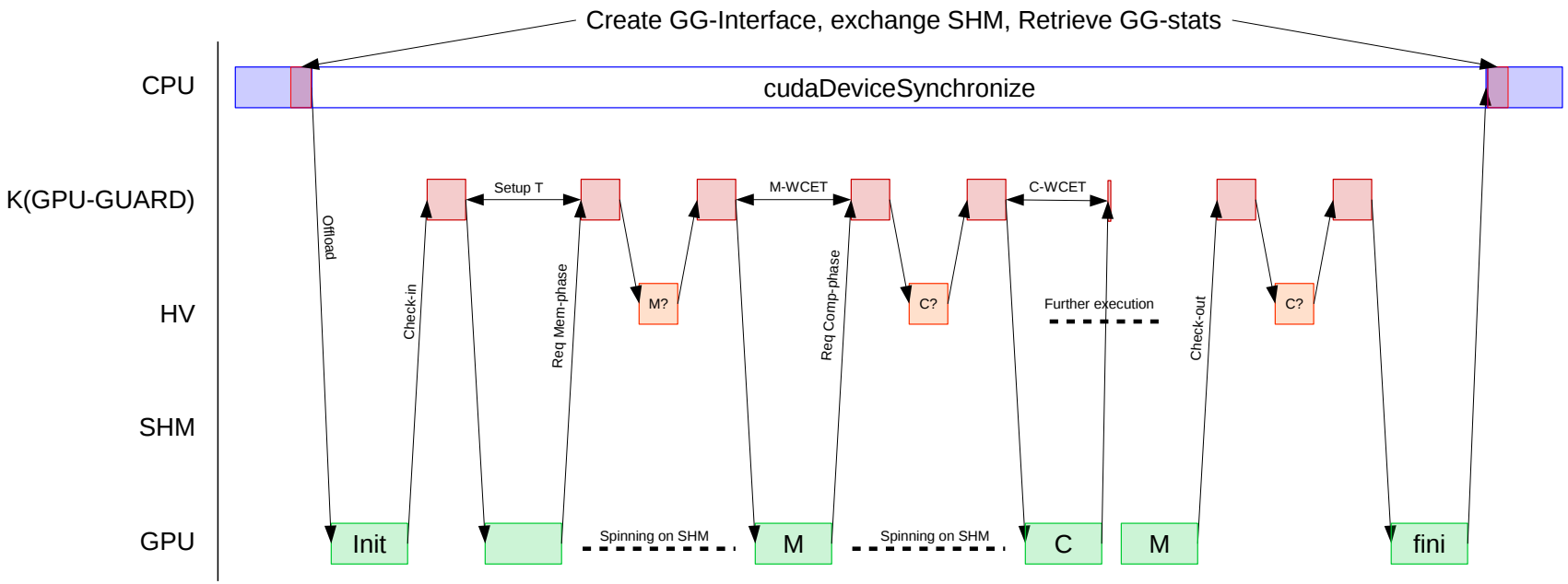


Problems with PREM on GPUs

- Memory bandwidth is almost always a bottleneck
- Compute-phases are shorter due to high parallelism
- Mutual exclusion for memory access kills performance
- Costly synchronization ($\approx 2\,\mu s$)
  - between CPU and GPU or
  - between multiple SMs in the GPU

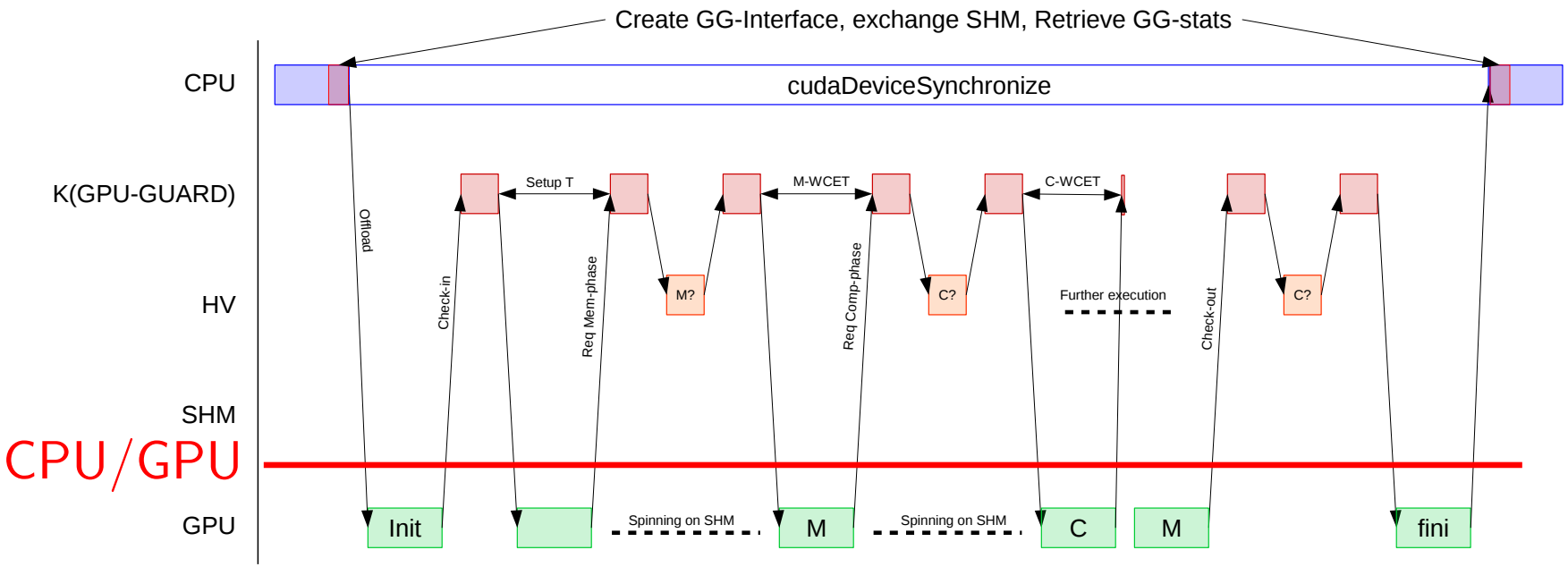


PREM on GPU: Early approach – GPUguard (ETHZ)



- Low performance due to excessive synchronization between CPU and GPU

PREM on GPU: Early approach – GPUguard (ETHZ)



- Low performance due to excessive synchronization between CPU and GPU

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Pros:

- Low synchronization overhead

Cons:

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Pros:

- Low synchronization overhead
- Applies not only to GPU but can span the whole chip

Cons:

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Pros:

- Low synchronization overhead
- Applies not only to GPU but can span the whole chip

Cons:

- Cannot handle dynamic workload

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Pros:

- Low synchronization overhead
- Applies not only to GPU but can span the whole chip

Cons:

- Cannot handle dynamic workload
- Over-provisioning due to uncertain execution time

Another approach: Time-Triggered scheduling

- GPU jobs are often offloaded in batches (e.g. one video frame)
  - the whole batch can be scheduled
  - all parameters are known at least at offload time
  - the processing pipeline is static (safety)

Pros:

- Low synchronization overhead
- Applies not only to GPU but can span the whole chip

Cons:

- Cannot handle dynamic workload
- Over-provisioning due to uncertain execution time
  - Reduced by our approach

Outline

Overview

Interference	Approach	When
CPU-CPU	PREM and TT scheduling	past

Overview

Interference	Approach	When
CPU-CPU	PREM and TT scheduling	past
GPU-GPU	"PREM" and TT scheduling	started in this paper

Overview

Interference	Approach	When
CPU-CPU	PREM and TT scheduling	past
GPU-GPU	"PREM" and TT scheduling	started in this paper
CPU-GPU	TT scheduling + ?	future

Overview

Interference	Approach	When
CPU-CPU	PREM and TT scheduling	past
GPU-GPU	"PREM" and TT scheduling	started in this paper
CPU-GPU	TT scheduling + ?	future

Experiments:

1. Synchronization overhead
2. Inter-kernel interference (2D convolution)
3. Detailed interference characterization (2D convolution)

Synchronization between GPU jobs/kernels

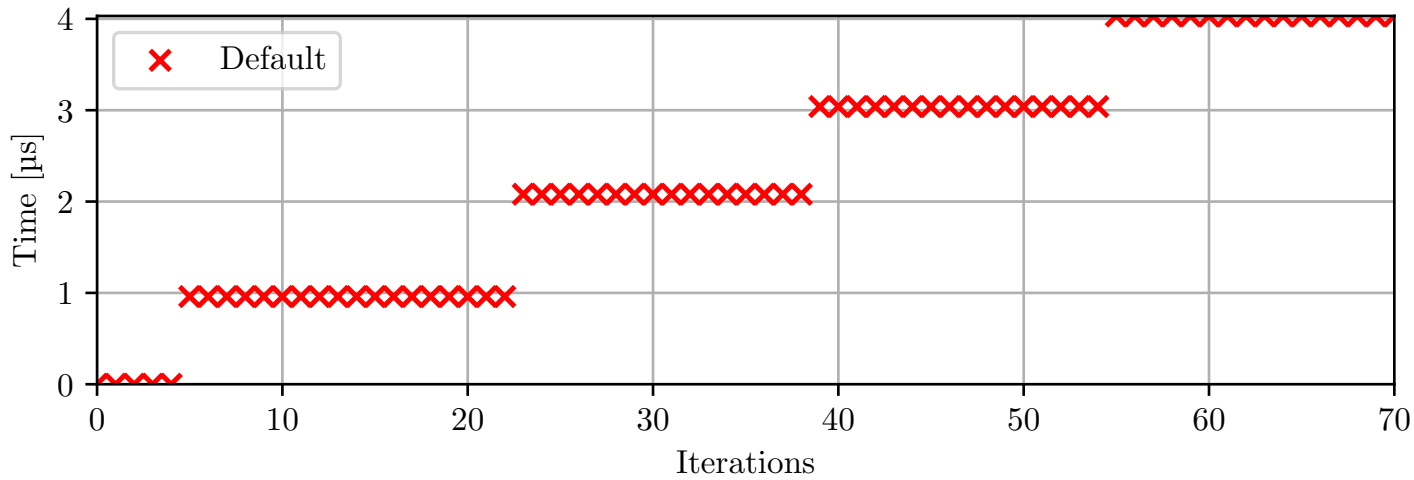
- Within one CUDA block (one SM of the GPU) – built-in

Synchronization between GPU jobs/kernels

- Within one CUDA block (one SM of the GPU) – built-in
- Across multiple CUDA blocks (SMs):
  - Spinlock-like in pinned (non-cached) memory: 2 µs

Synchronization between GPU jobs/kernels

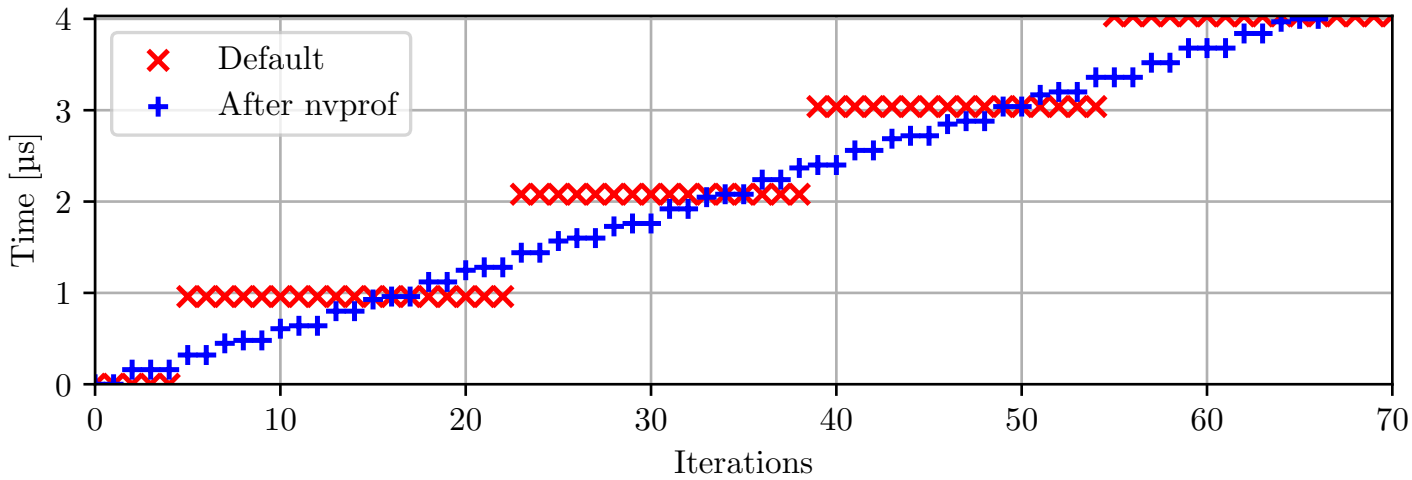
- Within one CUDA block (one SM of the GPU) – built-in
- Across multiple CUDA blocks (SMs):
  - Spinlock-like in pinned (non-cached) memory: 2 µs
  - Time-based (`globaltimer` register):



  - Default timer resolution is not sufficient: 1 µs

Synchronization between GPU jobs/kernels

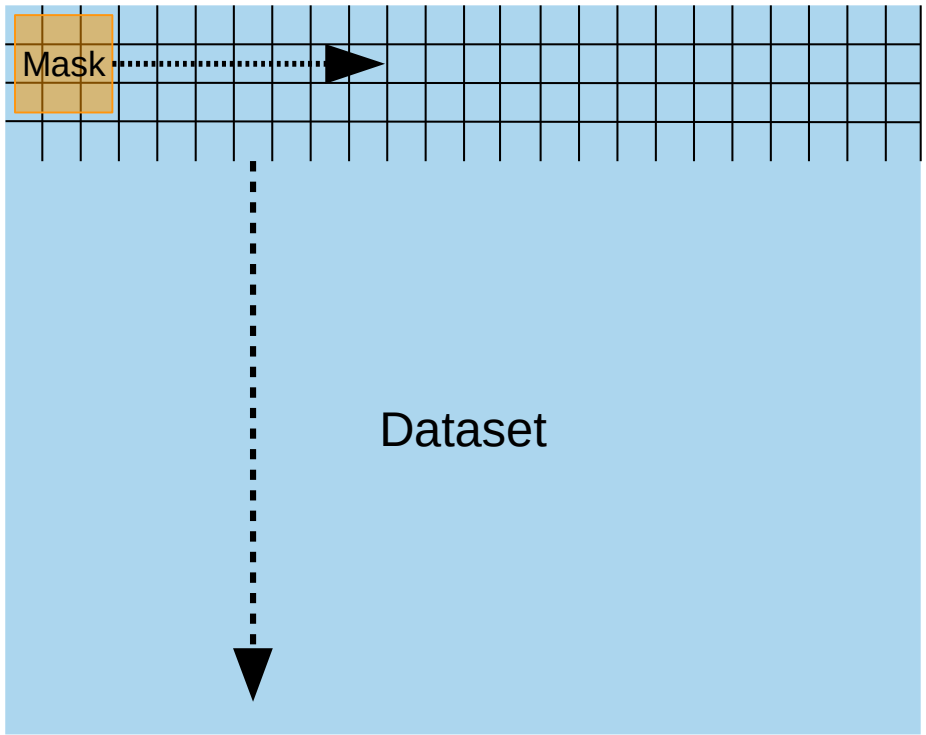
- Within one CUDA block (one SM of the GPU) – built-in
- Across multiple CUDA blocks (SMs):
  - Spinlock-like in pinned (non-cached) memory: 2 µs
  - Time-based (`globaltimer` register):



  - Default timer resolution is not sufficient: 1 µs
  - nvprof recofigures the resolution to about 160 ns

2D convolution benchmark

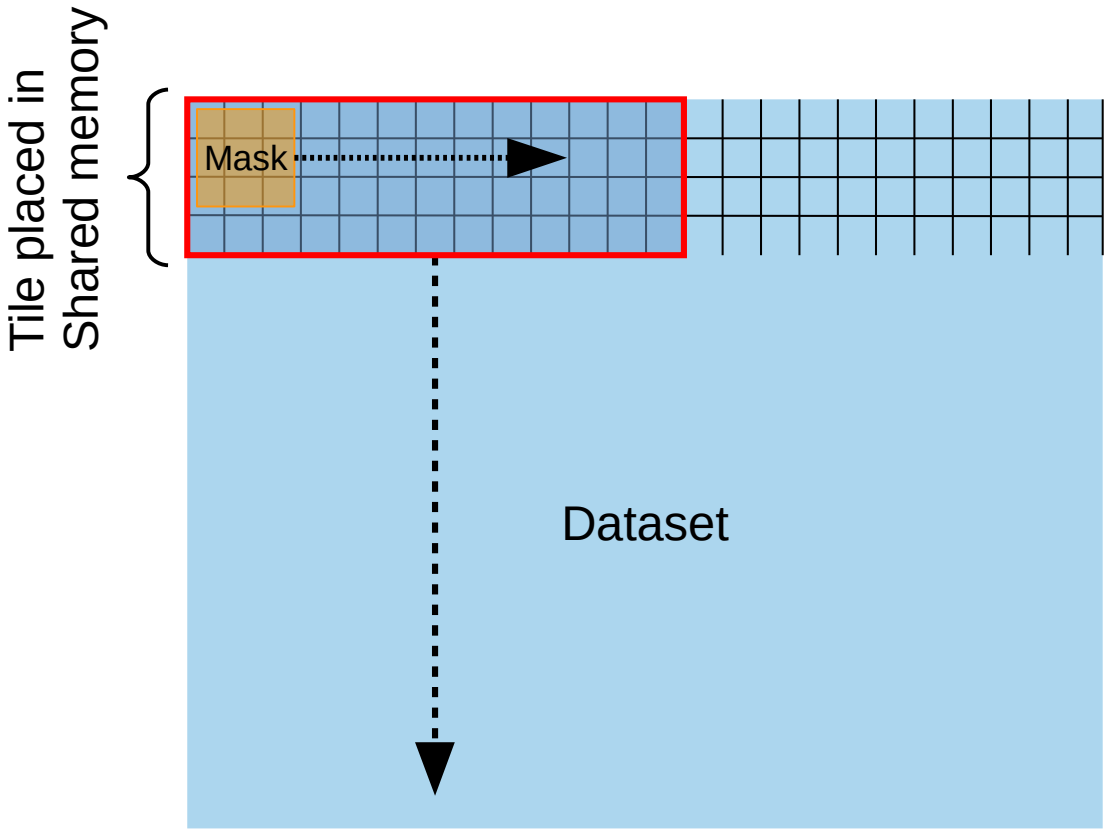
From Polybench-ACC benchmark suite



- Original (legacy) implementation

2D convolution benchmark

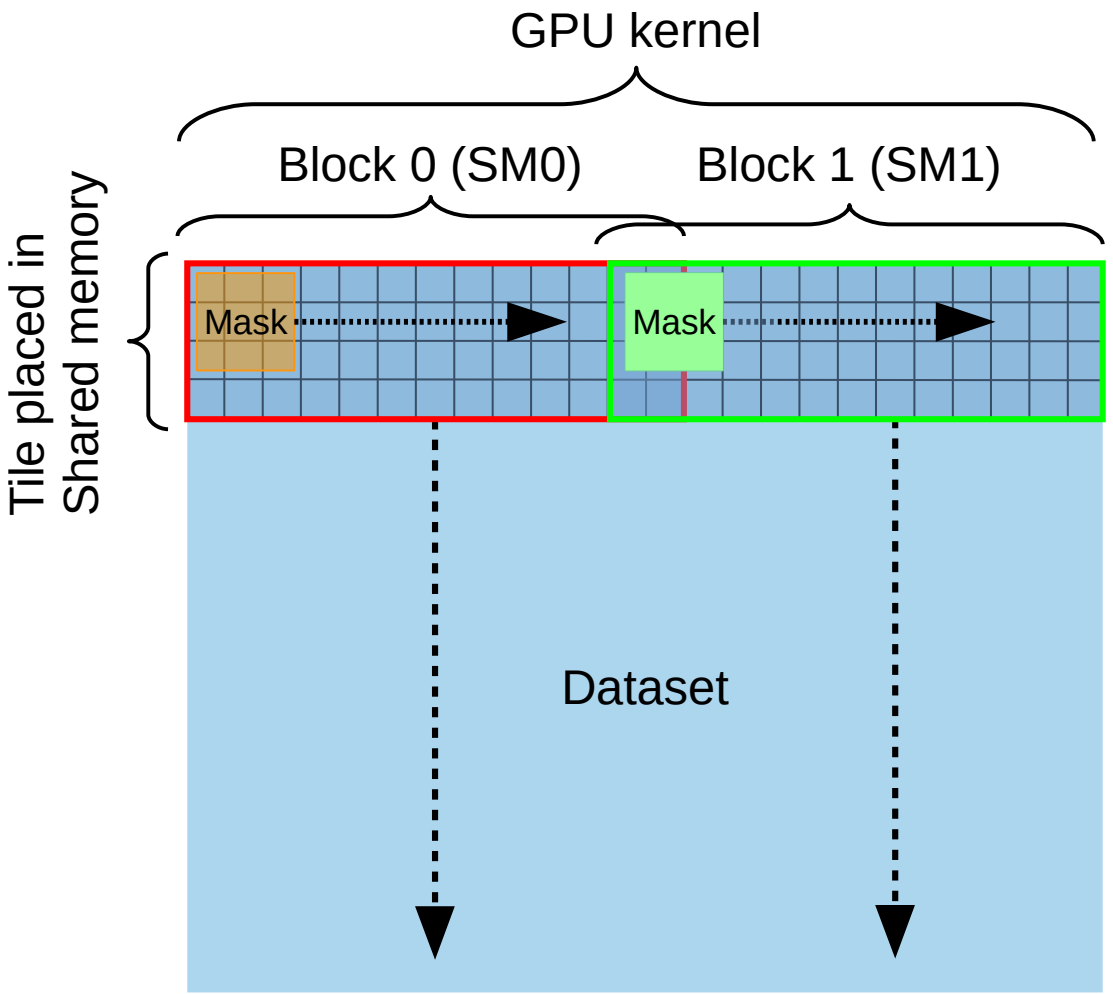
From Polybench-ACC benchmark suite



- Original (legacy) implementation
- Tiled implementation with prefetching
  - Tile size $\Rightarrow$ 4 kernels can run simultaneously

2D convolution benchmark

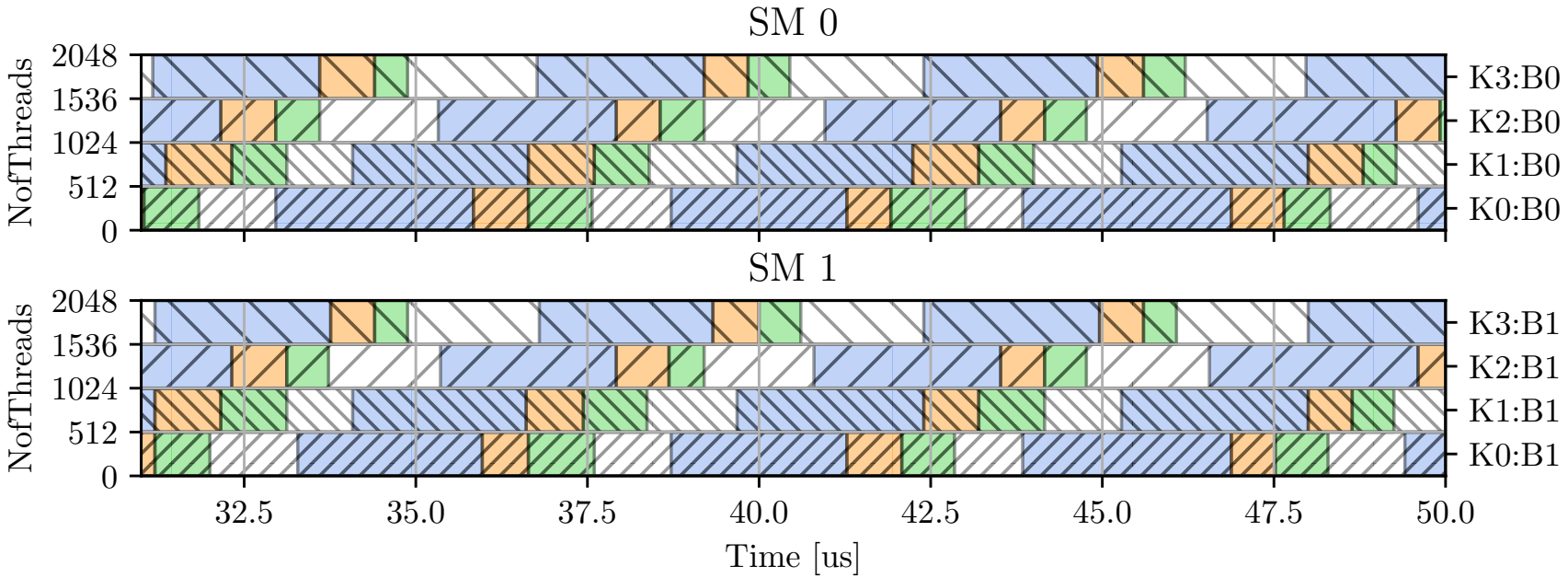
From Polybench-ACC benchmark suite



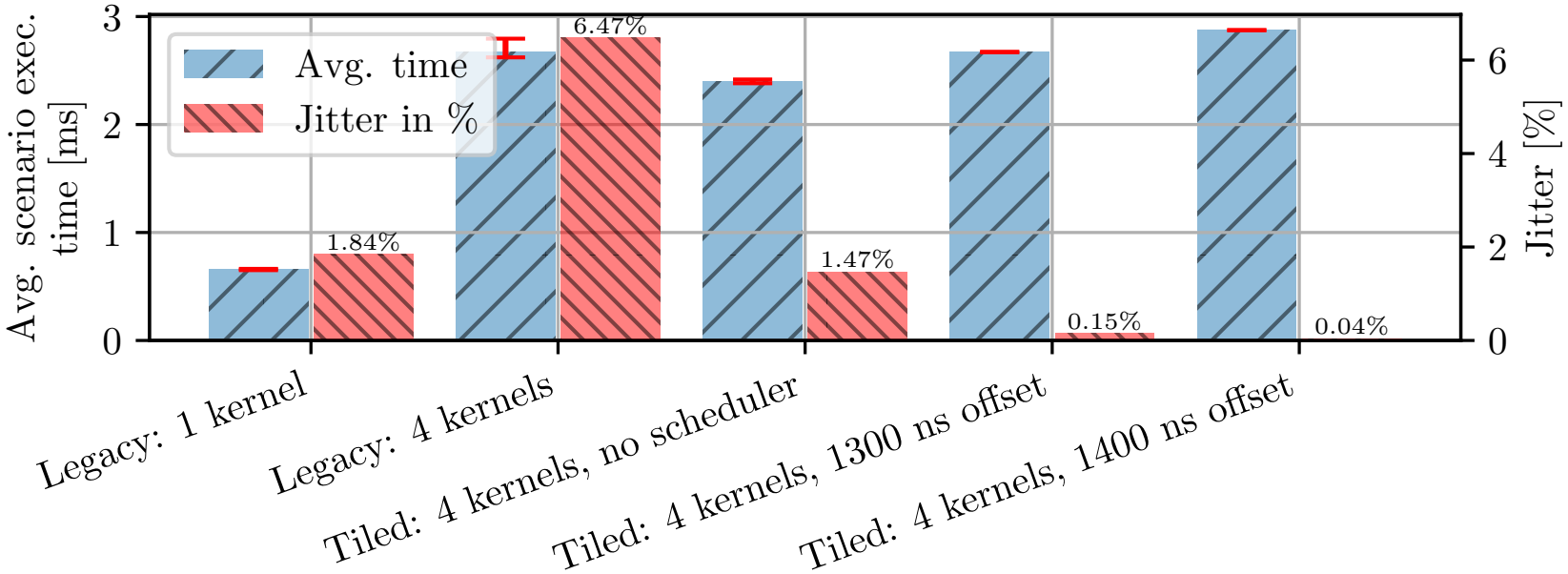
- Original (legacy) implementation
- Tiled implementation with prefetching
  - Tile size $\Rightarrow$ 4 kernels can run simultaneously

Tiled 2D convolution schedule

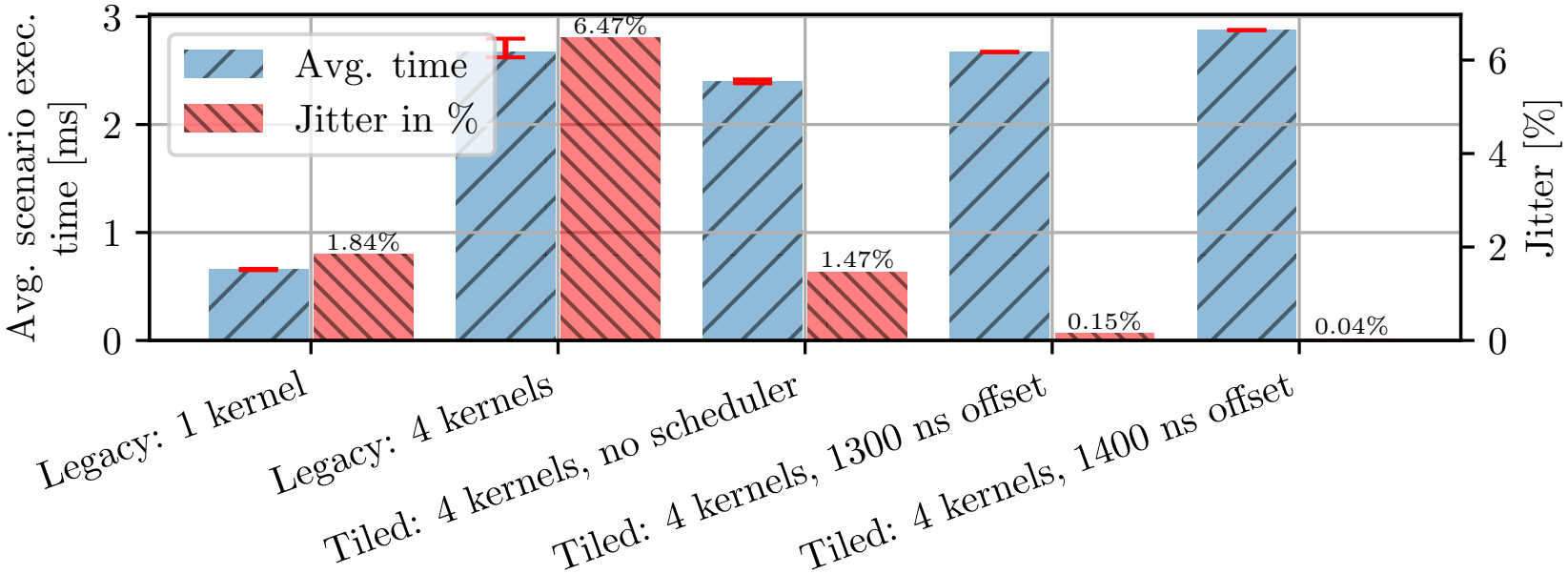
- 4 kernels, 2 streaming multiprocessors
- prefetch, compute, writeback phases + spinning
- different kernels started with different offsets

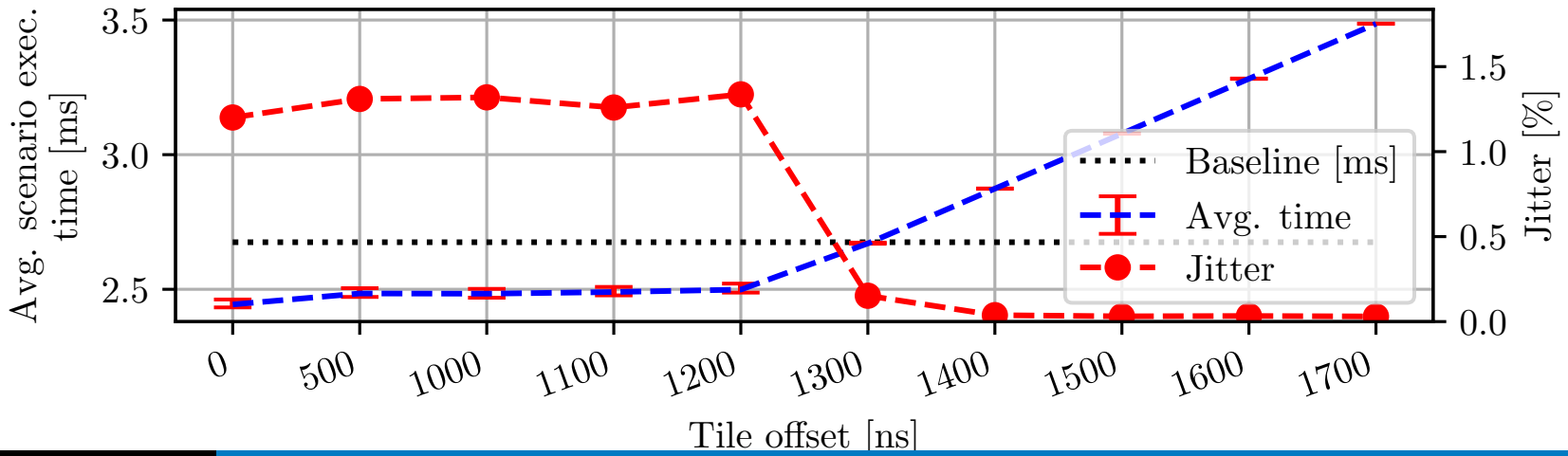


Results: Execution + jitter

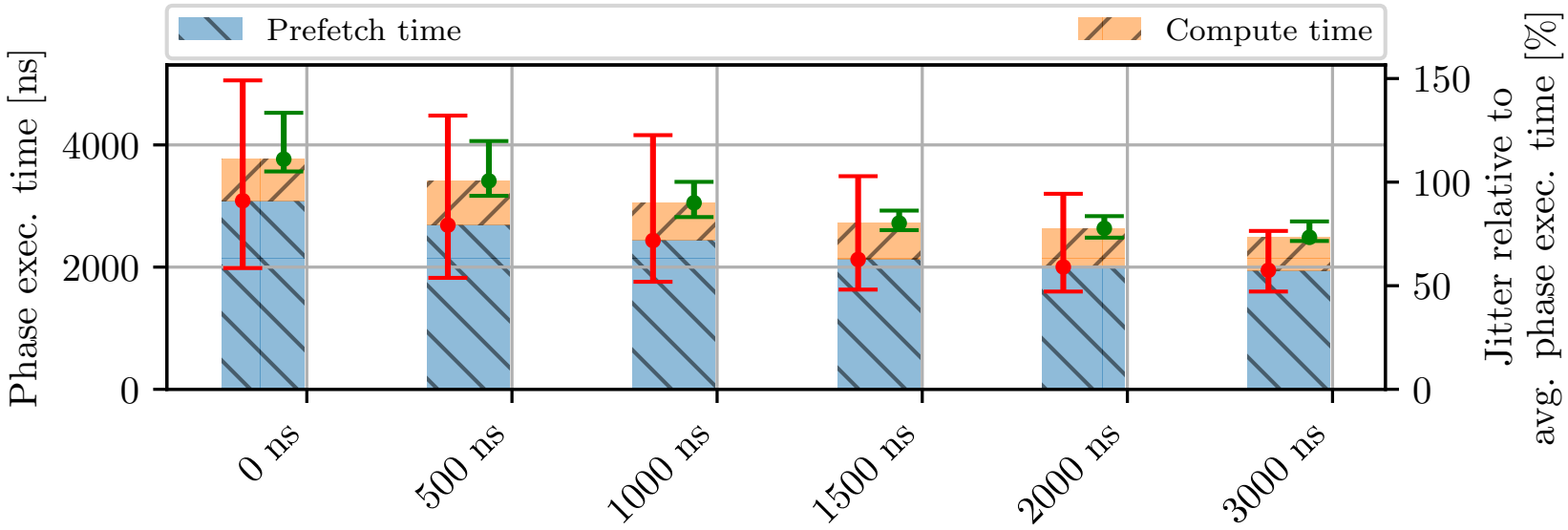


Results: Execution + jitter

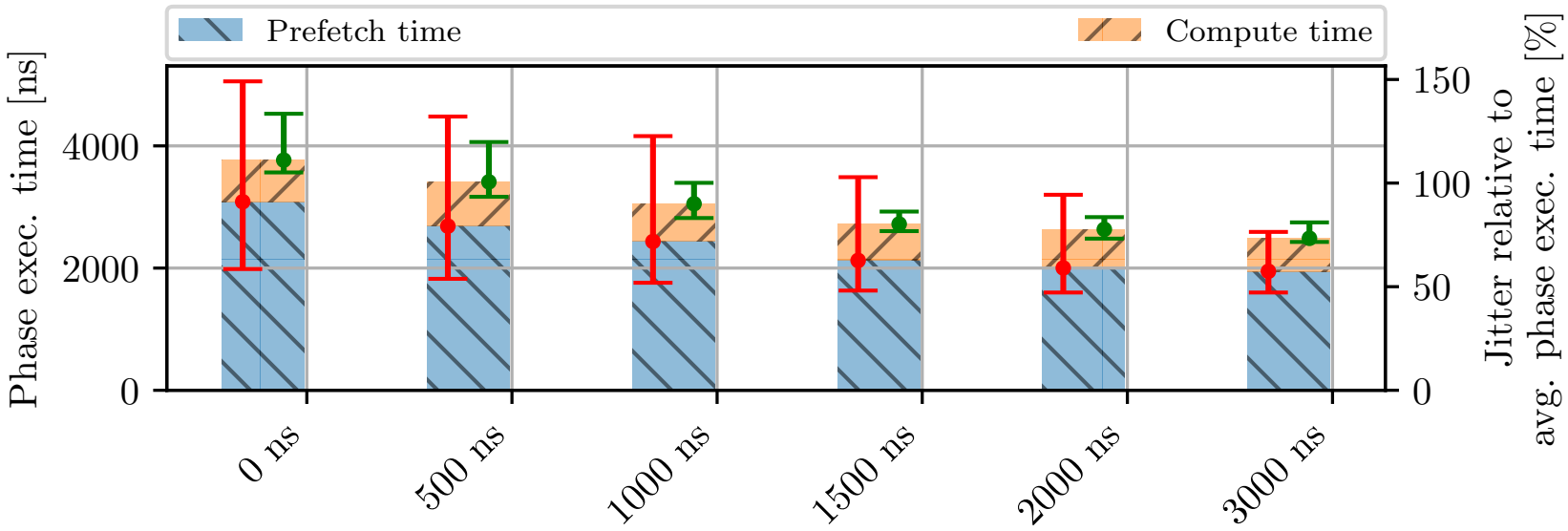




Interference between prefetch and compute phases

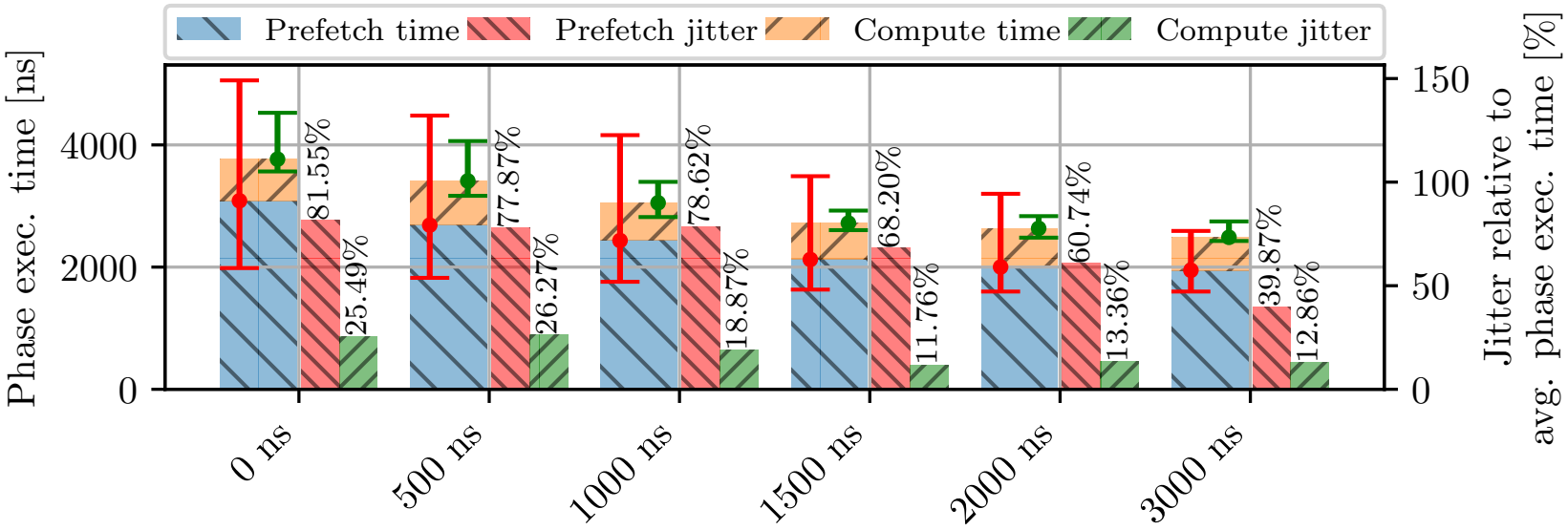


Interference between prefetch and compute phases



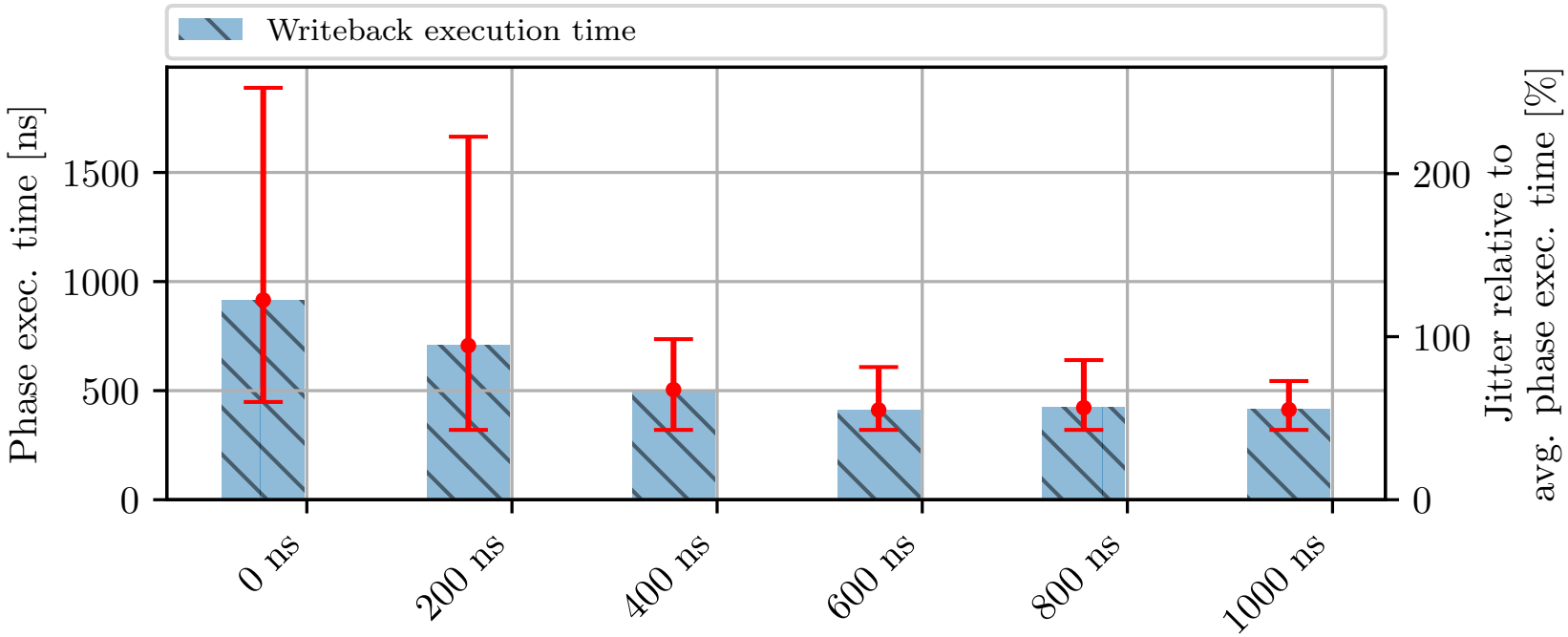
- Less overlap of prefetch phases ⇒ shorter execution time and smaller jitter
- Compute phases interfere with each other (shared memory bank conflicts)
  ⇒ prevents straighformward application of PREM

Interference between prefetch and compute phases

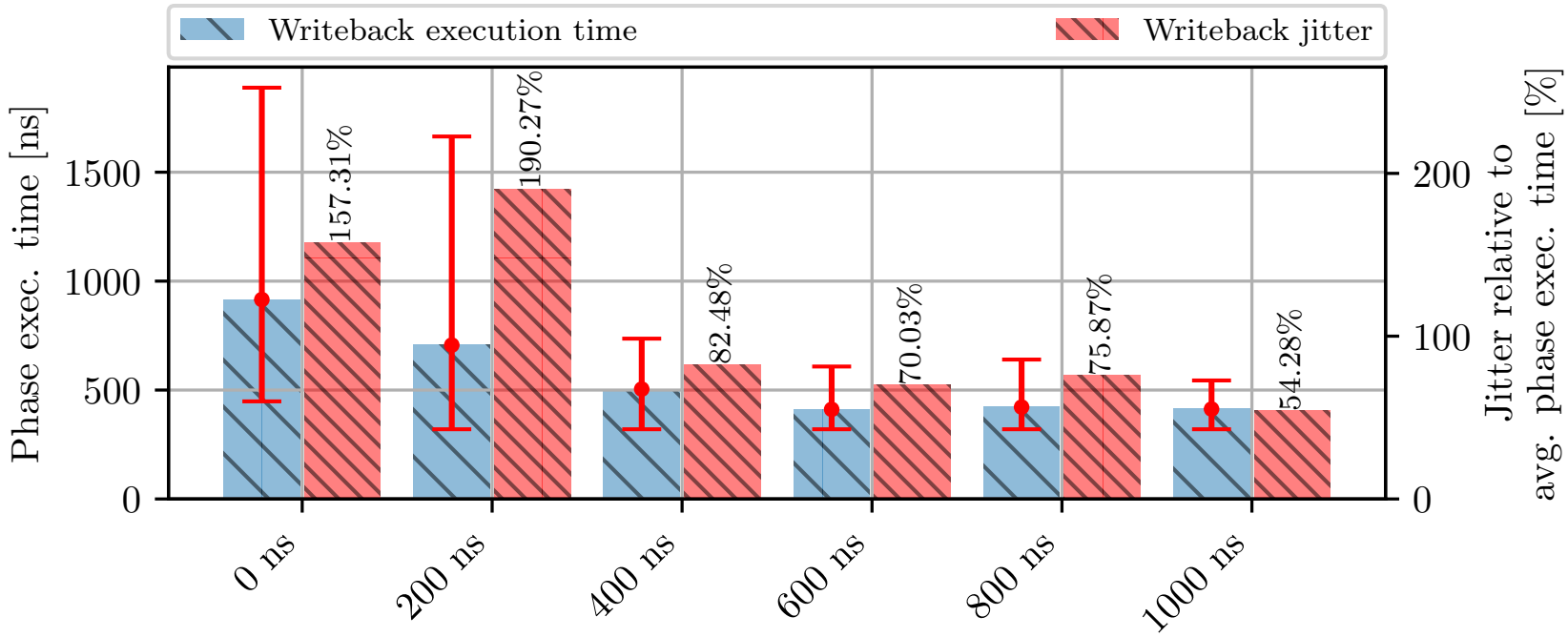


- Less overlap of prefetch phases ⇒ shorter execution time and smaller jitter
- Compute phases interfere with each other (shared memory bank conflicts)
  ⇒ prevents straighformward application of PREM

Interference between writeback phases



Interference between writeback phases



- Less overlap of writeback phases $\approx\Rightarrow$ shorter execution time and smaller jitter

Conclusion

- Time-triggered scheduling on TX2 GPU is possible

Conclusion

- Time-triggered scheduling on TX2 GPU is possible
- GPU `globaltimer` register has sufficient resolution (160 ns) after running `nvprof`

Conclusion

- Time-triggered scheduling on TX2 GPU is possible
- GPU `globaltimer` register has sufficient resolution (160 ns) after running `nvprof`
- Even very simple scheduling (adding offset) shows potential to reduce execution time jitter

Future work: Interference-aware scheduling of complex GPU workloads