Understanding Memory Interference in CPU-GPU Embedded Systems

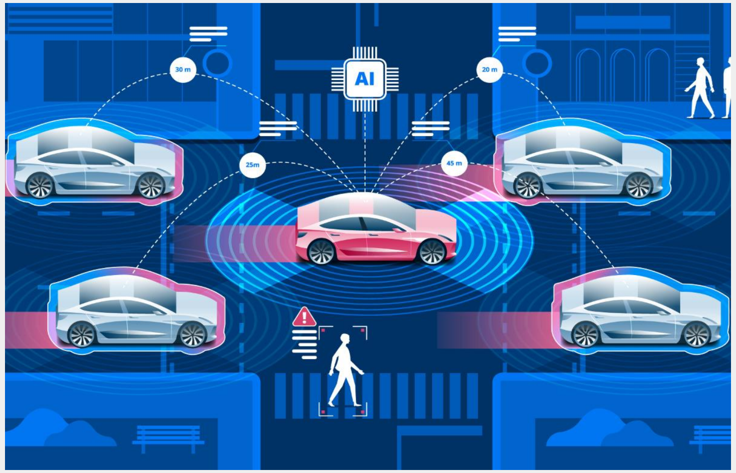
Alessio Masola – alessiomasola@unimore.it

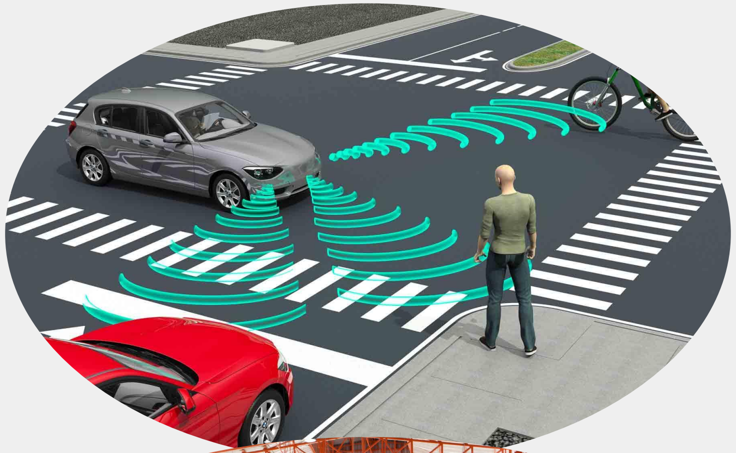
Nicola Capodieci – nicola.capodieci@unimore.it

HiPeRT Lab
High-Performance Real-Time Lab

Brief Background Introduction

- Many tasks to perform in critical environments





Goal

To **understand the interference effect on shared memory resources between CPU and GPU in integrated** chips

- **Latency deterioration of GPU** kernels that run concurrently with CPU memory intensive applications.
  - Which GPU kernels' performance metrics can predict memory sensitivity?
  - CPU intensive jobs can delay **CPU to GPU command submission**?
- How can we deal with the interference and find a way to exploit it?

Goal

- Understanding Memory Interference in CPU-GPU embedded systems
- Interference to the CPU submitter

Reference Hardware Architecture: ***Jetson Xavier***

Find a way to predict the interference introduced by contention on the level of shared memory (LPDDR4 - SDRAM) between GPU and CPU?

<u>Shared Memory Level</u>

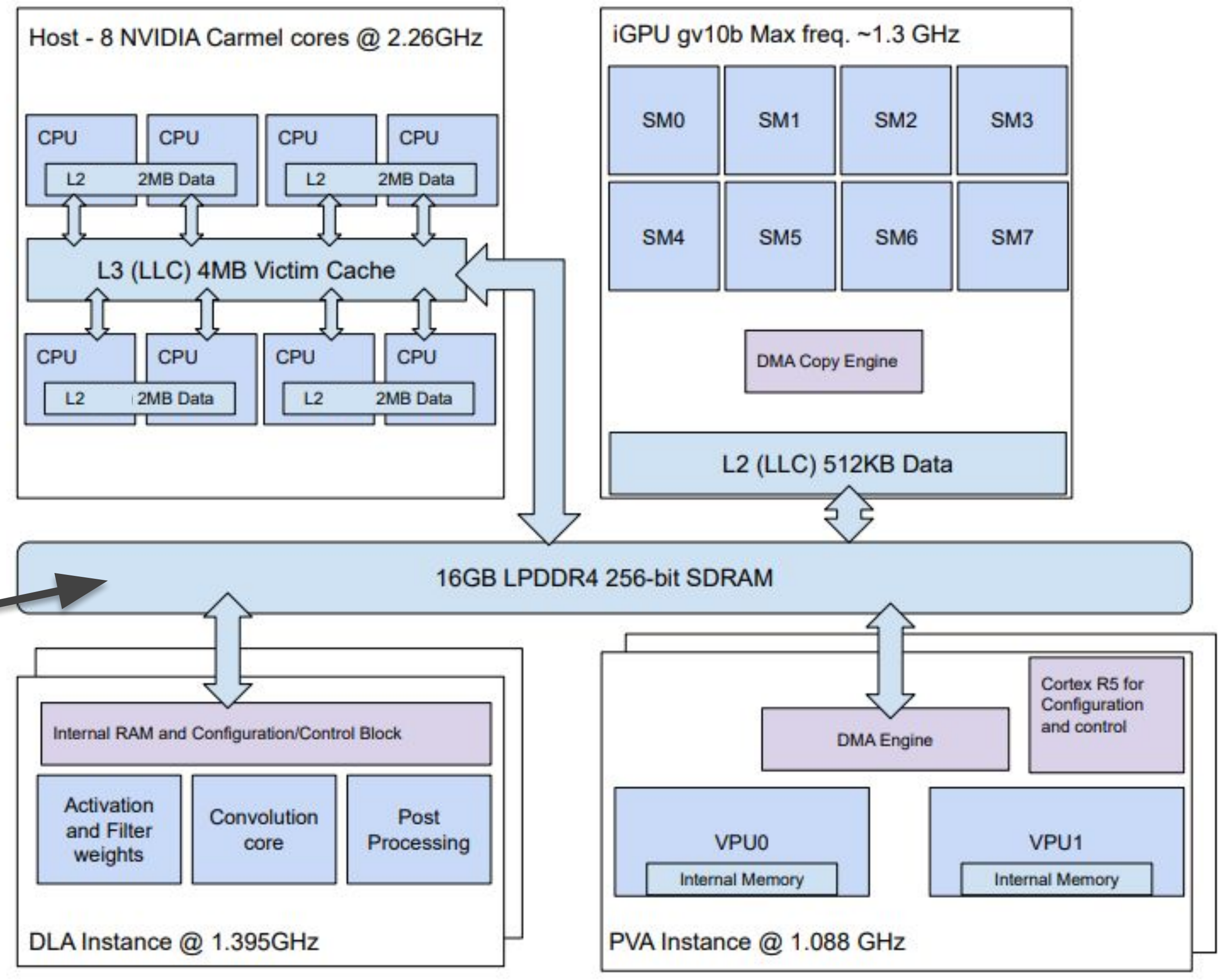
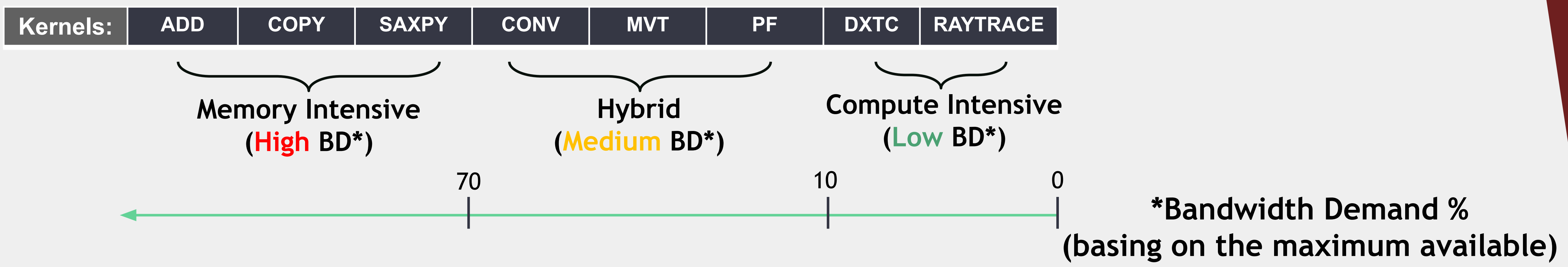


Fig. 2: Compute engines and related memory contention points in the NVIDIA Jetson Xavier.

Tested Kernels

NVIDIA profiling tools (NVPROF) allows us to understand and optimize the performance of CUDA, OpenACC or OpenMP applications.



KERNEL NAME	Data Size In (MiB)	Data Size Out (MiB)	Taken From
ADD	40*2	40	Synthetic / In house implementation
SAXPY	40*2	40	Synthetic / In house implementation
COPY	40	40	Synthetic / In house implementation
RAYTRACE	-	1.5234	In House Implementation
DXTC	0.003 + 0.5	0.25	Cuda Samples
CONV	0.00001 + 40	40	In House Implementation
MVT	64 + 0.01*4	0.01*2	Polybench
PF	0.5 + 255.5	0.5	Rodinia

NVPROF Collected Metrics

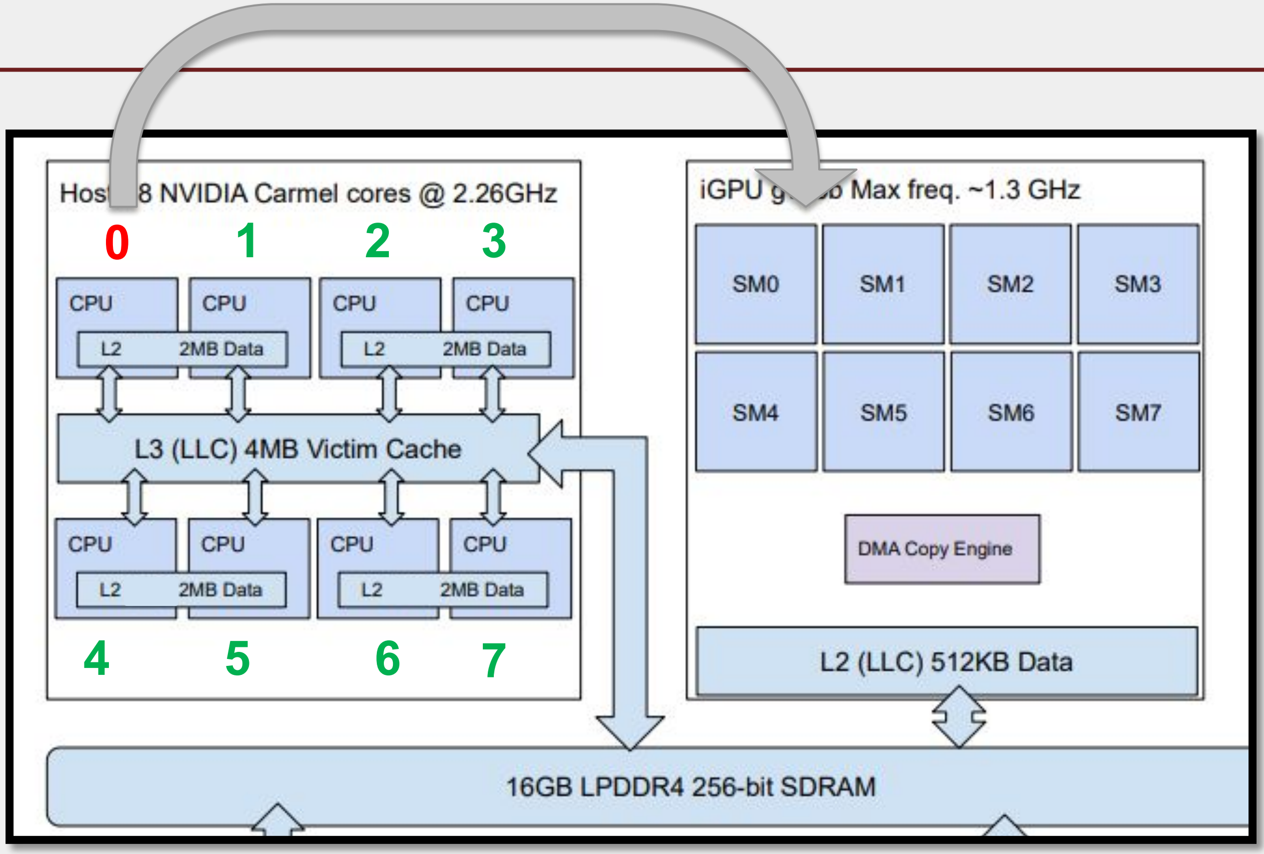
Metrics Name
L2 Throughput (Reads)
L2 Throughput (Writes)
L2 Cache Hit Rate
L2 Cache Utilization
System Memory Read Throughput
Global Memory Load Efficiency
Global Memory Store Efficiency
Instructions Executed
Instructions Issued
Issue Stall Reasons (Instructions Fetch)
Issue Stall Reasons (Execution Dependency)
Issue Stall Reasons (Data Request)
Issue Stall Reasons (Other)
Issue Stall Reasons (Memory Throttle)
Executed IPC
Issued IPC
Multiprocessor Activity
Eligible Warps Per Active Cycle
Achieved Occupancy
Load/Store Function Unit Utilization
Warp level instructions for global loads
System Memory Read Bytes
System Memory Write Bytes
Timing kernels Execution

Metrics about:

- Memory behaviour
- Compute behaviour
- Kernel completion lantencies

Available architecture's metrics: *nvprof --query-events*
Maximum board's performance: *sudo nvpmodel -m 0*

Test Scenario



CPU ID	Work
0	NVPROF of GPU's Work
1 .. 7	Thread with Interference (memset/hesoc-mark)

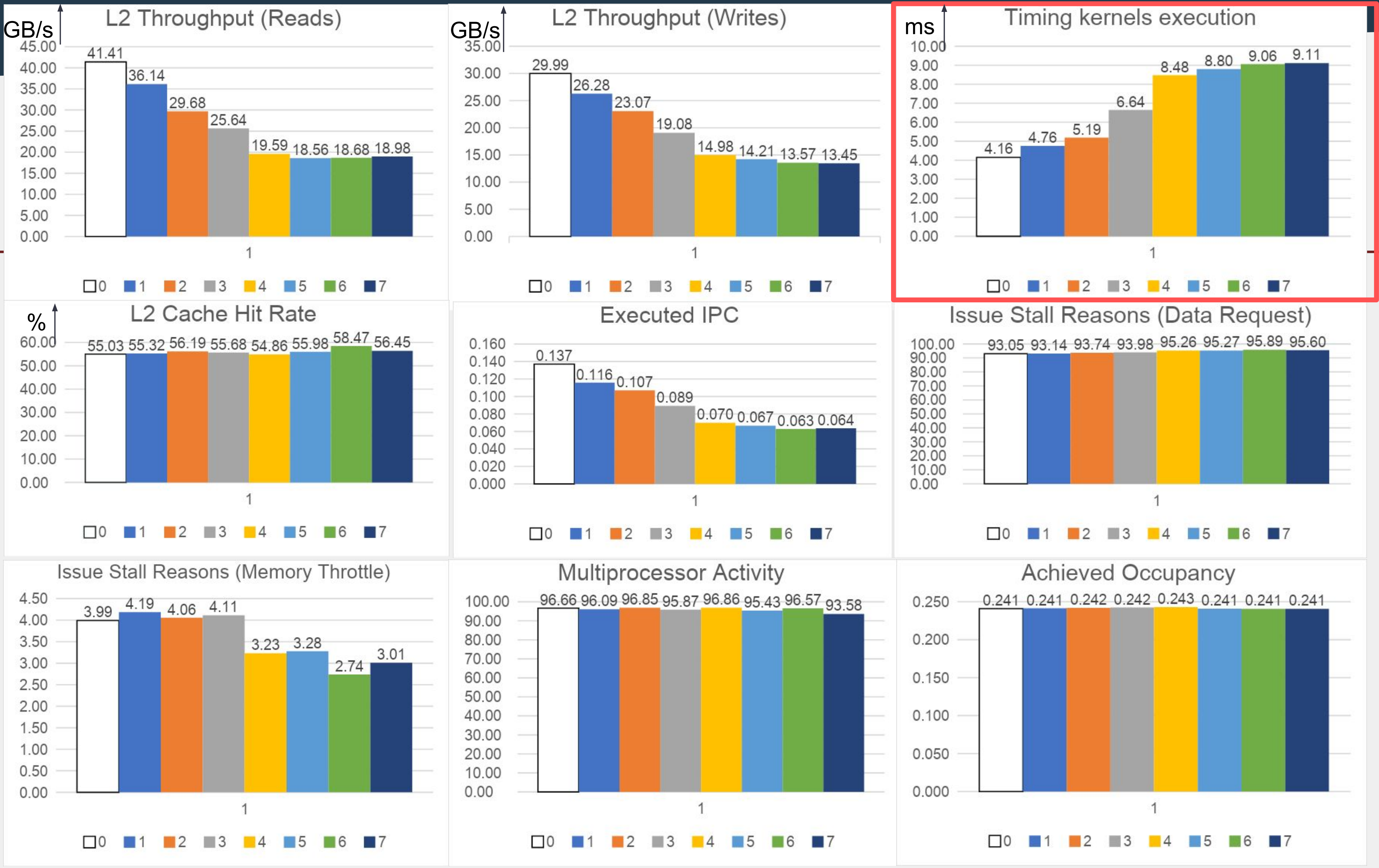
The interference was performed with a script that launches different **Meminterf of 50 mb** with an high amount of iterations, in order to saturate the L2 and L3 CPU's caches to create traffic on the SDRAM that is shared with the GPU.

Memory Interference reference: https://git.hipert.unimore.it/mem-prof/hesoc-mark

What We Observed

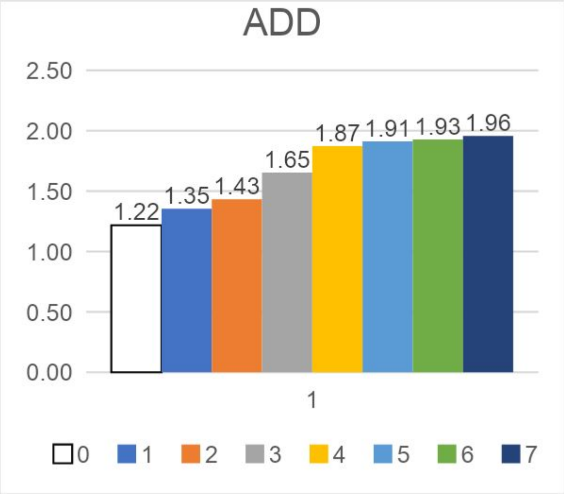
- Lots of data!
- **How** those metrics **change** as a **function of the magnitude of the memory interference.**
- We are **interested in correlating** such variations to the magnitude of interference in order to understand which metrics predict memory sensitivity towards a CPU aggressor.

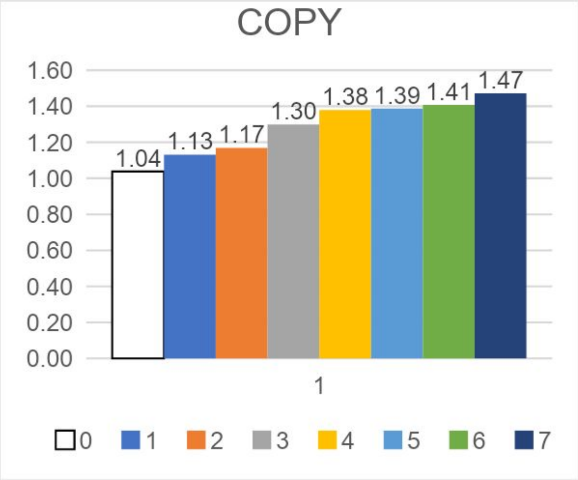
MVT Kernel

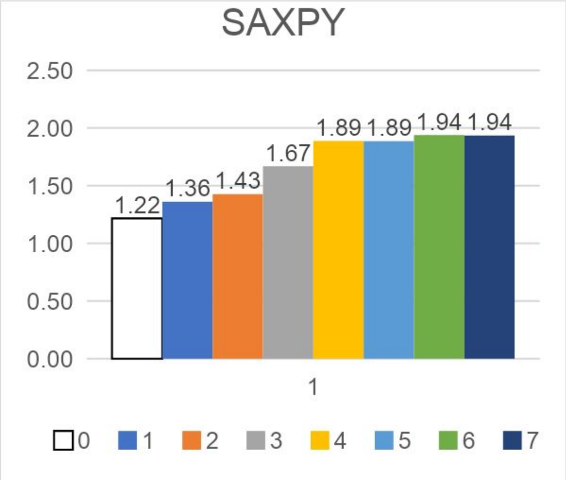

L2 Throughput (Reads)
GB/s
41.41 36.14 29.68 25.64 19.59 18.56 18.68 18.98
L2 Throughput (Writes)
GB/s
29.99 26.28 23.07 19.08 14.98 14.21 13.57 13.45
Timing kernels execution
ms
4.16 4.76 5.19 6.64 8.48 8.80 9.06 9.11
L2 Cache Hit Rate
%
55.03 55.32 56.19 55.68 54.86 55.98 58.47 56.45
Executed IPC
0.137 0.116 0.107 0.089 0.070 0.067 0.063 0.064
Issue Stall Reasons (Data Request)
93.05 93.14 93.74 93.98 95.26 95.27 95.89 95.60
Issue Stall Reasons (Memory Throttle)
3.99 4.19 4.06 4.11 3.23 3.28 2.74 3.01
Multiprocessor Activity
96.66 96.09 96.85 95.87 96.86 95.43 96.57 93.58
Achieved Occupancy
0.241 0.241 0.242 0.242 0.243 0.241 0.241 0.241
0 1 2 3 4 5 6 7

Kernels Execution Latencies

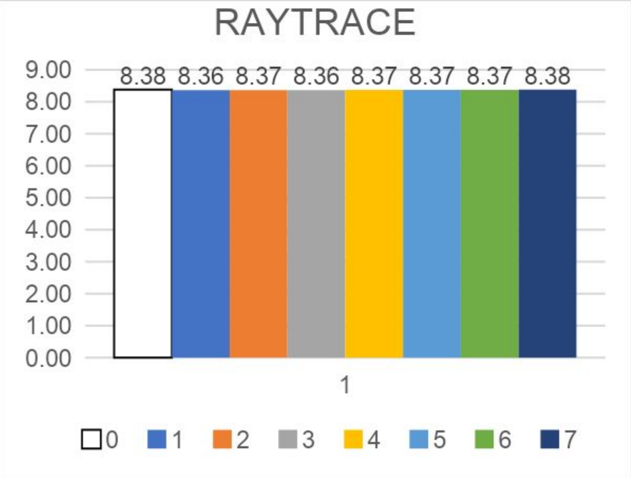
Memory Intensive

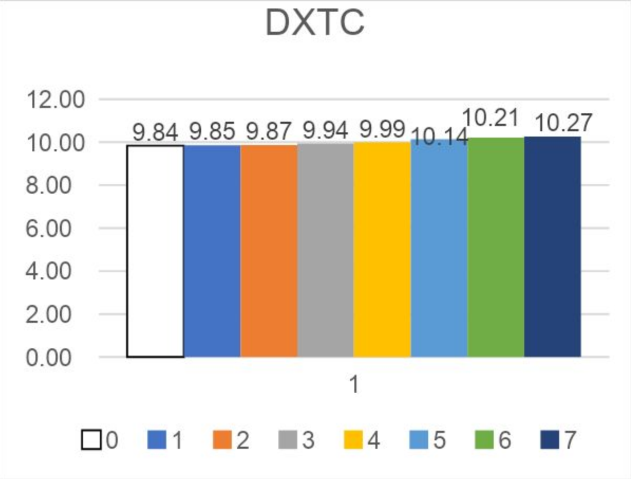


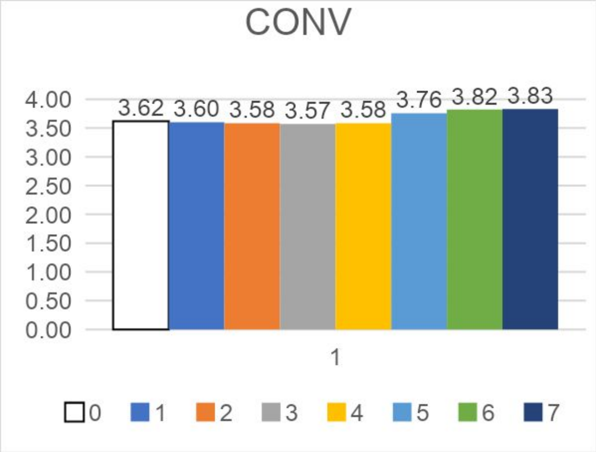


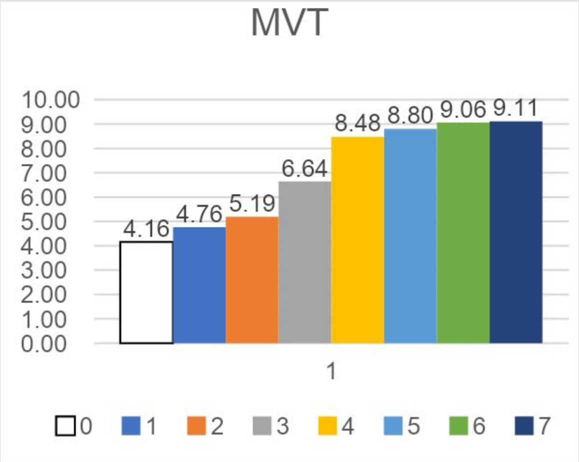


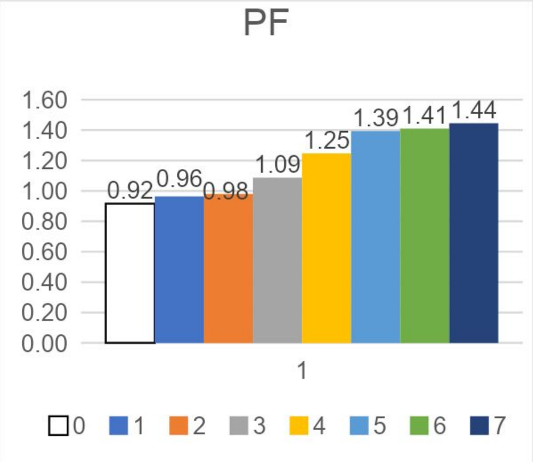
Compute Intensive











Hybrid

ms

Interference threads

Correlation Factor

Which metrics **most influence a slowdown**? -> correlation matrix

Pearson Correlation:

The correlation factor [-1,1]

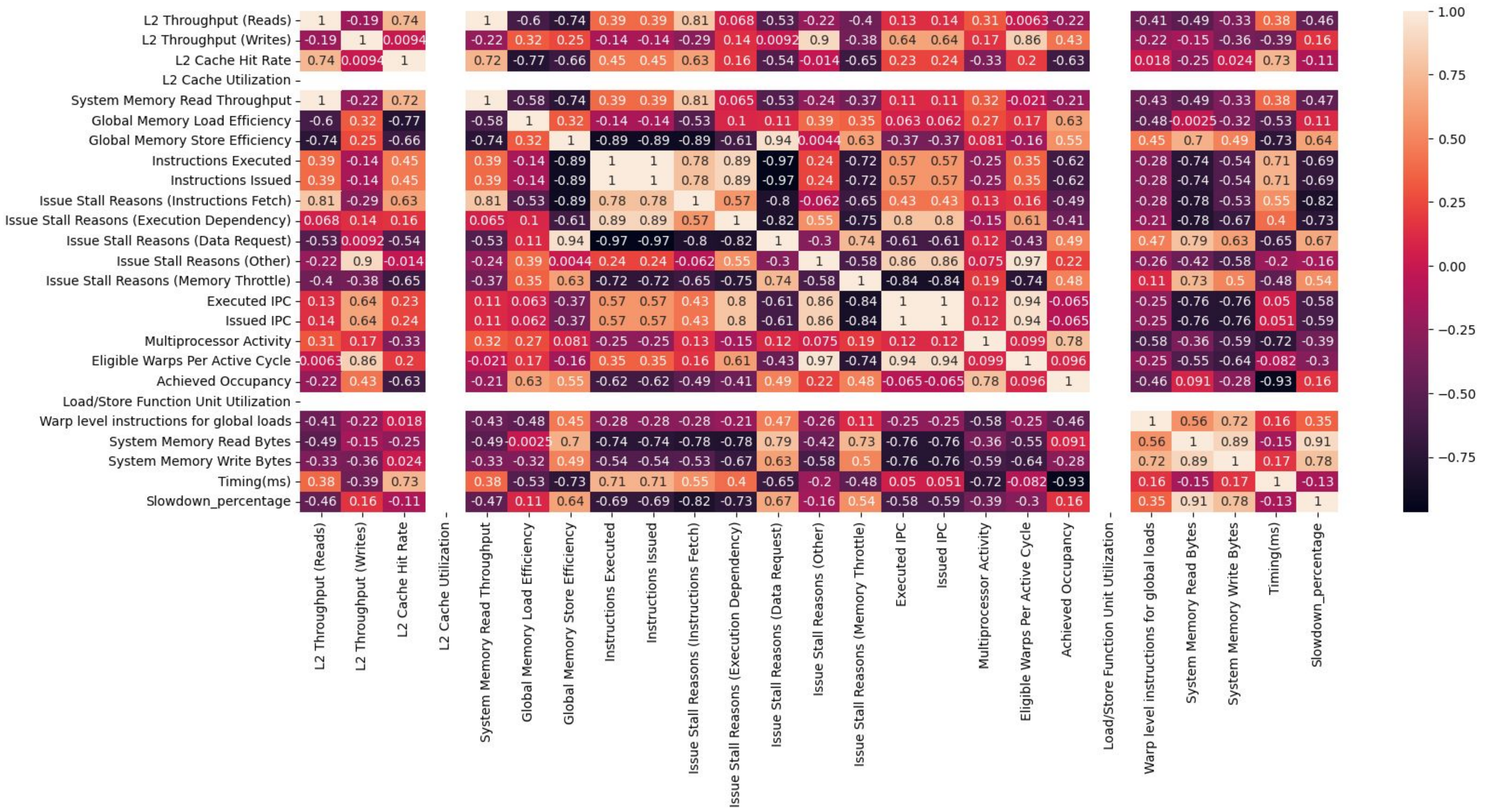
- **0** : **no correlation**
- Near **1** : **highly and strong linear correlation**
- Near **-1** : **reverse correlation**

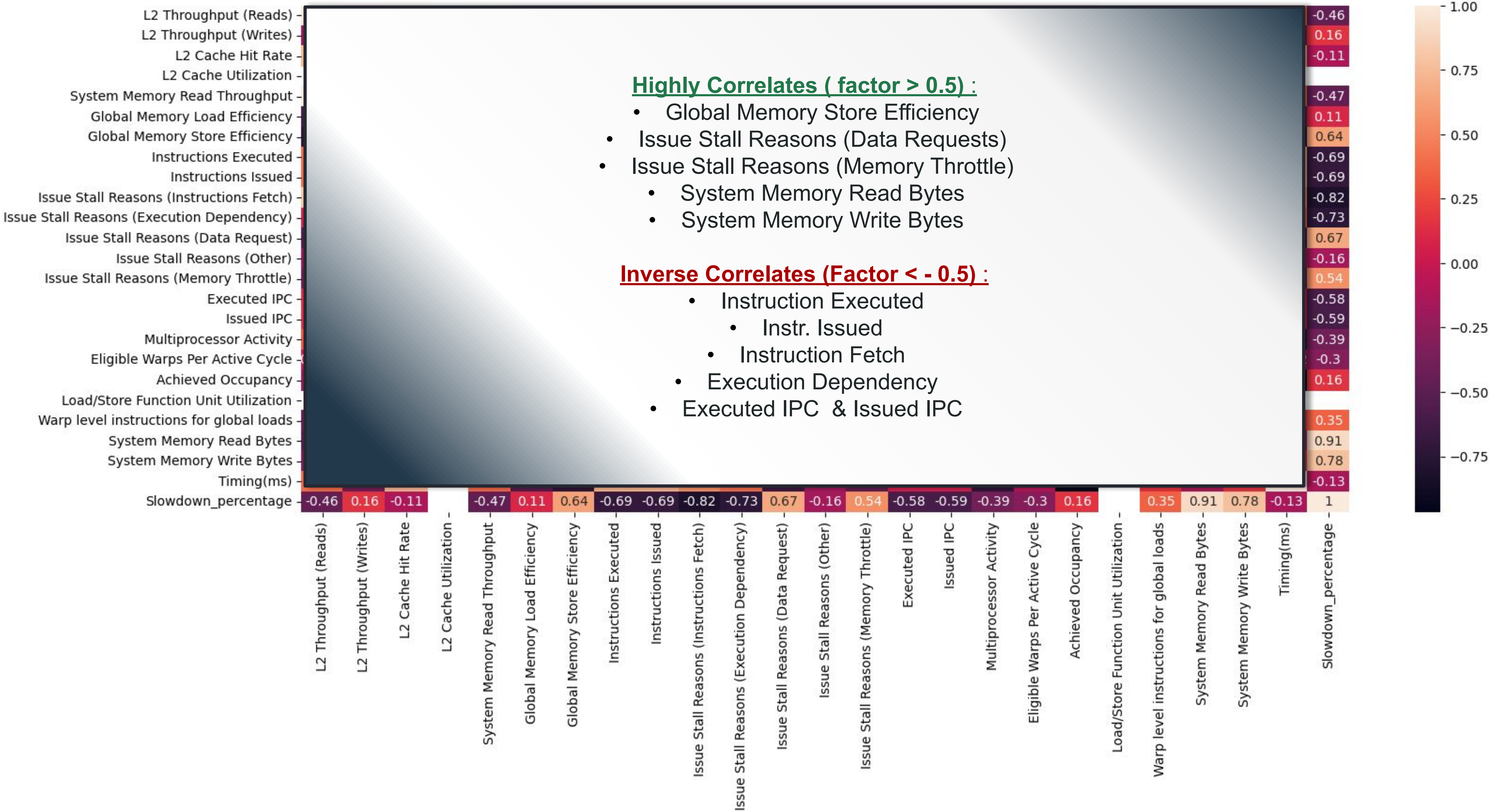
Data used for a **slowdown correlation**: *all kernels' metrics when 7 interfering* threads were running.

Slowdown factor: $\frac{T_{with\ 7}}{T_{Baseline}}$

$T_{with\ 7}$ → *Execution timing with 7 interferents*

$T_{Baseline}$ → *Baseline execution timing*


L2 Throughput (Reads)
L2 Throughput (Writes)
L2 Cache Hit Rate
L2 Cache Utilization
System Memory Read Throughput
Global Memory Load Efficiency
Global Memory Store Efficiency
Instructions Executed
Instructions Issued
Issue Stall Reasons (Instructions Fetch)
Issue Stall Reasons (Execution Dependency)
Issue Stall Reasons (Data Request)
Issue Stall Reasons (Other)
Issue Stall Reasons (Memory Throttle)
Executed IPC
Issued IPC
Multiprocessor Activity
Eligible Warps Per Active Cycle
Achieved Occupancy
Load/Store Function Unit Utilization
Warp level instructions for global loads
System Memory Read Bytes
System Memory Write Bytes
Timing(ms)
Slowdown_percentage
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



Highly Correlates (factor > 0.5) :

- Global Memory Store Efficiency
- Issue Stall Reasons (Data Requests)
- Issue Stall Reasons (Memory Throttle)
- System Memory Read Bytes
- System Memory Write Bytes

Inverse Correlates (Factor < - 0.5) :

- Instruction Executed
- Instr. Issued
- Instruction Fetch
- Execution Dependency
- Executed IPC & Issued IPC

Conclusion and What's Next...

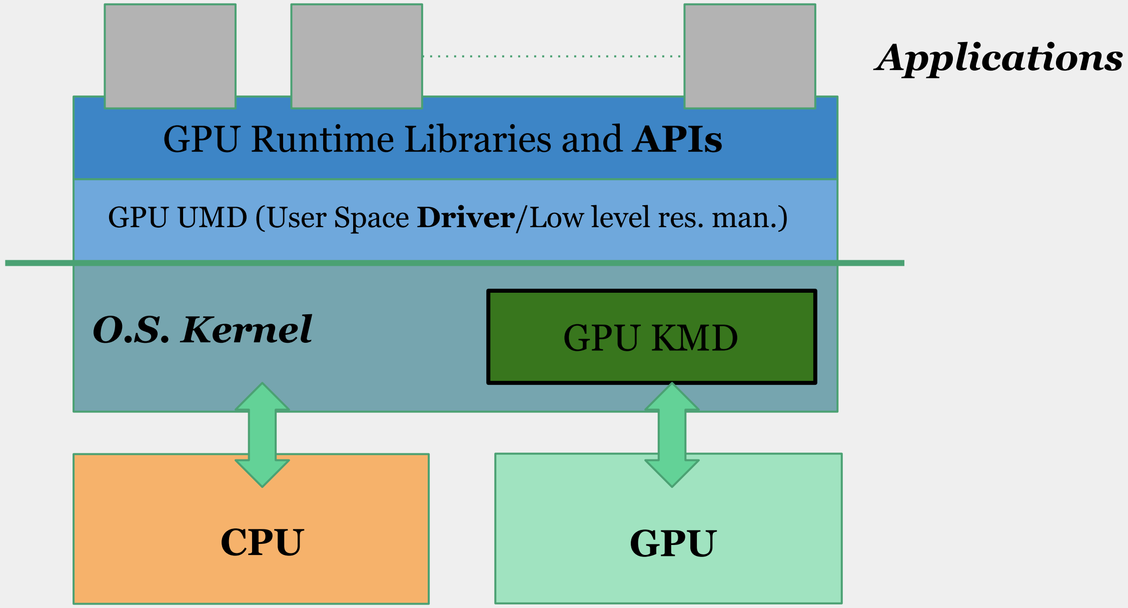
- Understanding **memory interference** from CPU to GPU is not trivial
- We shed some light on how **the kernel characteristics** might be used as **predictors** to kernel completion latencies
- We can use those correlation factors to propose a model able **to predict memory interference**
- Such a model can be used to enrich current **real-time task models** in order to put the basis for memory aware scheduling.

Interference to the CPU Submitter

High-Performance Real-Time Lab

What is a Submitter?

- Each call to the GPU API is made by the CPU.
- Each call traverses all the software and hardware depicted in this picture.
- Ideally: we want that the time taken by a CPU call to reach the GPU HW must be short and predictable.
- In reality: many kernel calls -> so much CPU overhead, **and memory interference might still be a factor.**

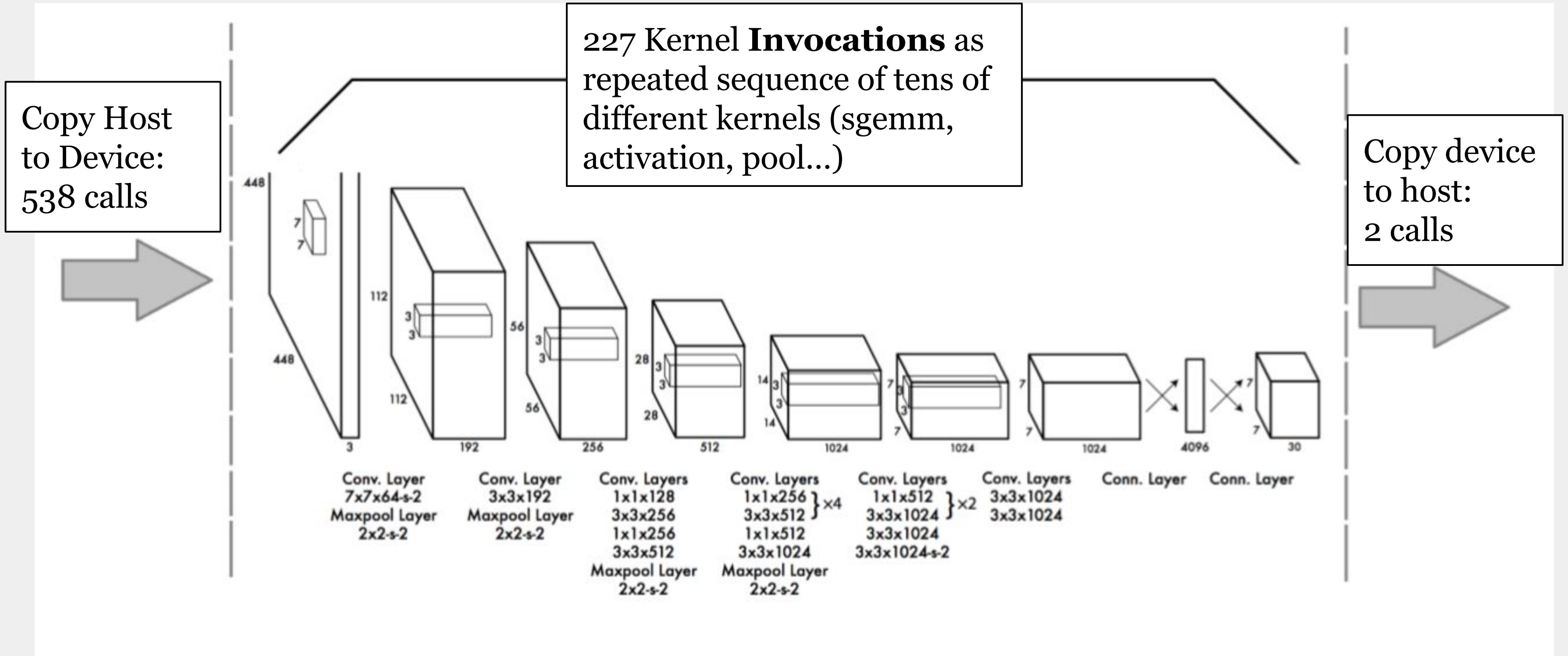


*Cavicchioli, Roberto, et al. "**Novel methodologies for predictable CPU-to-GPU command offloading.**" 31st Euromicro Conference on Real-Time Systems (ECRTS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.*

Why is it Important?

- Kernels are constantly submitted from the CPU to the GPU.
- A single kernel usually **does not** represent a schedulable task (too fined grained!)
  - (see real world example in the next slide)
- Each kernel submission is a CPU task that has to be scheduled and might suffer from interference

Neural Network Workloads on GPU [YoloV3]



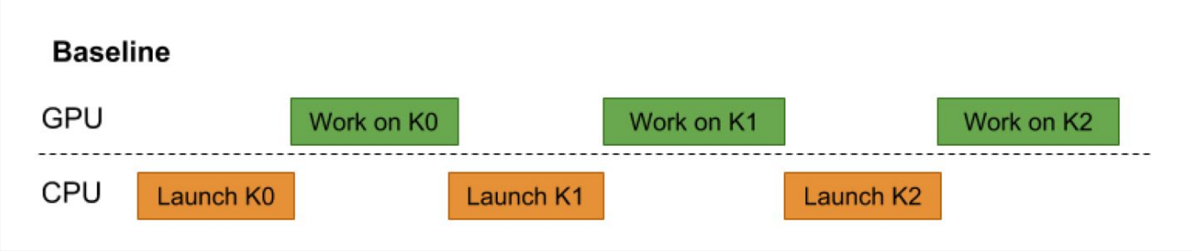
HIGH CPU SUBMISSION **OVERHEAD -> THREAT** TO SYSTEM PREDICTABILITY

Starting from the ECRTS19 [1] Paper

- Four submission models (baseline, CUDA graphs, CUDA CDP and Vulkan)
- Four different baseline latencies as a function of how many kernels per task are submitted
- **How each of these methodologies suffer from memory interference**

*[1] Cavicchioli, Roberto, et al. "**Novel methodologies for predictable CPU-to-GPU command offloading**." 31st Euromicro Conference on Real-Time Systems (ECRTS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.*

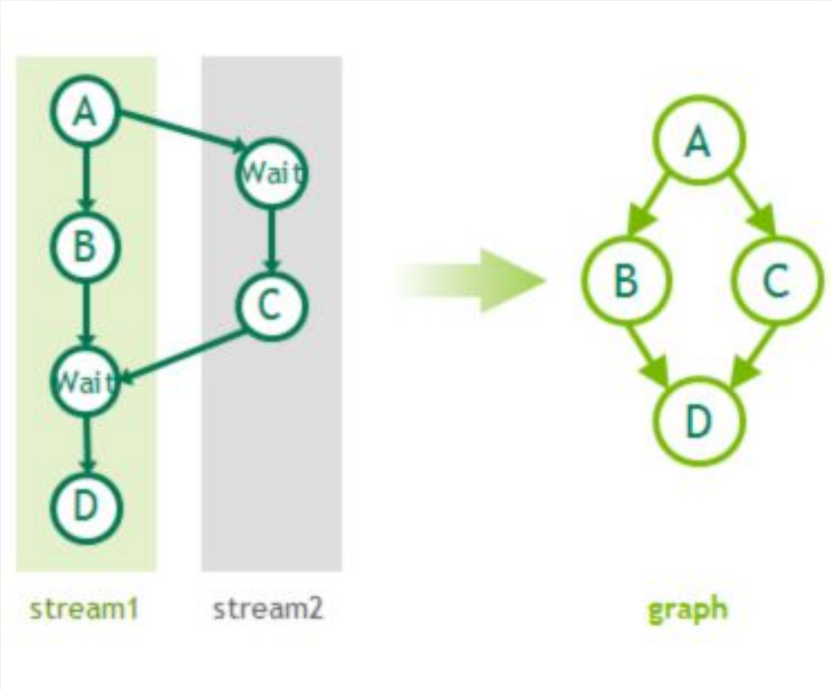
Submission Model: CUDA Baseline



- Copy and Compute commands (kernels) are **constantly streamed to the GPU**.
  - **CUDA streams –> FIFO QUEUE**.

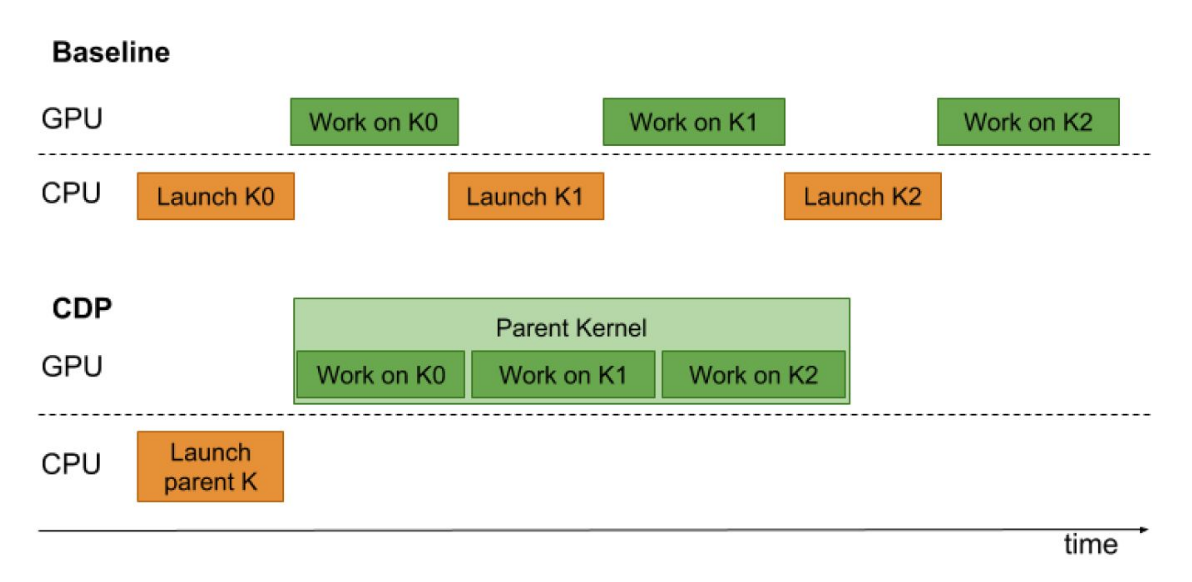
- CPU is constantly busy submitting commands
  This takes time and threats predictability:
  - a delayed submission -> **delayed GPU-side** execution...

Submission Model: Cuda Graphs



- Allows to **construct graphs of GPU commands**
- Allows **copy operations with arbitrary intra- and inter-stream** synchronization

Submission Model: Cuda Dynamic Parallelism (CDP)



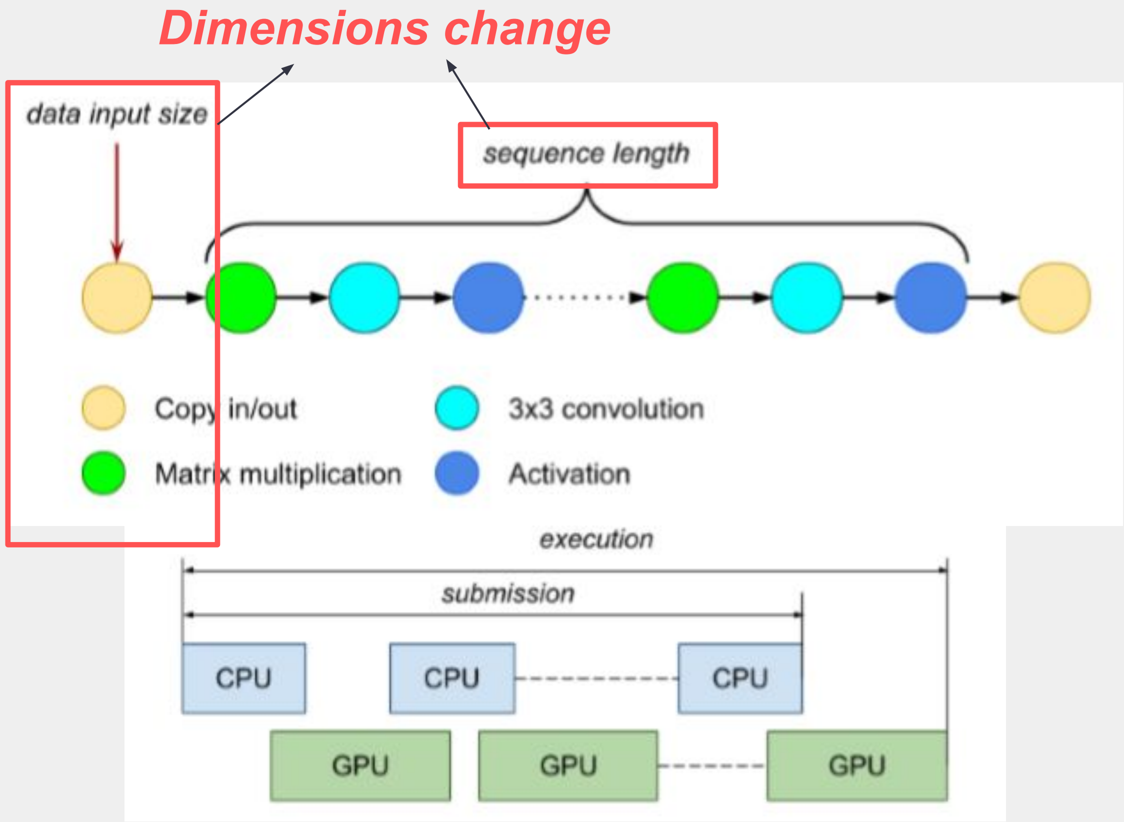
- Allows a **kernel to launch «nested» kernels**
- **No OS/Driver** intercations **between nested calls**
- Beneficial performance with recursive algorithms with variable depth recursion

- **Introduces Stalls** for deep call-stack value
- **Involves kernels, not host jobs or copies**

Submission Model: Vulkan

- **Alternative to CUDA**
- **Recently** (2016) released API specifications (Khronos Group) for both **graphics and compute** on massively parallel accelerators
- **OpenGL successor**, but no assumptions w.r.t. GPUs or application domain
- **Novel paradigm for CPU->GPU interactions** (lower level abstraction, no verification/validation at runtime)
- … specs say Vulkan is **predictable**…

ECRTS19 Experimental Setup

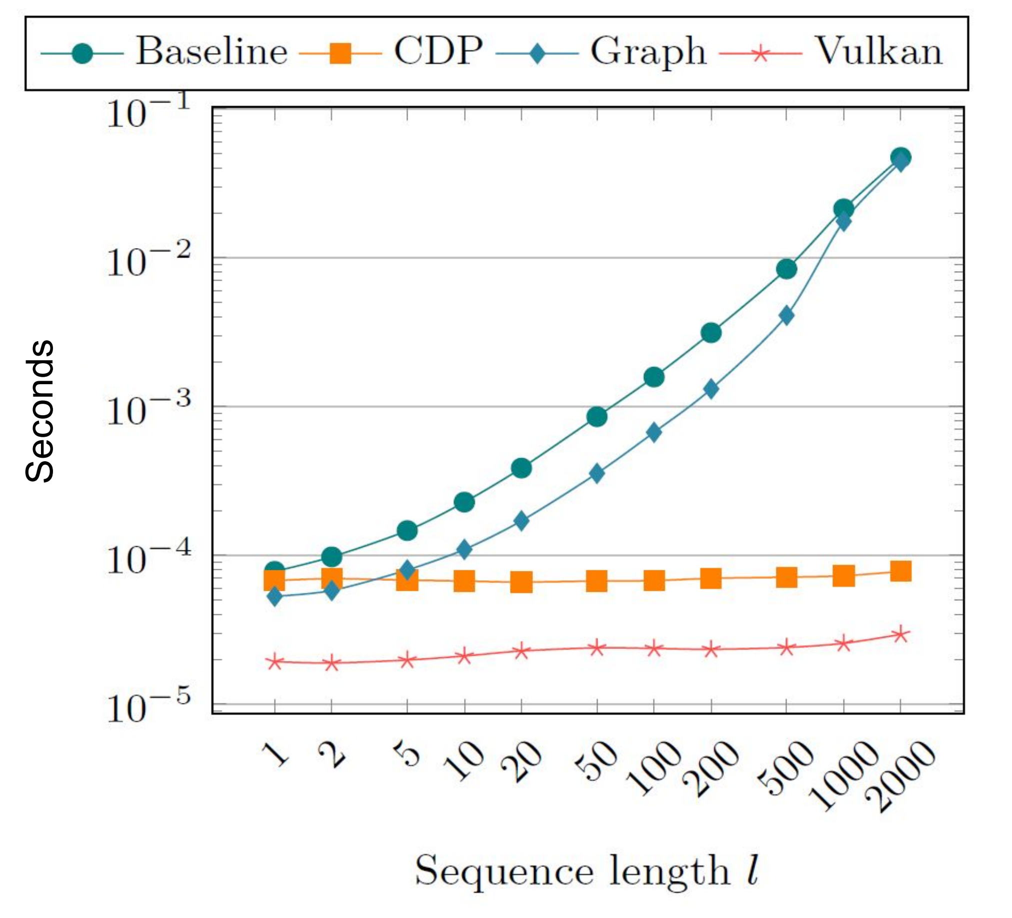


- **Submission latencies**
- Execution times
- Driver interactions
  - CUDA baseline
  - CDP
  - CUDA Graphs
  - Vulkan (VK)

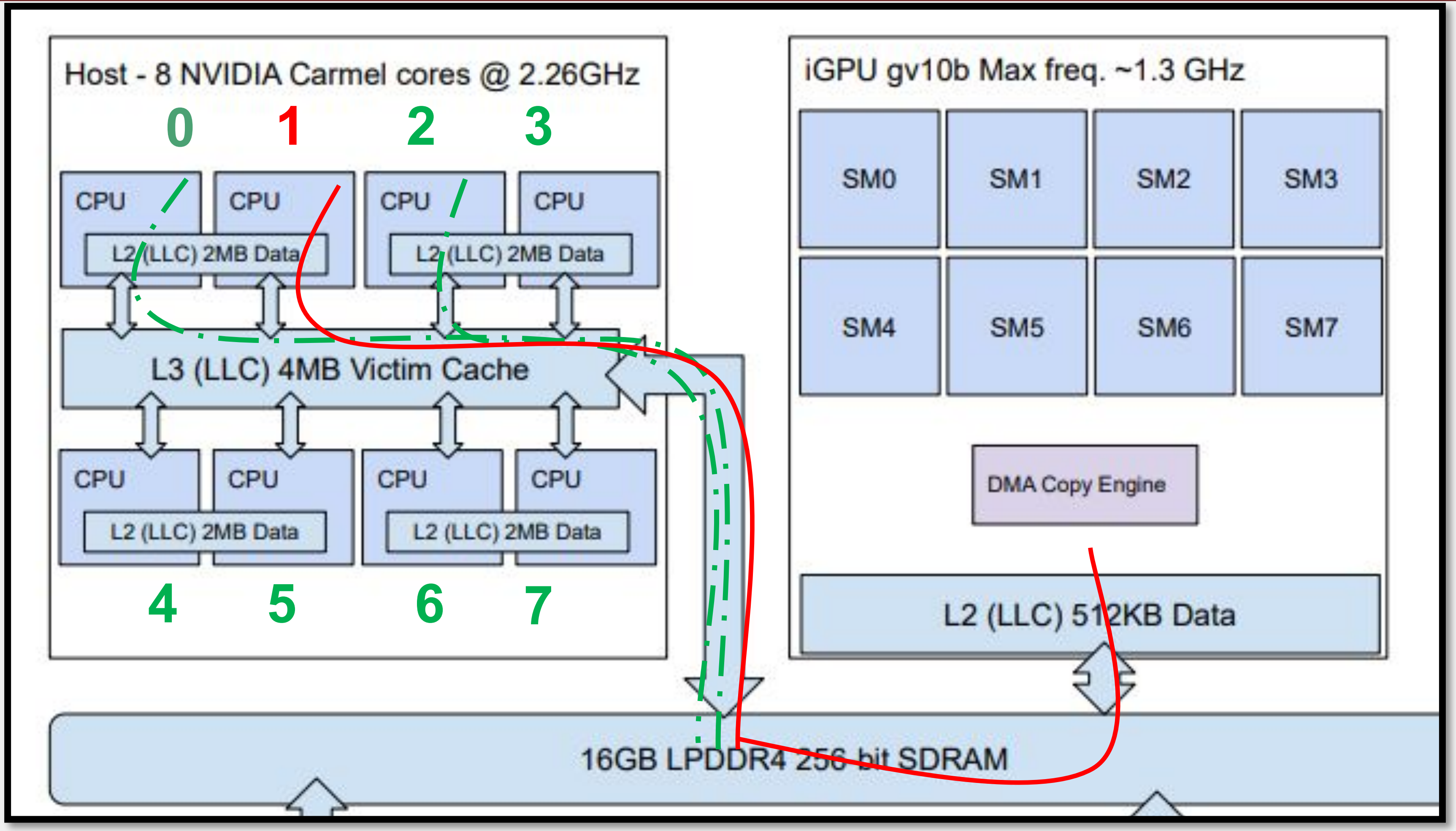
*[1] Cavicchioli, Roberto, et al. "**Novel methodologies for predictable CPU-to-GPU command offloading**." 31st Euromicro Conference on Real-Time Systems (ECRTS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.*

Conclusion of the Previous Work (ECRTS19)

Average **submission time**, **<u>without</u> interferences**.

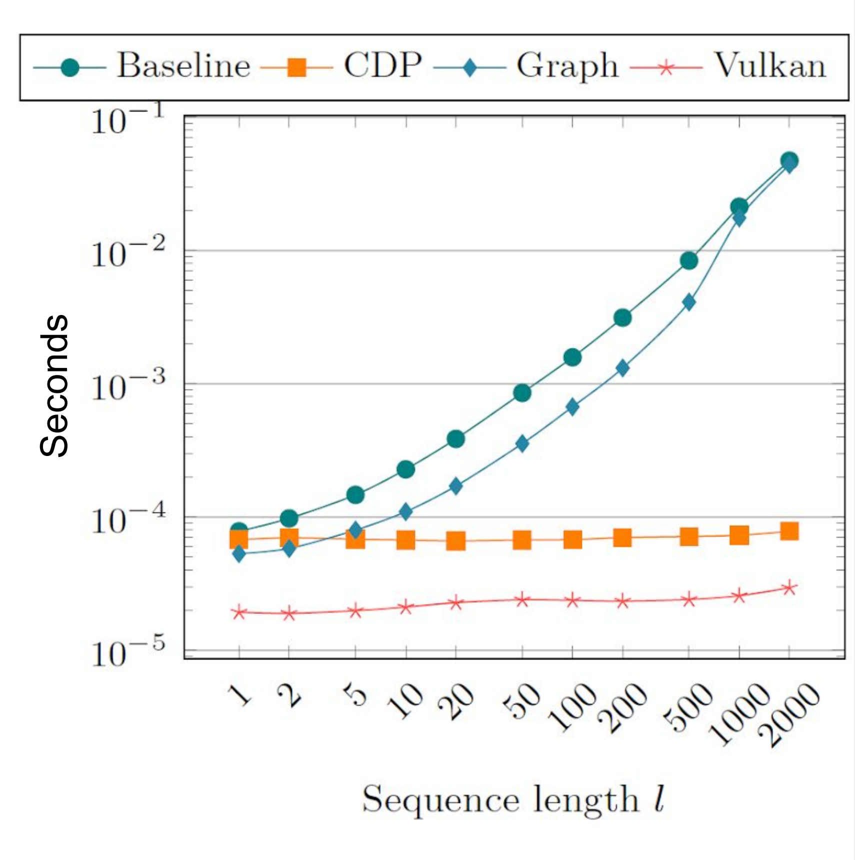


What Happens if...

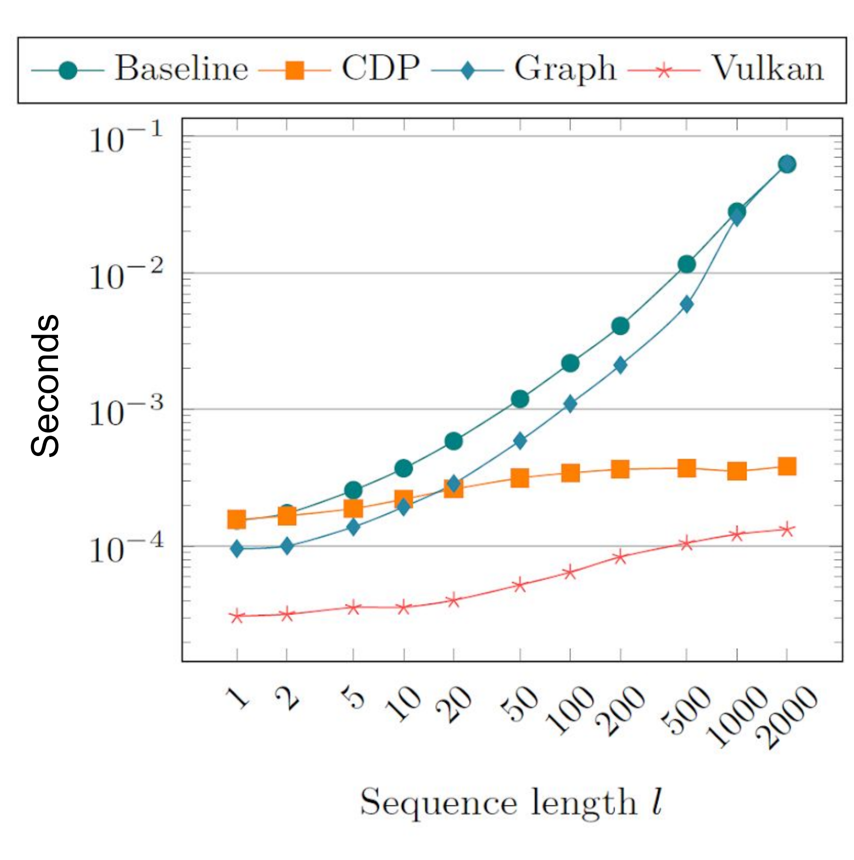


Submission Time with Interference

No Interference (AVG VALUE)

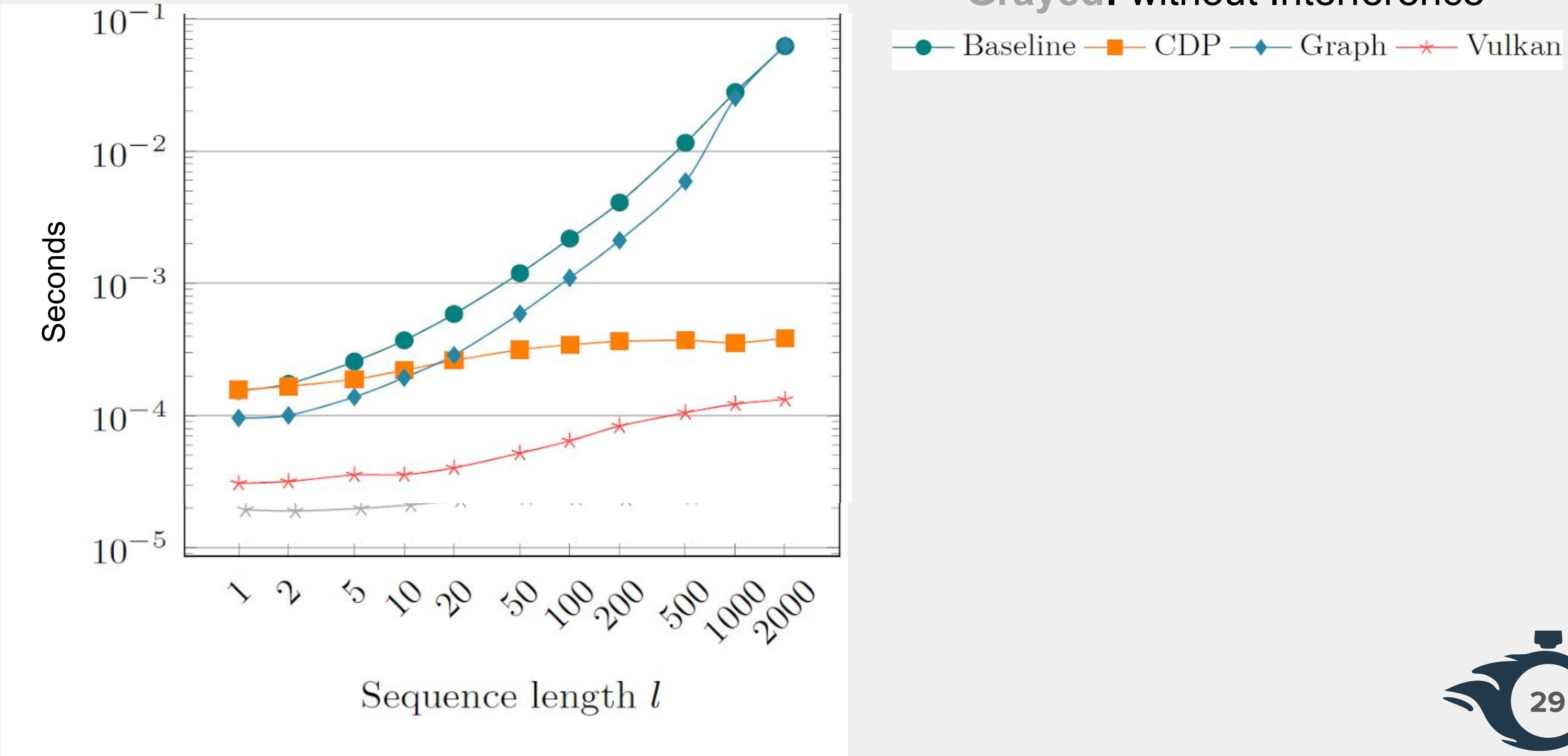


7 thread Interference (AVG VALUE)



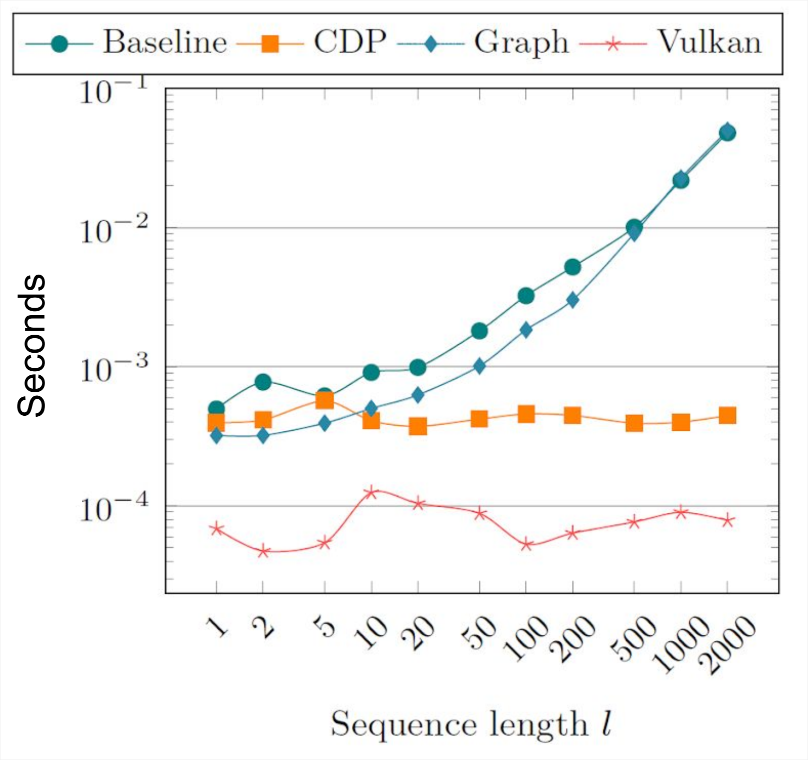
A Closer Look on the Effect of Interference

Colored: *with* Interference
Grayed: without Interference

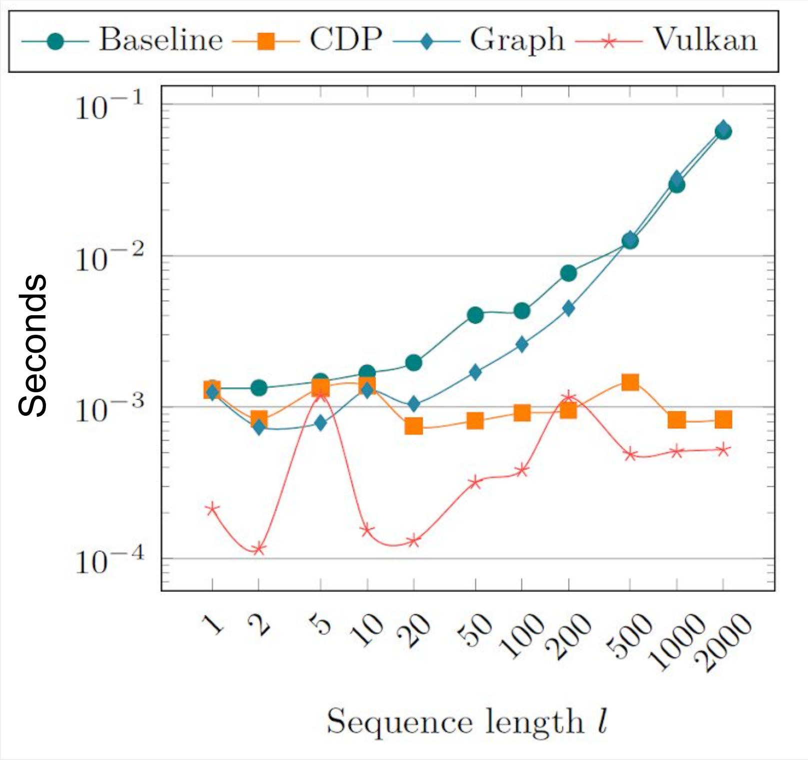


With the Maximum Measured Value

No Interference (MAX VALUE)

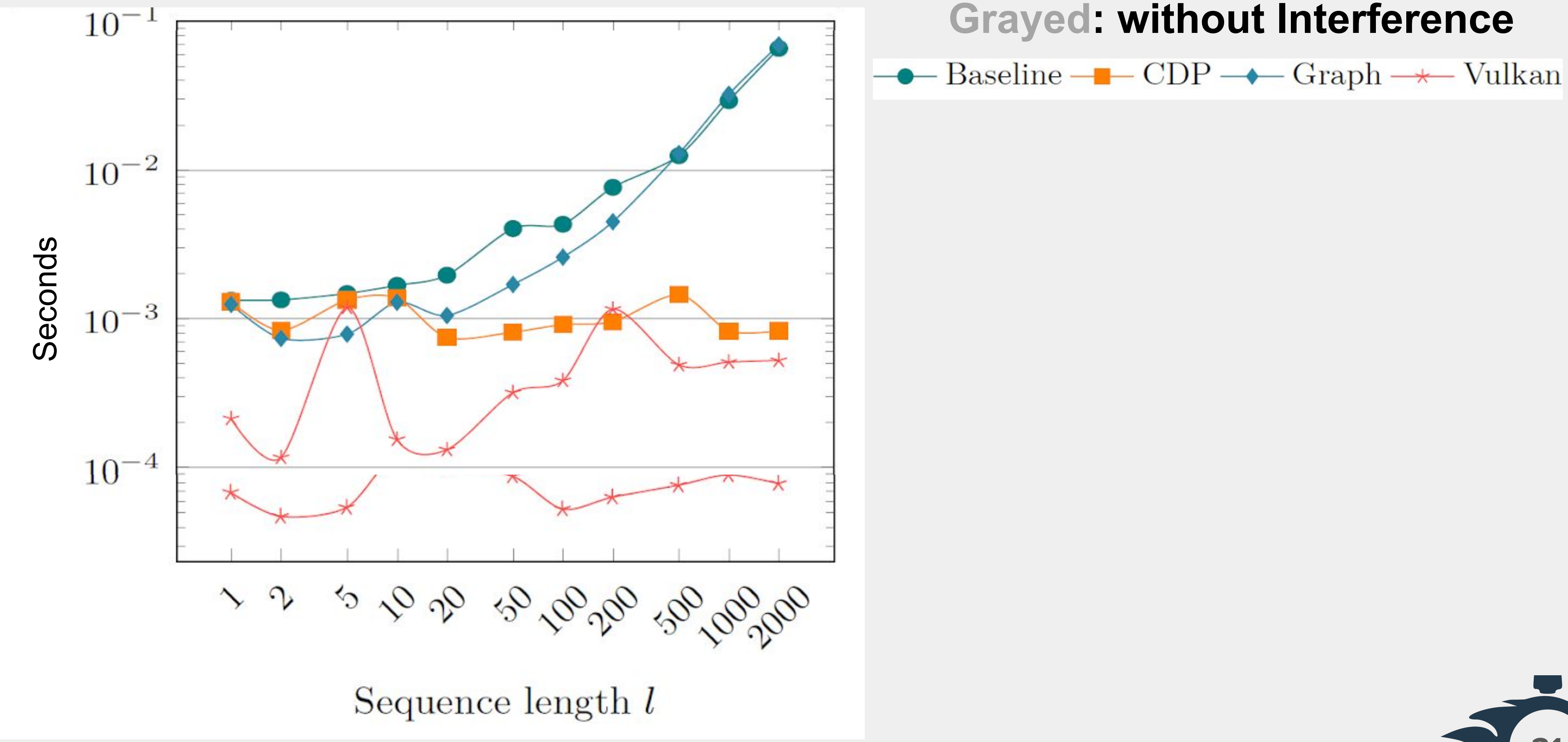


7 thread Interference (MAX VALUE)



A Closer Look ...

Colored: with Interference
Grayed: without Interference



Maximum Experienced Latency Degradation

Average Submission Factor (sequence length value)

interferences_THs	Baseline	CDP	Cuda Graph	Vulkan
1	2.24 (1)	2.27 (1)	1.97 (1)	1.63 (10)
2	1.18 (1)	2.01 (2000)	1.13 (200)	1.15 (2)
3	1.16 (100)	2.50 (2000)	1.20 (200)	1.54 (2000)
4	1.22 (1)	2.46 (2000)	1.21 (200)	1.55 (2000)
5	1.27 (1)	3.06 (1000)	1.29 (1000)	2.46 (2000)
6	1.34 (1)	2.99 (1000)	1.38 (2000)	2.31 (2000)
7	1.97 (1)	5.20 (500)	1.80 (1)	4.77 (1000)
1*	1.16 (1)	4.14 (200)	1.35 (200)	3.19 (1000)

Maximum Submission Factor (sequence length value)

interferences_THs	Baseline	CDP	Cuda Graph	Vulkan
1	1.45 (5)	1.49 (10)	1.45 (10)	2.22 (1)
2	1.55 (1)	1.19 (1)	1.37 (5)	1.93 (5)
3	1.21 (20)	1.58 (10)	1.61 (2)	1.56 (2)
4	1.45 (5)	1.58 (200)	1.58 (1)	1.78 (2)
5	2.24 (5)	2.10 (1000)	1.43 (50)	16.28 (1)
6	2.08 (1)	2.71 (100)	1.81 (5)	5.61 (2)
7	2.68 (1)	3.69 (500)	3.89 (1)	21.83 (5)
1*	1.98 (1)	2.15 (1)	2.06 (1)	2.55 (2000)

Conclusion and What’s Next…

- **CPU to GPU submitters must be scheduled as well!**

- Submission methodologies play a huge role in terms of CPU overhead
  - Vulkan still works best…
  - …but significantly suffers from memory interference

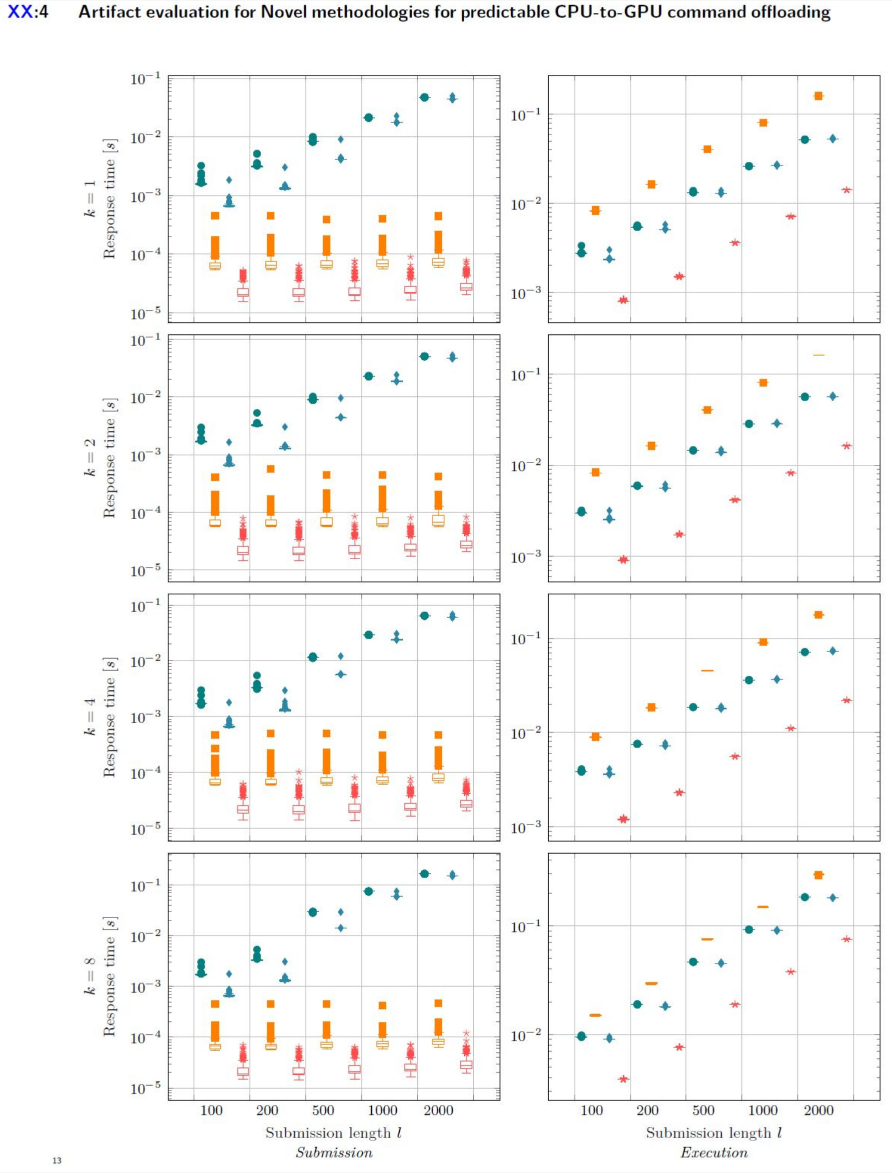
These measures will help the system engineers to account for a more accurate worst-case execution/response time.

Questions?

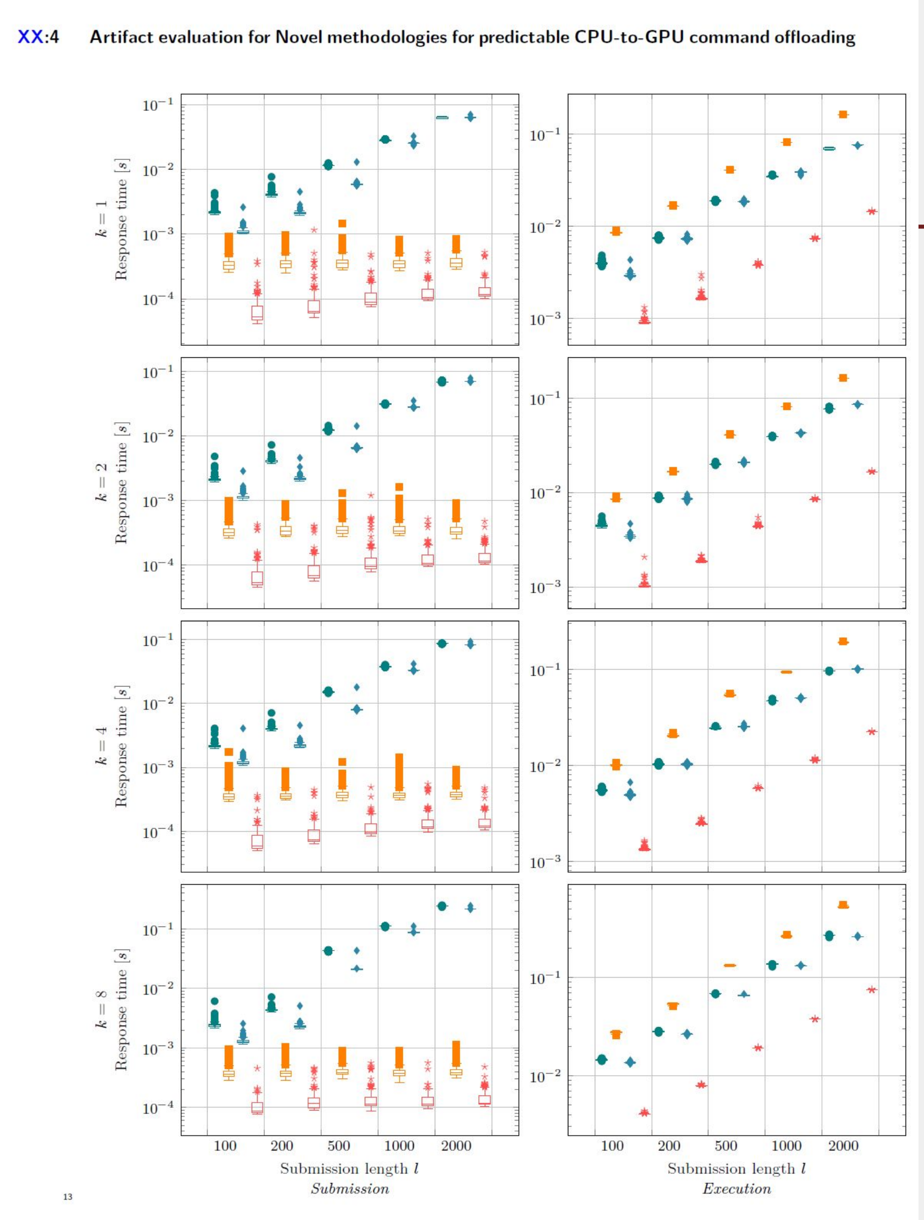
Thank you!

Alessio Masola

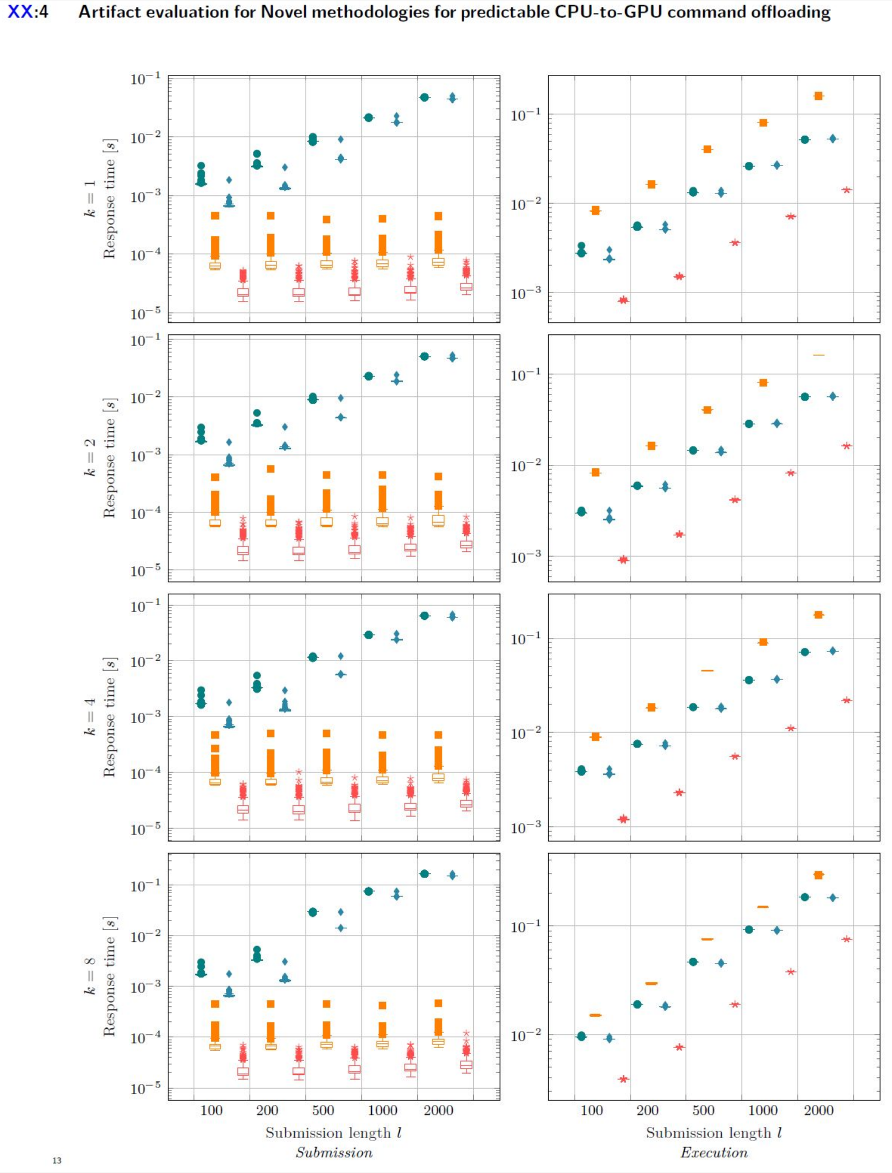
Baseline



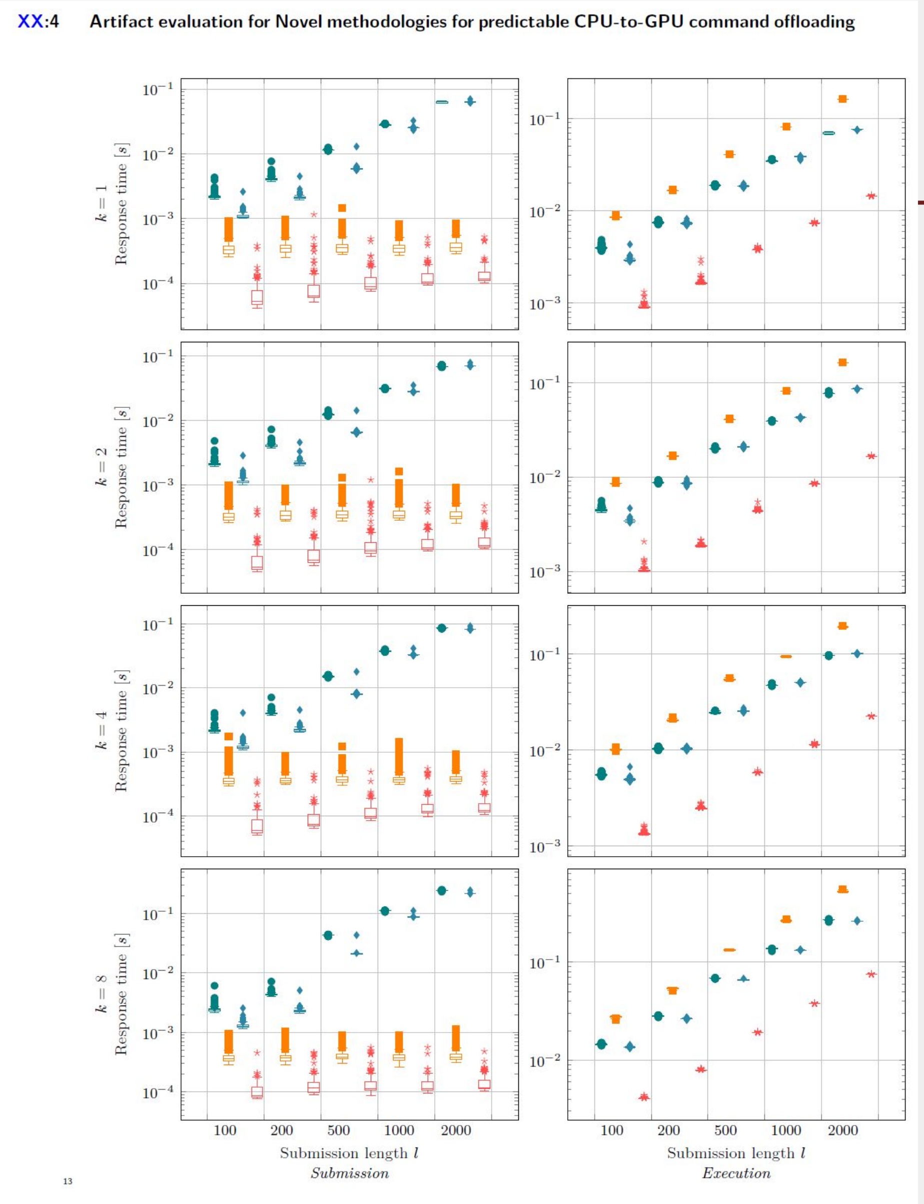
7 Interferences



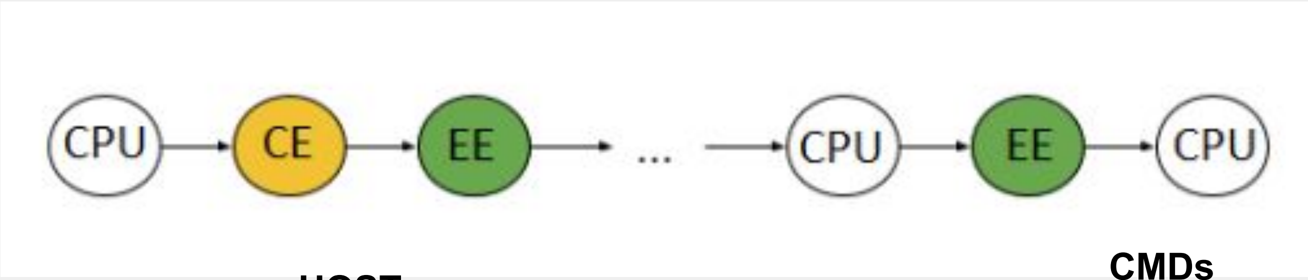
Baseline

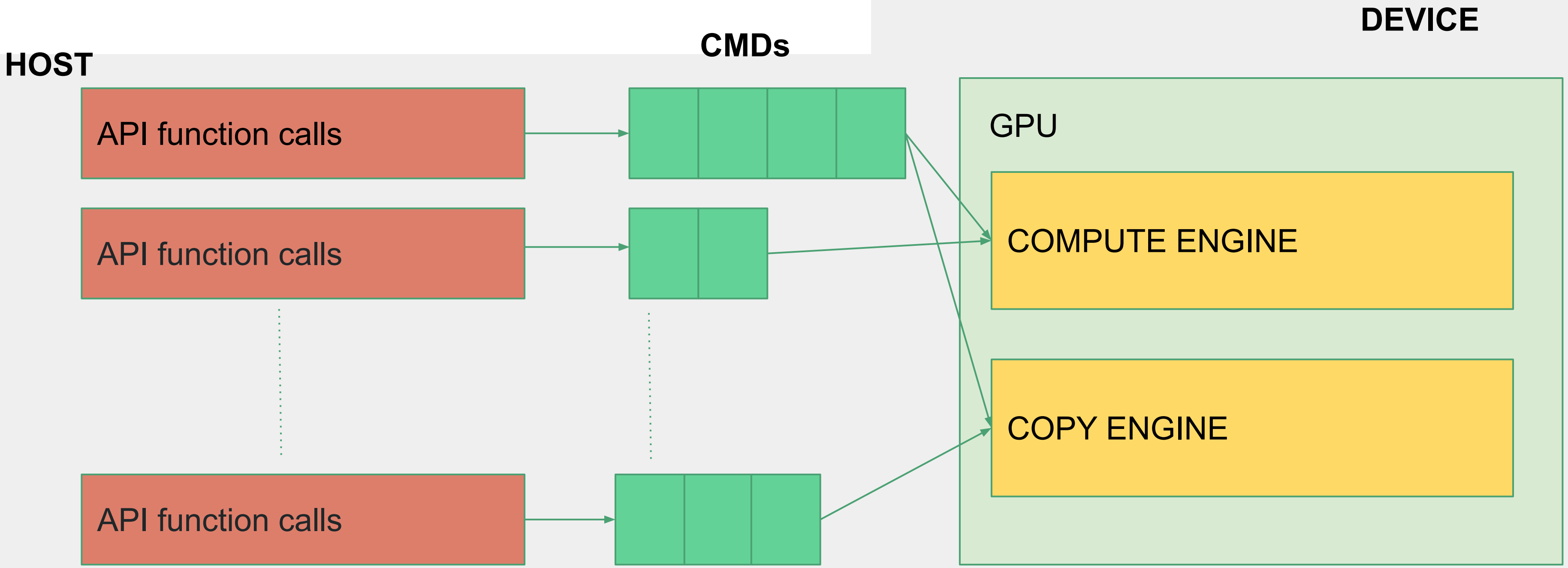


7 Interferences



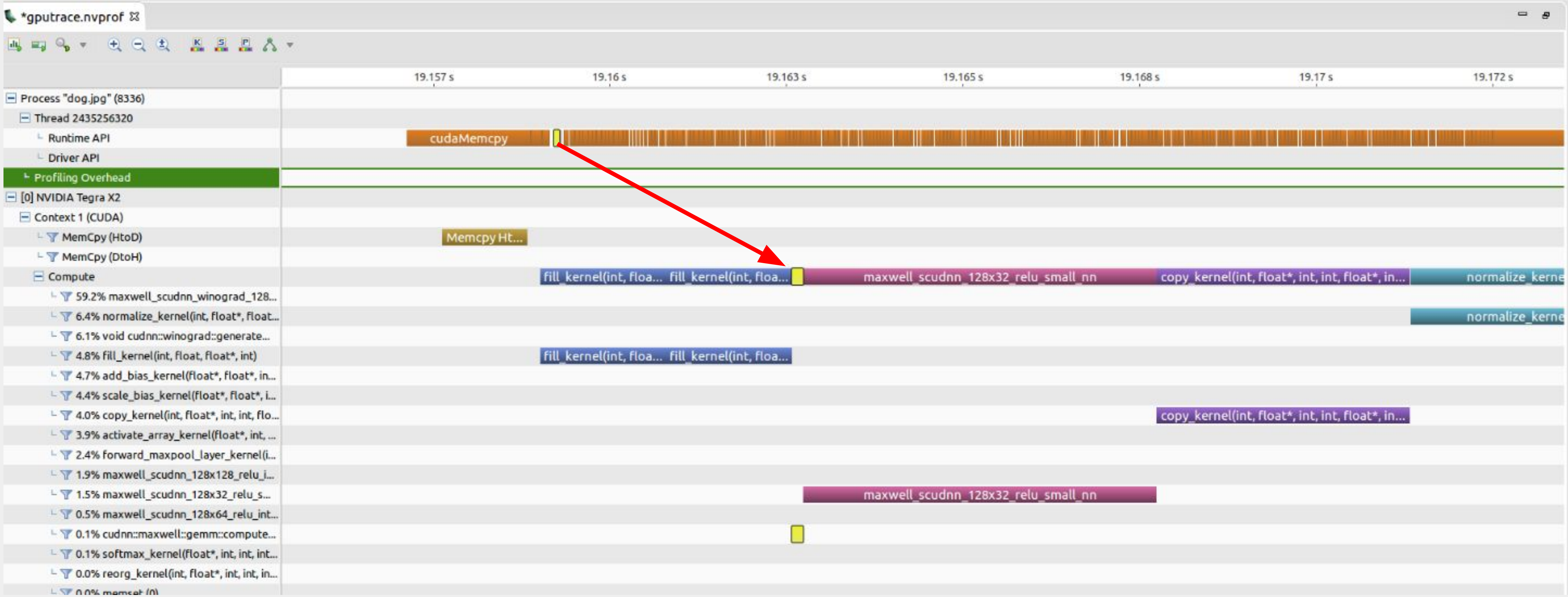
How it works? Submission Model





impact of CPU-to-GPU kernel submissions may be indeed relevant for typical real-time workloads

Nvprof trace (YOLOv3)



Submissions and Models

Comparison between:

Baseline *Vs* **Cuda CDP** *Vs* **Cuda Graph** *Vs* **Vulkan**