UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

High-Performance Real-Time Lab

The importance of memory in the next generation of real-time systems

Paolo Burgio
paolo.burgio@unimore.it

Future embedded systems

Industry 4.0

The four horsemen

1. **Heavy workloads**
   - Sensor-fusion and image-processing
2. **Reduced power consumption**
   - Smaller batteries and renewable power sources
3. **Quickly interact** with the environment
   - Prompt elaboration of sensor data
4. Run **highest criticality** workloads
   - Replacing safety-critical human activities

Internet-of-Things

Cyber-physical systems

Autonomous driving

Artificial intelligence

Health and medicine

Real-Time multi-core systems?

Multi- and many-core platforms are the solution for 1-2(-3)

- ✓ Climbing "the power wall"
- ✓ High Performance @ poor Watts

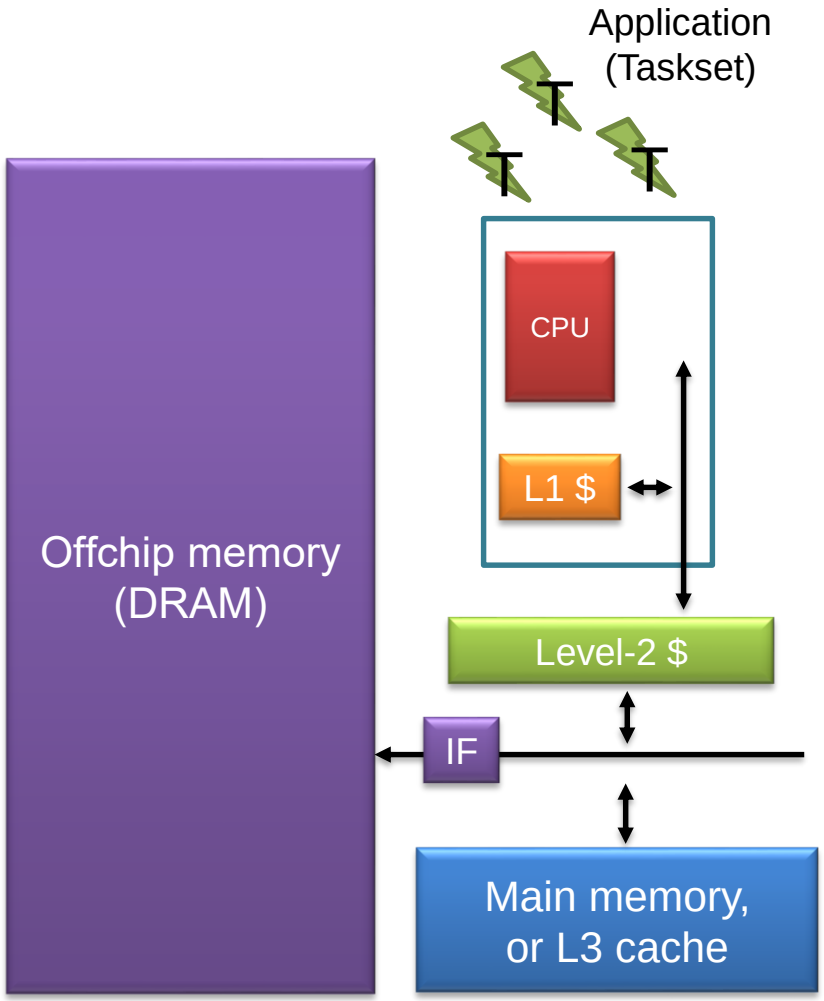
Real-Time system: produce result in a guaranteed/bounded amount of time

- ✓ By construction
- ✓ Application fields: automotive, avionics, industry, medical…

The keyword: predictability

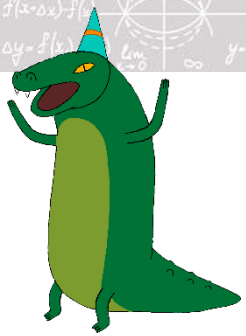
- ✓ Provide the correct result.…when expected
- ✓ The system must be simple to analyze

Real-Time systems – traditional approach



Single-core, multiple tasks/applications

1. Analyze the system (HW/SW)
2. Derive a (mathematical?) model
3. Do some ~~magic~~ mathematics…



…guaranteed timing bounds!

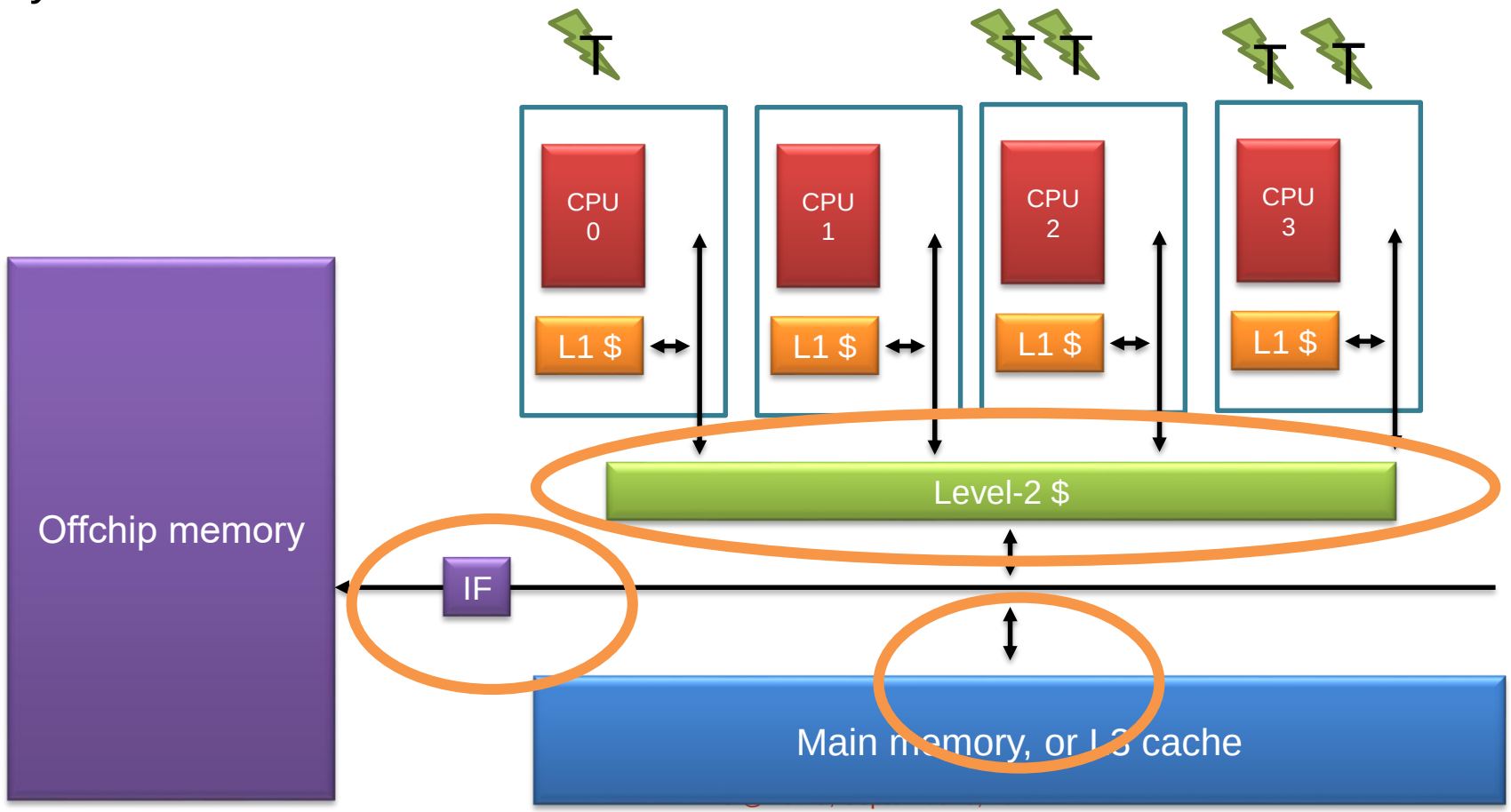
Optimal sharing of the core between task

- ✓ ..and guaranteed by construction
- ✓ *Scheduling* (also, mapping)

Multi-core systems

Architectural bottlenecks

- ✓ Shared memory banks
- ✓ Caches ($)
- ✓ I/Os

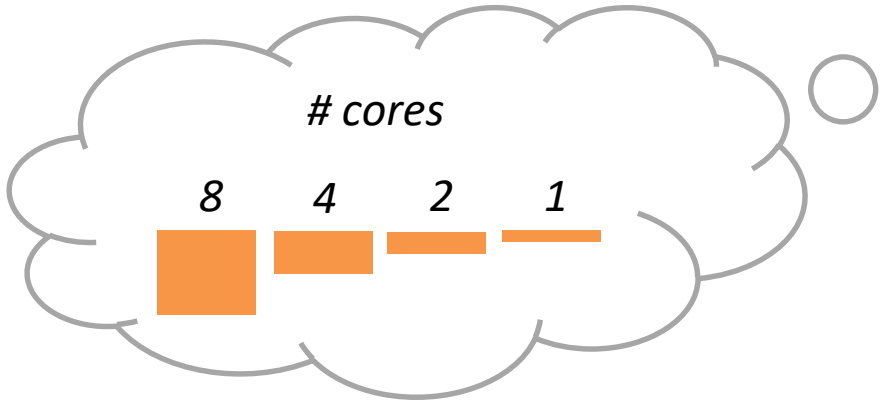


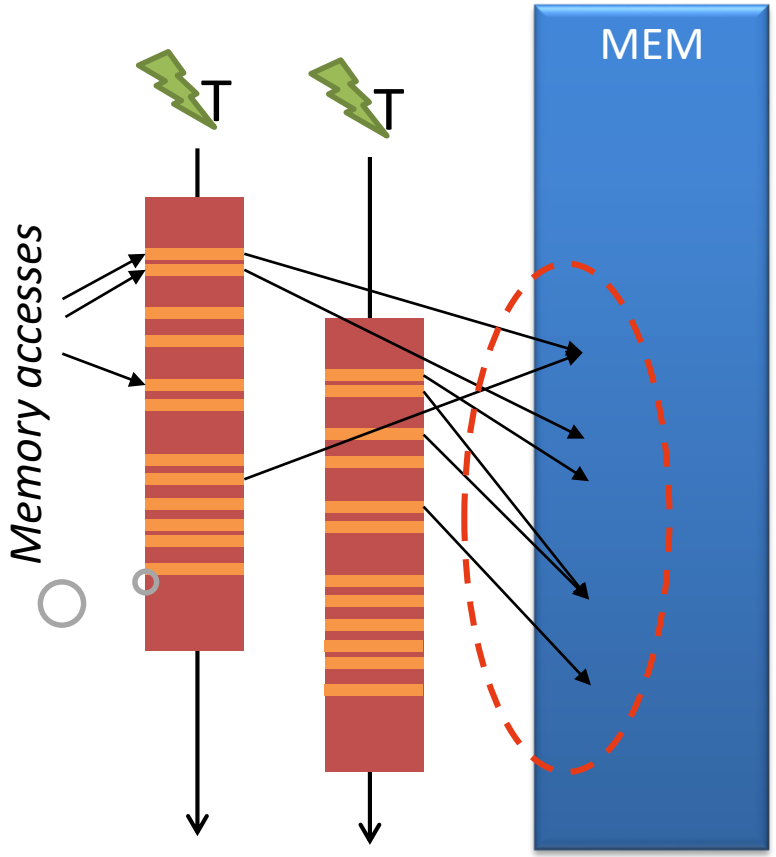
It's (mainly) a memory issue!

Beyond traditional tecnhiques

1. More parameters
   - Shared resources (e.g., memory, SSDs, IOs, caches..)
   - The complexty of analysis grows exponentially w/number of cores

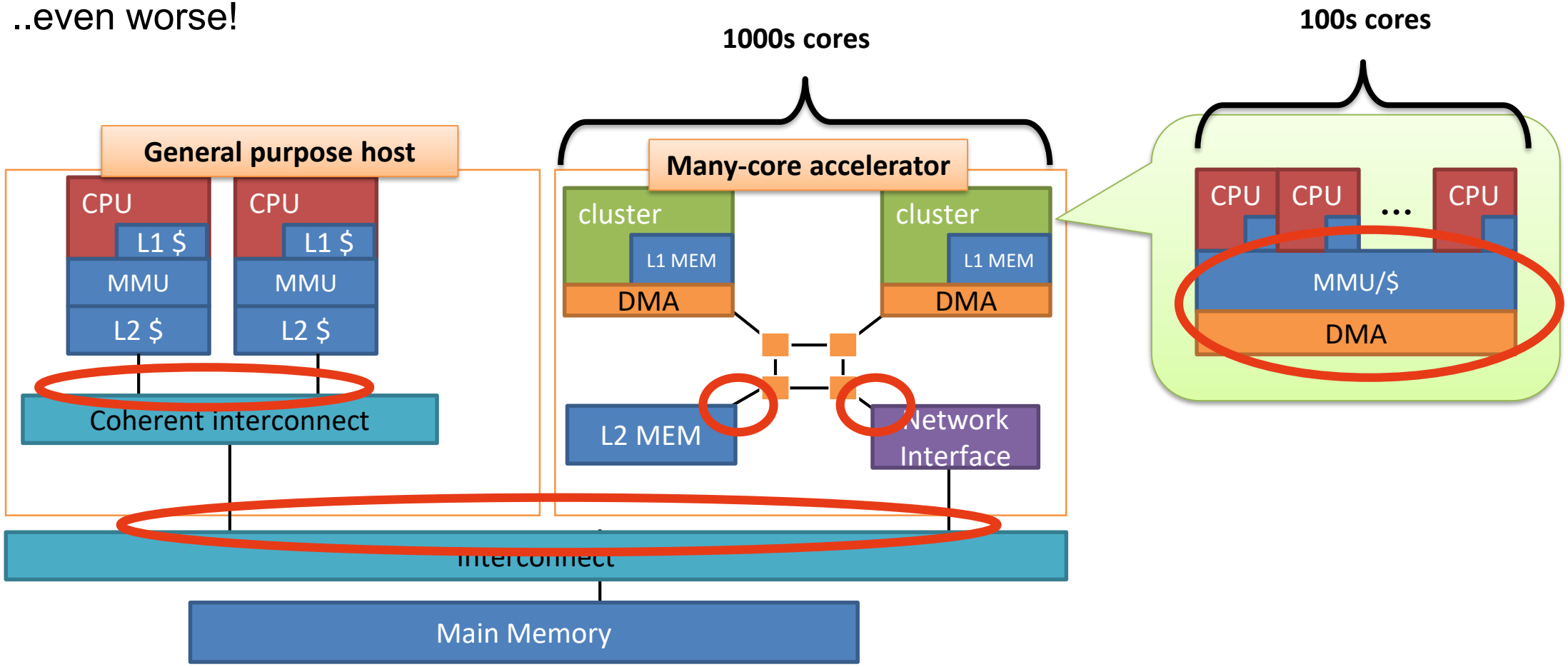
2. Mem accesses: instead of thin lines, big bars
   - The mostly accessed resource in the system
   - Traditional techniques are too conservative (bounds too high)





Many-core systems

- ✓ Thousands cores arranged in CLUSTERS
- ✓ Host-acccelerator architecture (e.g., GP-GPUs)
- ✓ ..even worse!



Knowledge of the platform is power

Two motivating examples

✓ Both from real systems

1. Many-core accelerator-based platforms
   - Quad-/Octa-core as host
   - Integrated GPU – iGPU of FPGA
   - Powerful enough to run neural networks

2. Reference industrial system
   - Multi-core ARM
   - Multi-OS (embedded Linux + Win for UI)
   - Hypervisor-based

Testbed #1: "automotive" platforms

Qualitatively analyze and characterize the conflicts due to parallel accesses to main memory by both CPU cores and iGPU

1. NVIDIA Tegra K1 w/Kepler GPU
2. NVIDIA Tegra X1 w/Maxwell GPU
3. NVIDIA Tegra X2 w/Parker GPU – automotive-grade
4. Intel i7-6700 w/intel GPU
5. Xilinx Zynq Ultrascale multi-core + FPGA (+GPU)

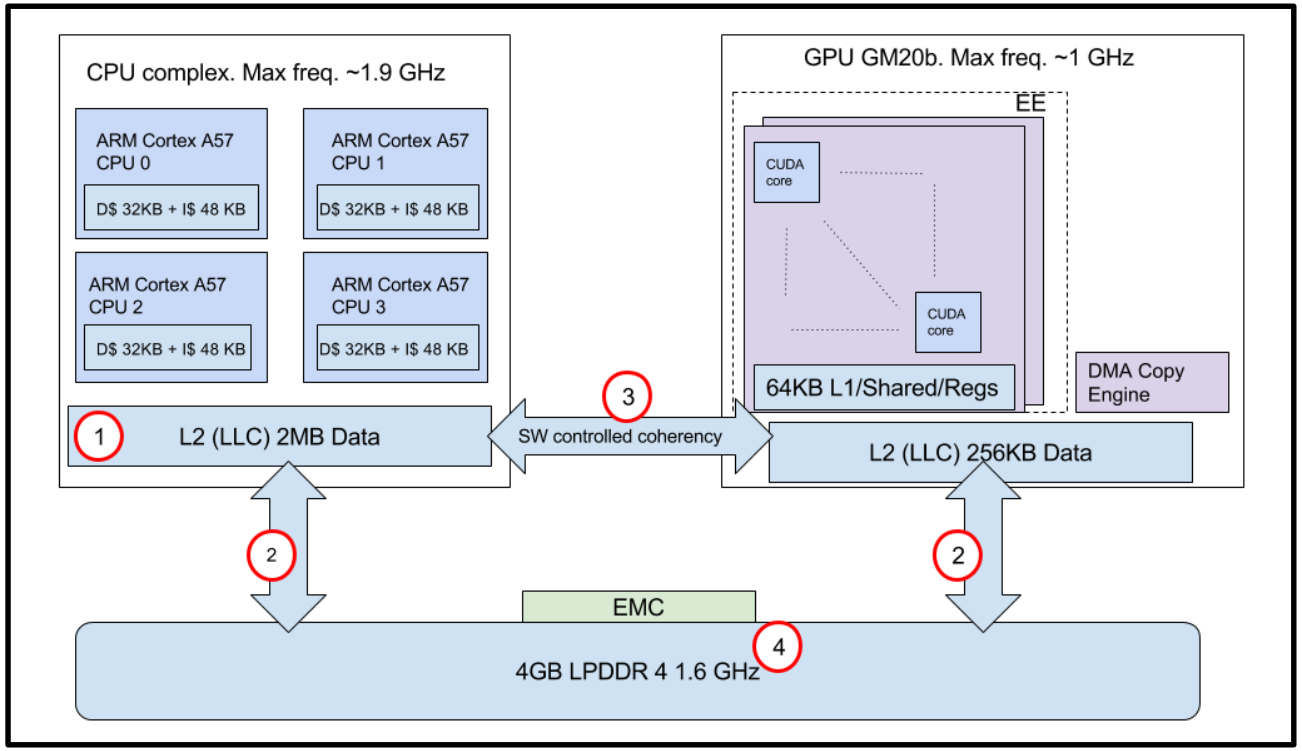
HERCULES

Roberto Cavicchioli, Nicola Capodieci and Marko Bertogna, *"Memory Interference Characterization between CPU cores and integrated GPUs in Mixed-Criticality Platforms"*, 22nd IEEE International Conference on Emerging Technologies And Factory Automation

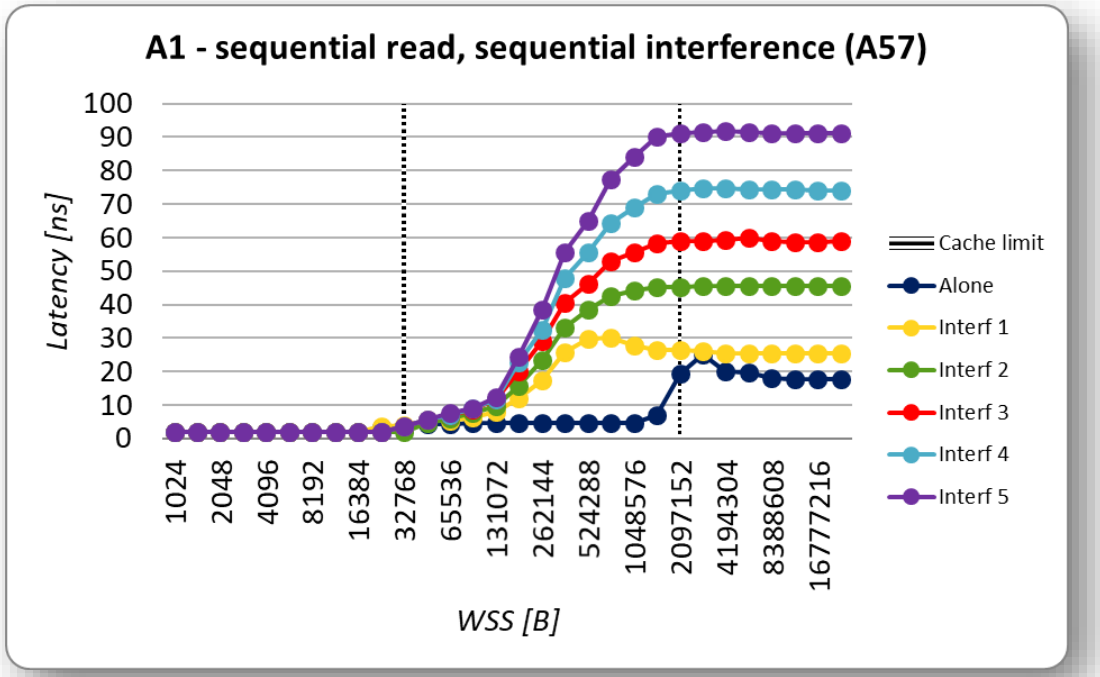
NVIDIA Tegra K2

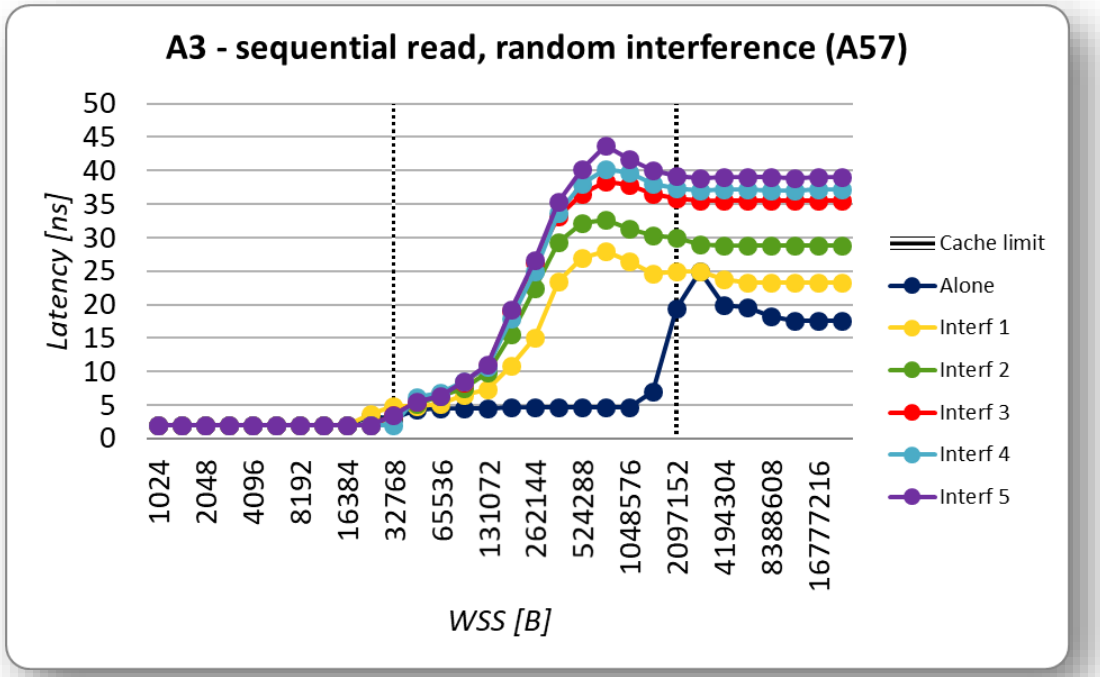
- ✓ Shared memory between CPU/GPU complex
  - "Unified Virtual Memory"
  - Unlike traditional "discrete" GPU systems

Notable contention points (1)



Test 'A' - Tegra X2 – A57

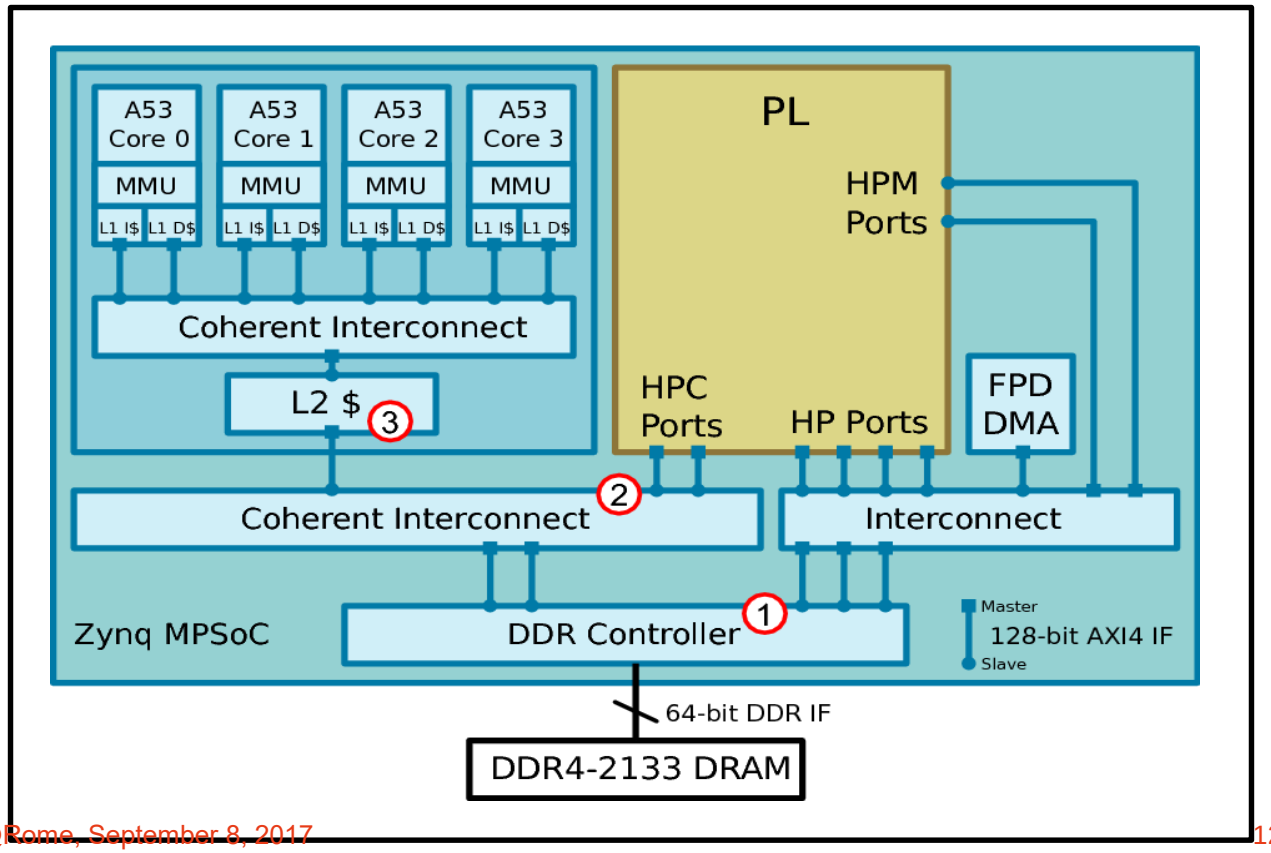




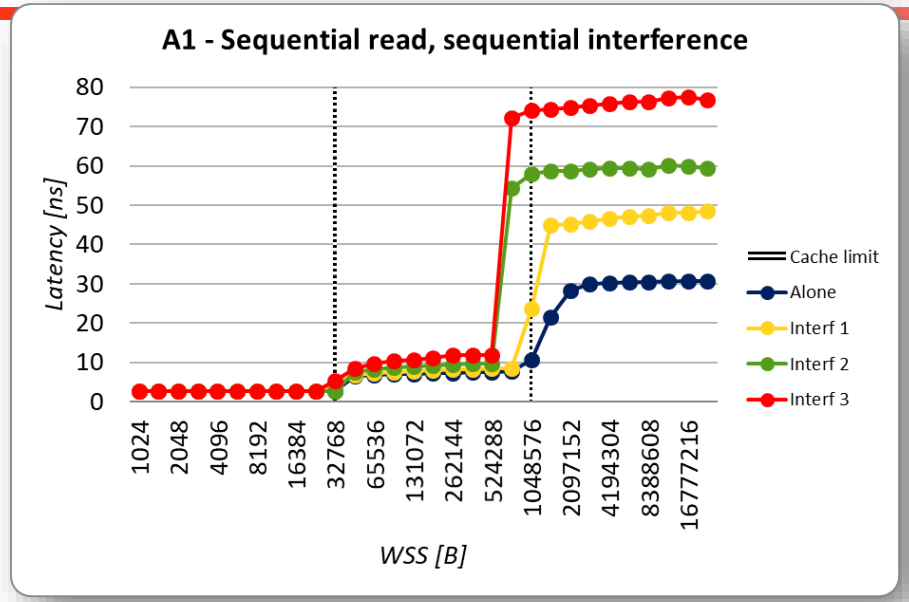
Xilinx Zynq Ultrascale

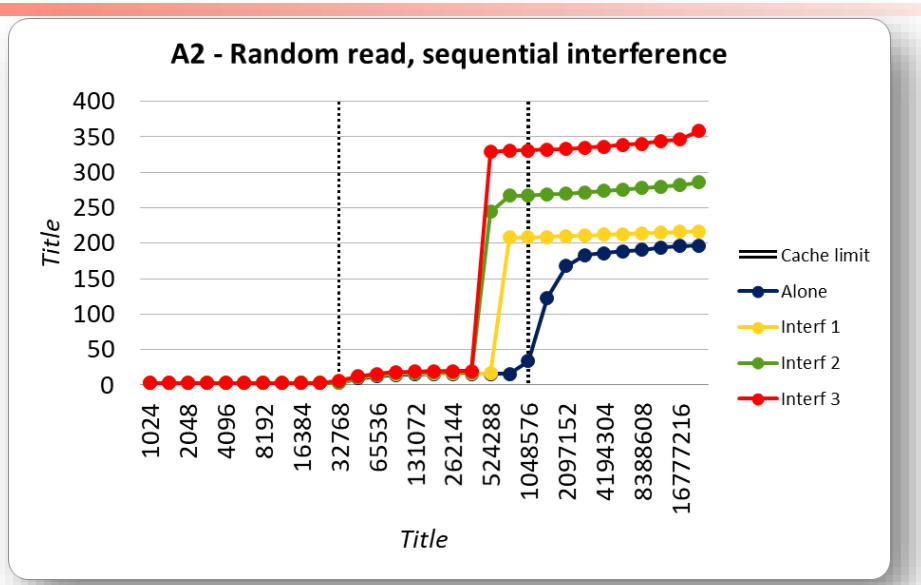
- ✓ Last-generation FPGA-based heterogeneous SoC
  - FPGA = (re-)programmability
- ✓ ARM A53 Quad-core as host *"PS"*
- ✓ FPGA as accelerator *"PL"*

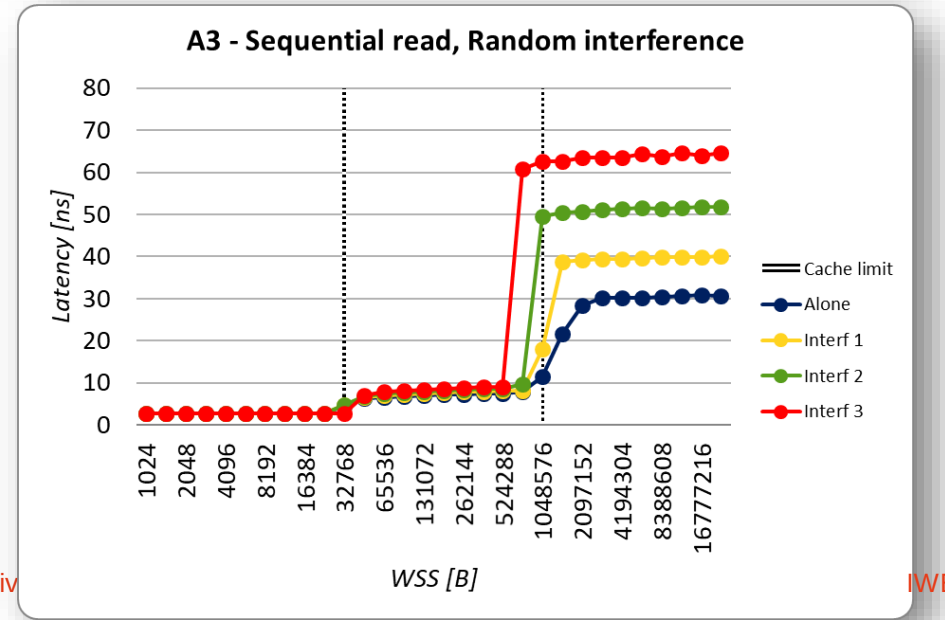
Notable contention points (1)

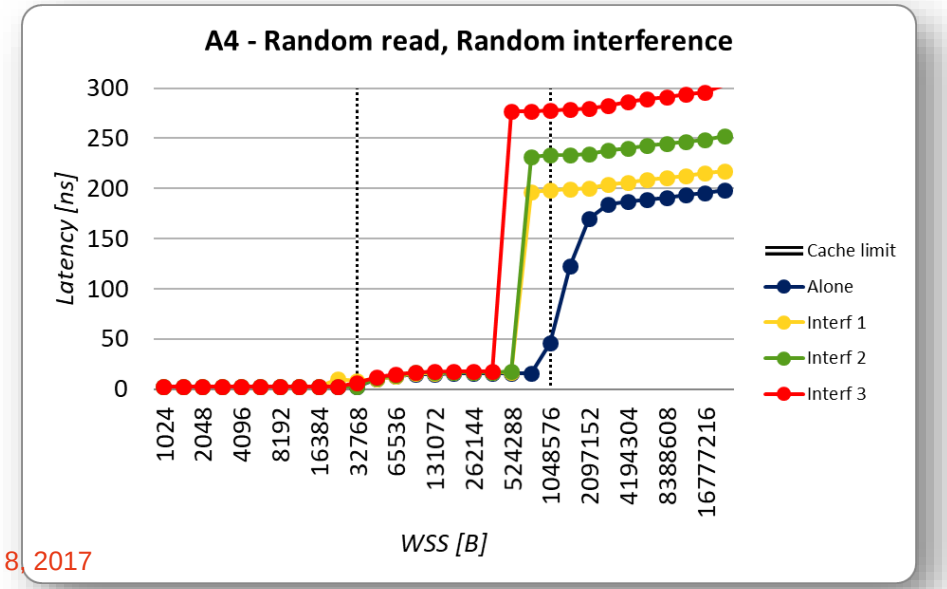


Test 'A' - Xilinx Zynq

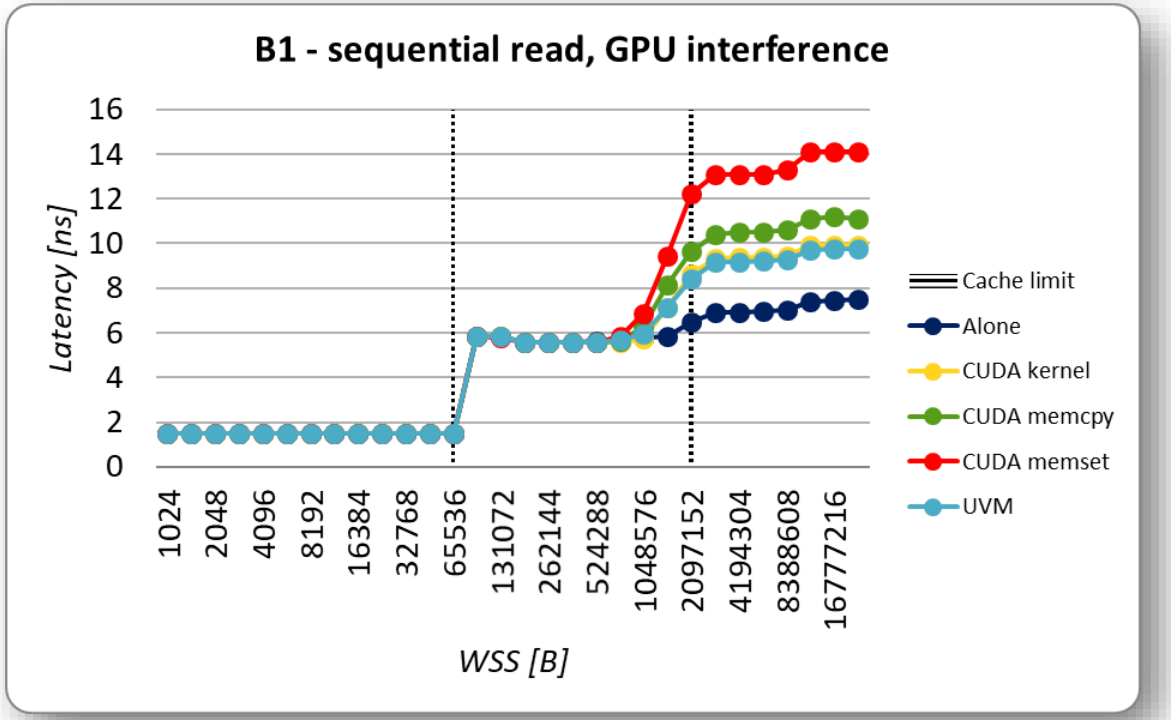


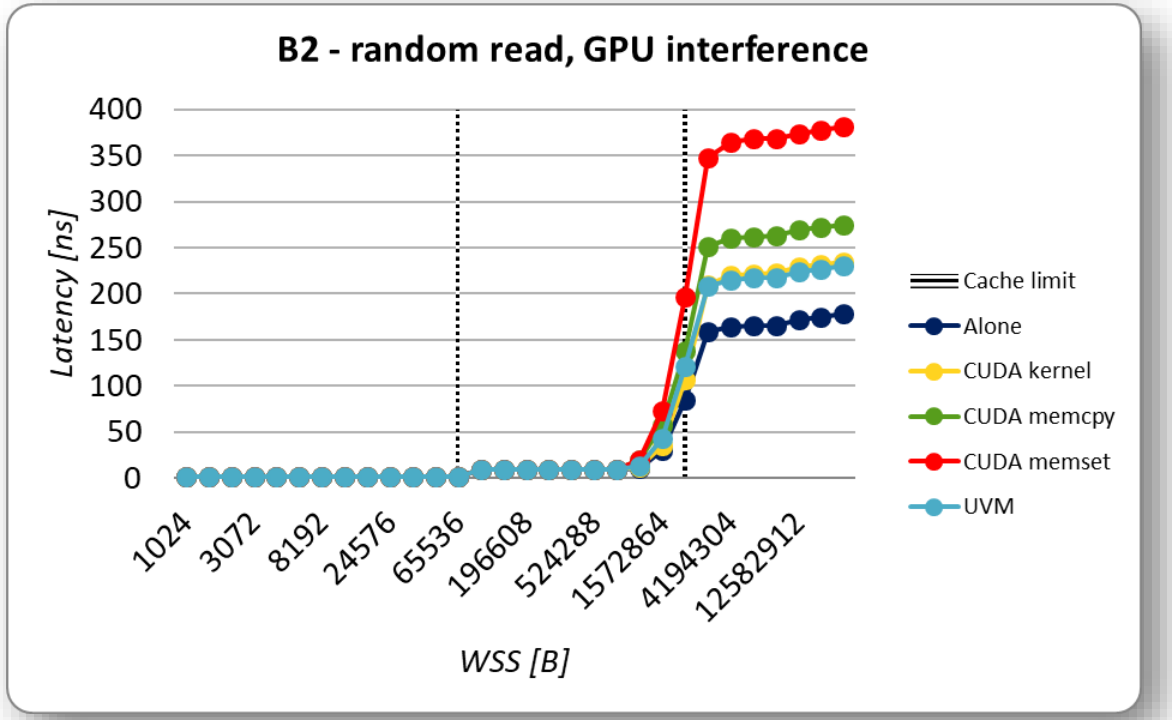




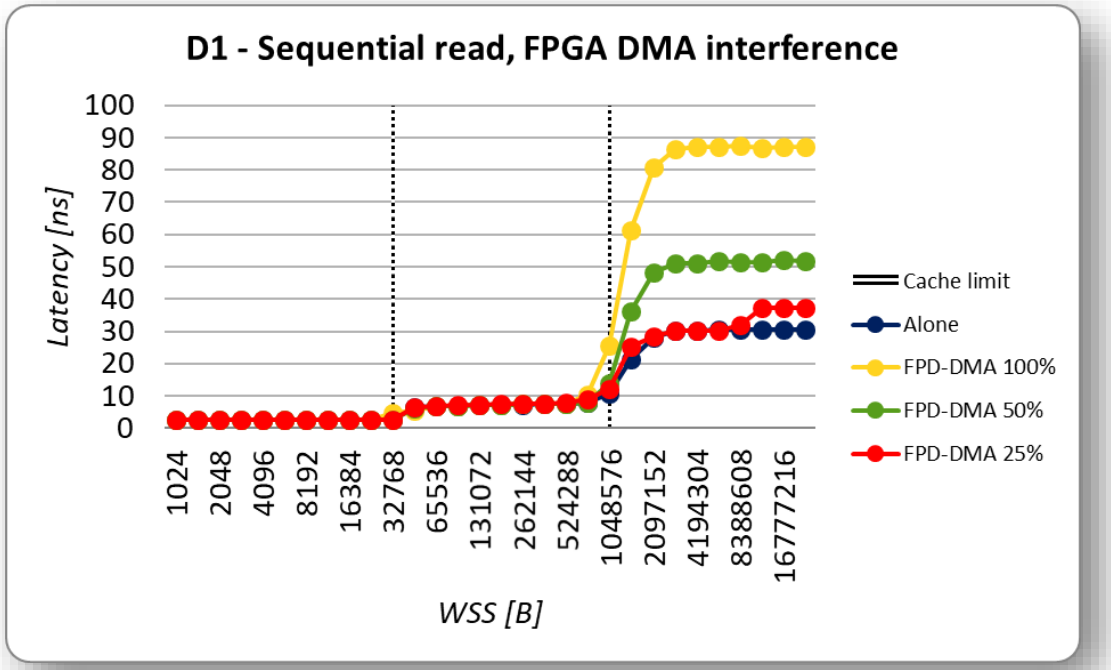


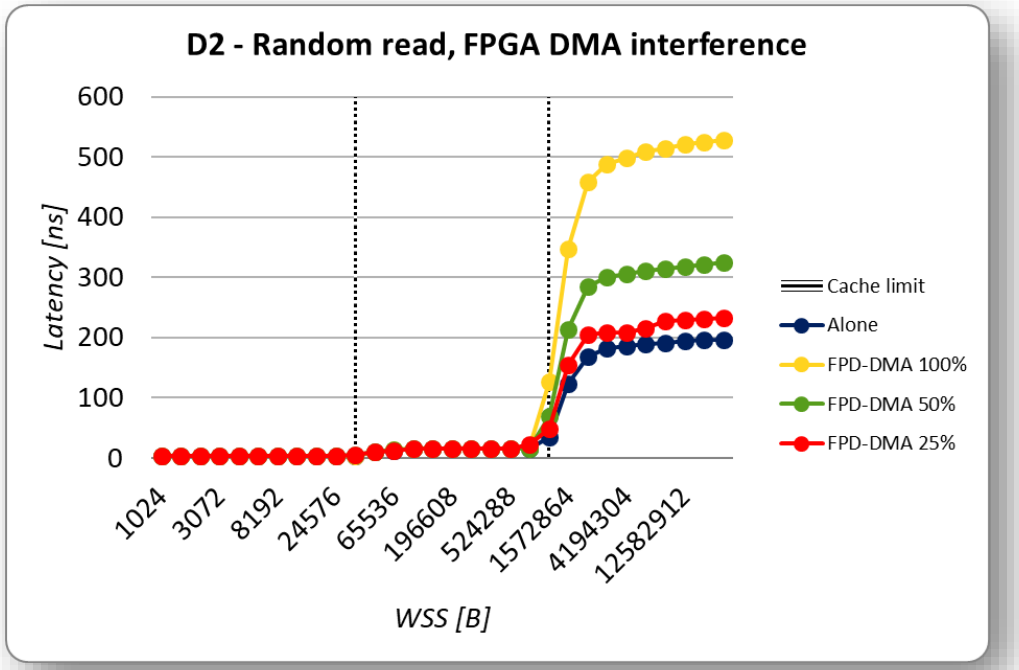
Test 'B' - Tegra X2



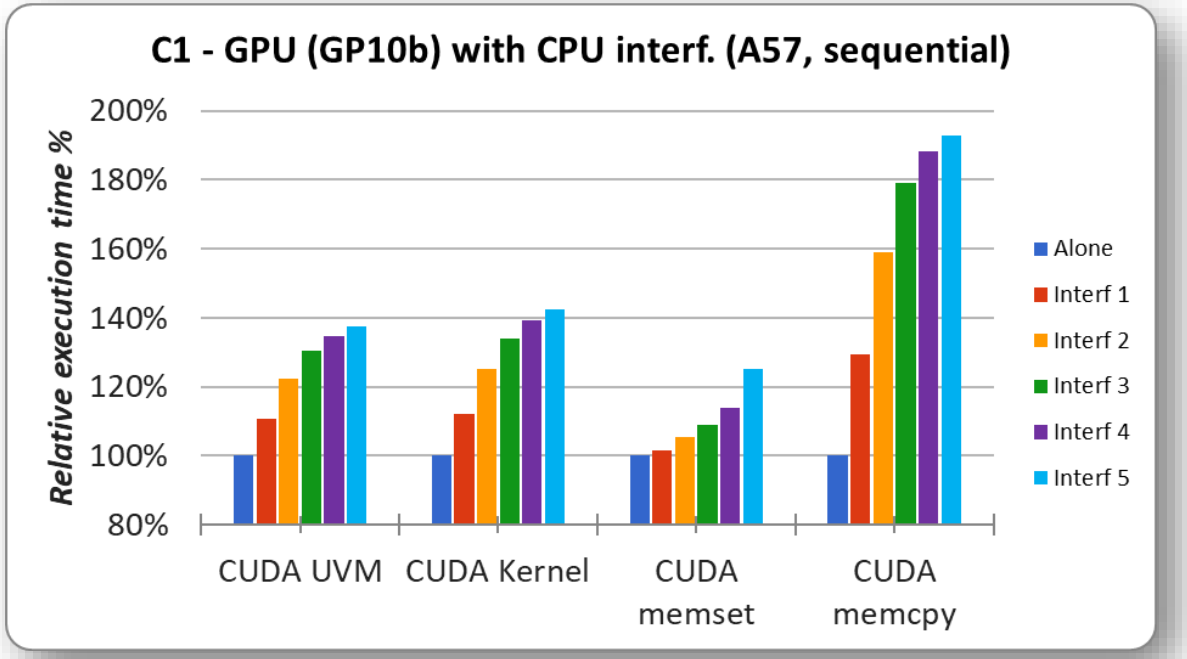


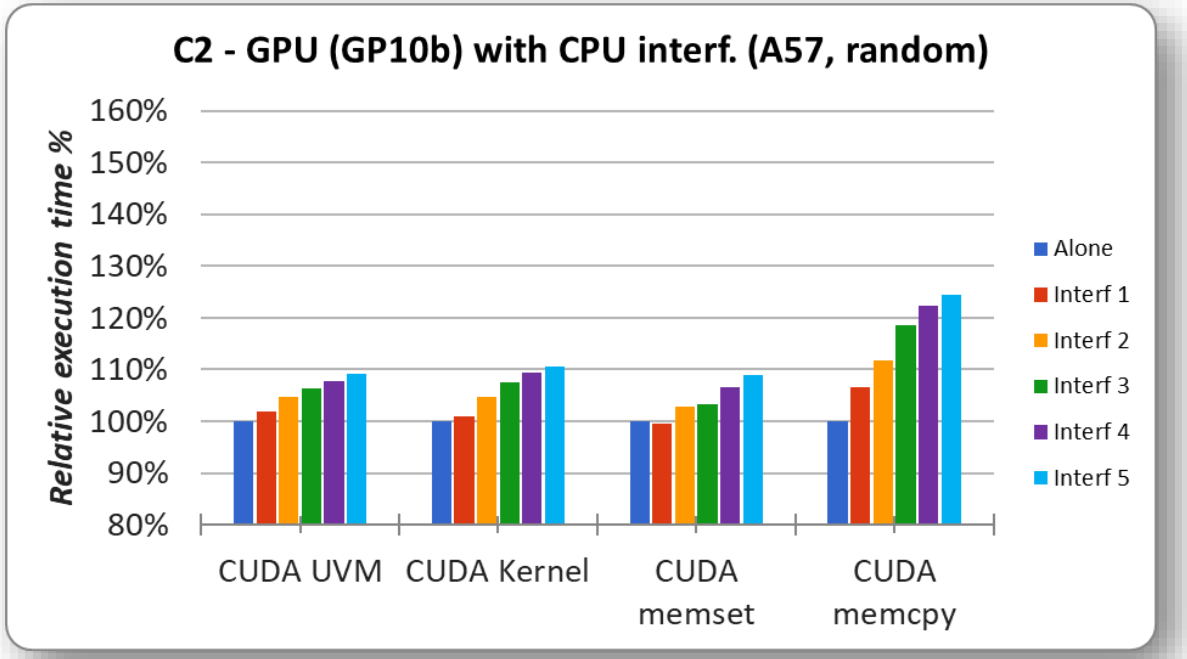
Test 'B' - Xilinx Ultrascale



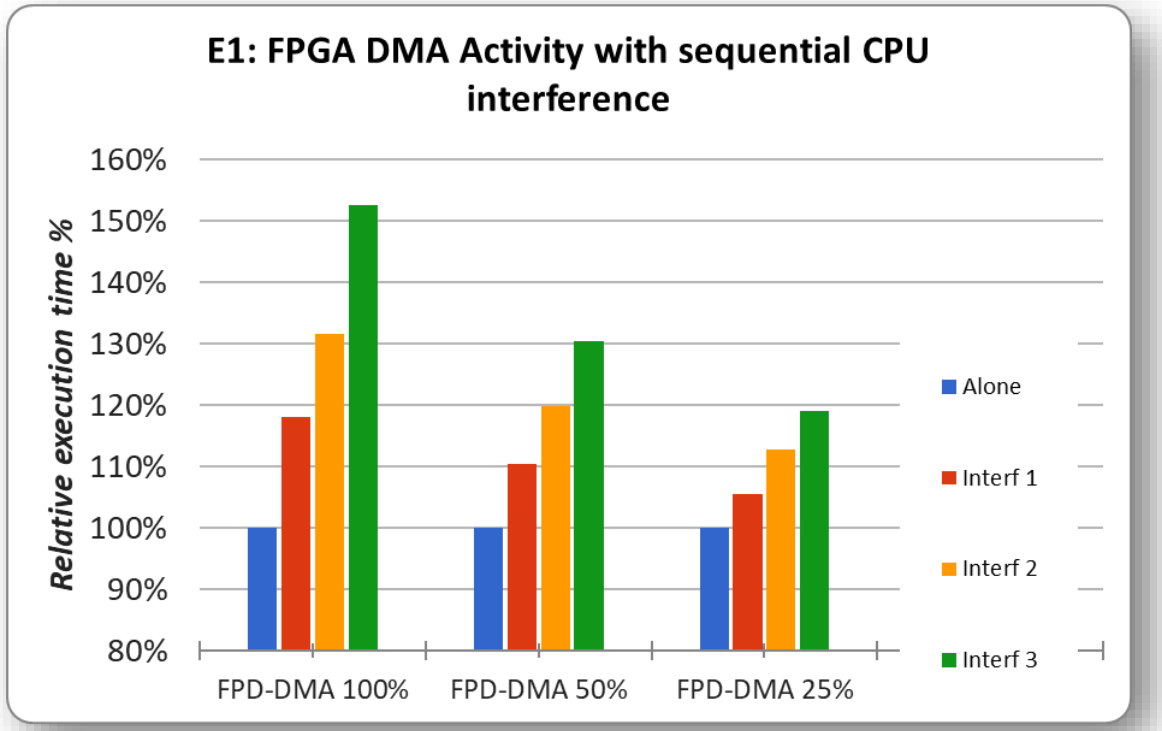


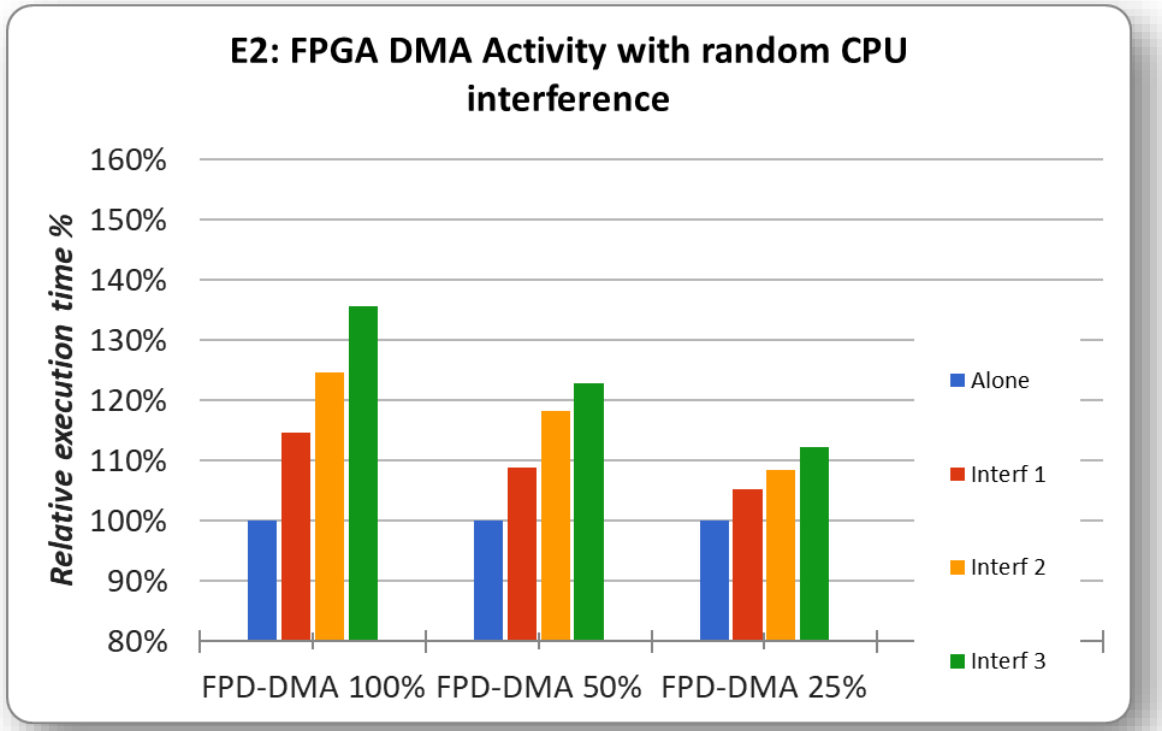
Test 'C' - Tegra X2 – A57





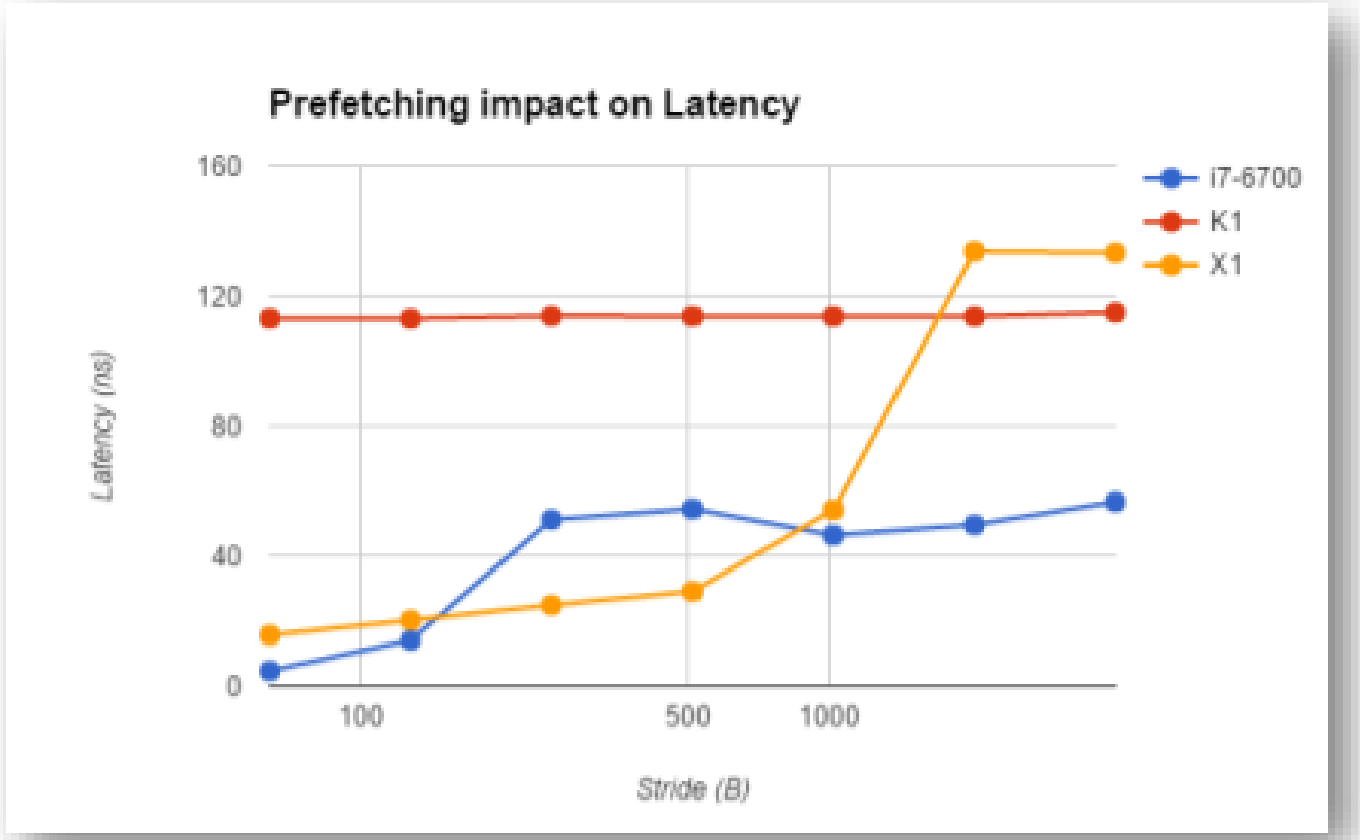
Test 'C' - Xilinx Ultrascale





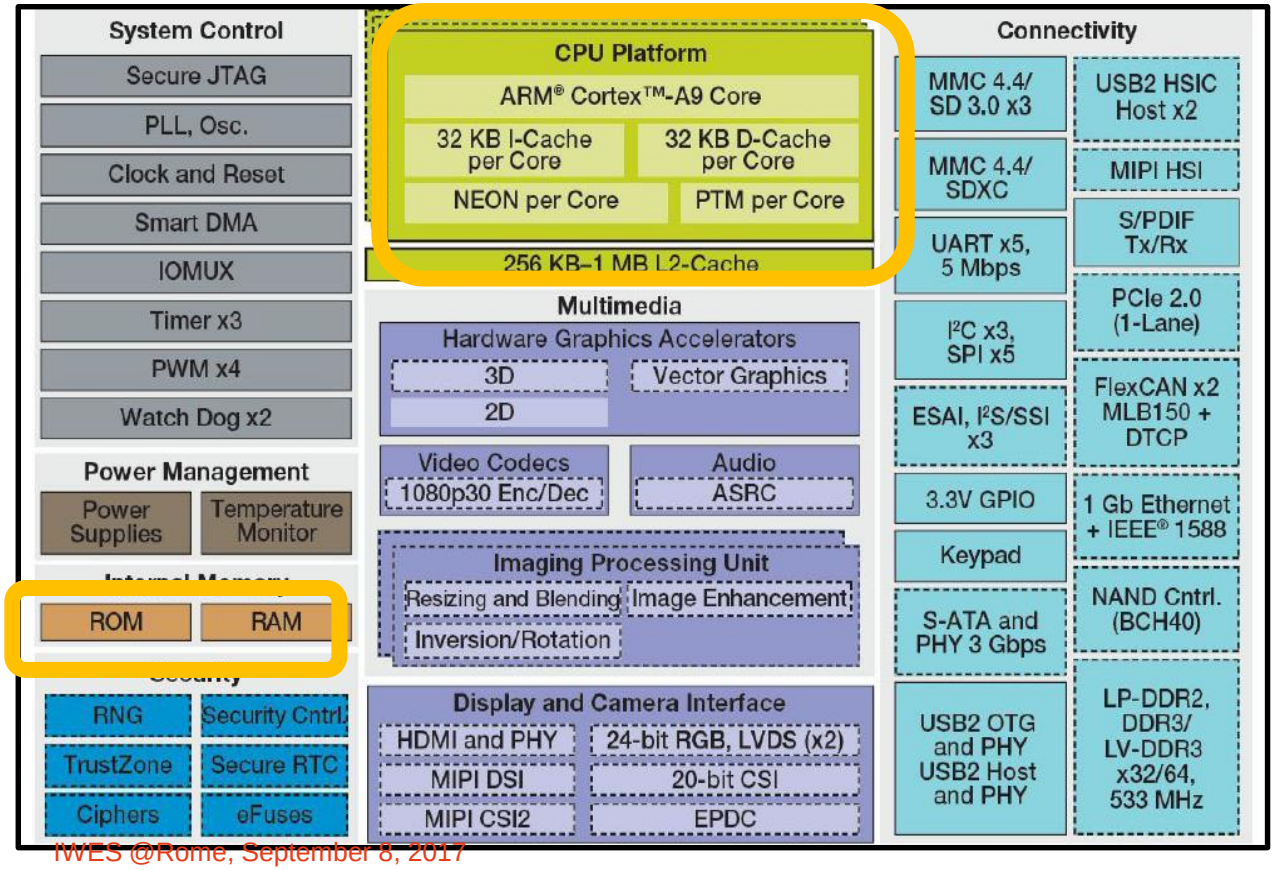
Prefetching

- ✓ Interfere with prefetching mechanism
- ✓ Interfering cores read at increasing strided addresses



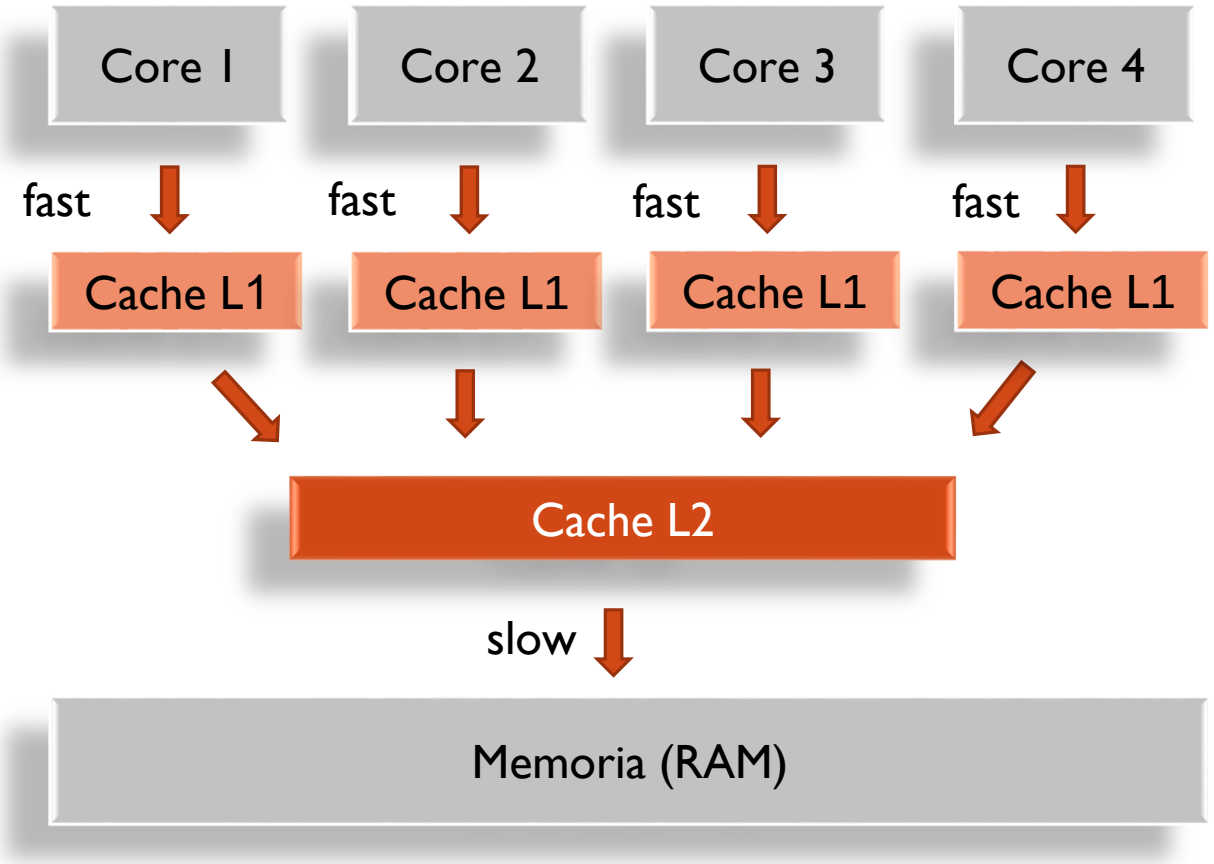
Testbed #2: industrial platform

- ✓ NXP iMX6 from Egicon
  - Components for F1 teams, industrial telescopic arms
  - Credits to Francesco Bellei

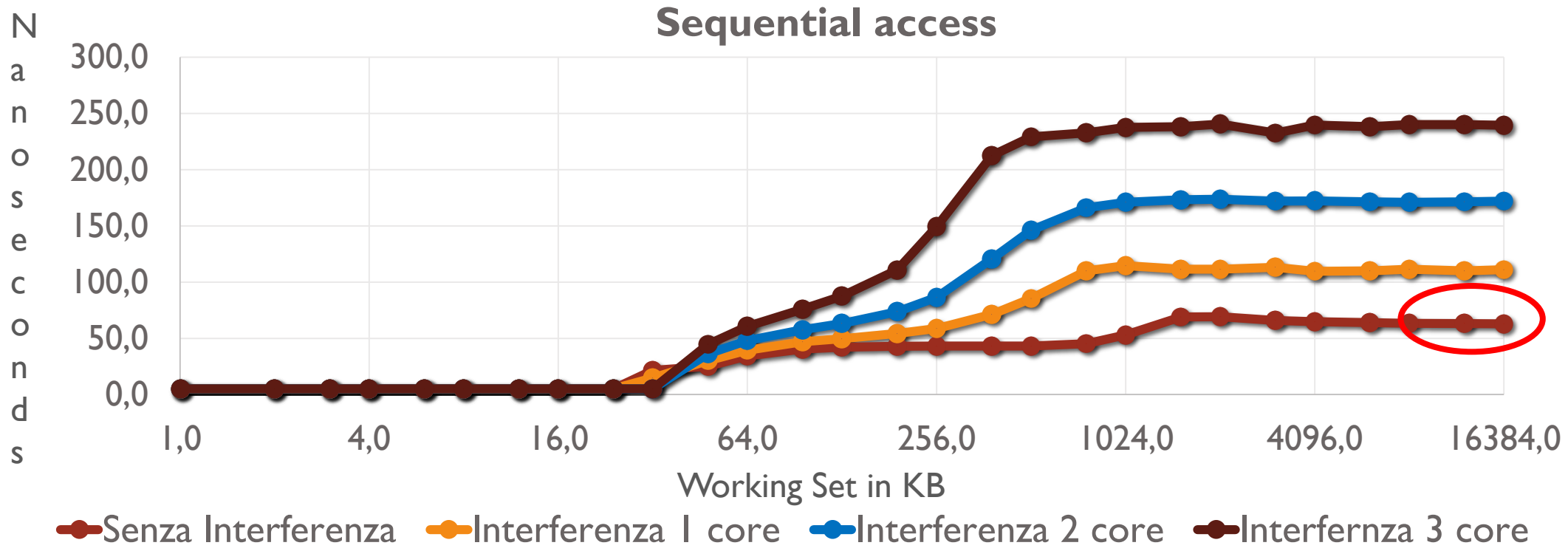


iMX6 mem hierarchy

- ✓ More "traditional"

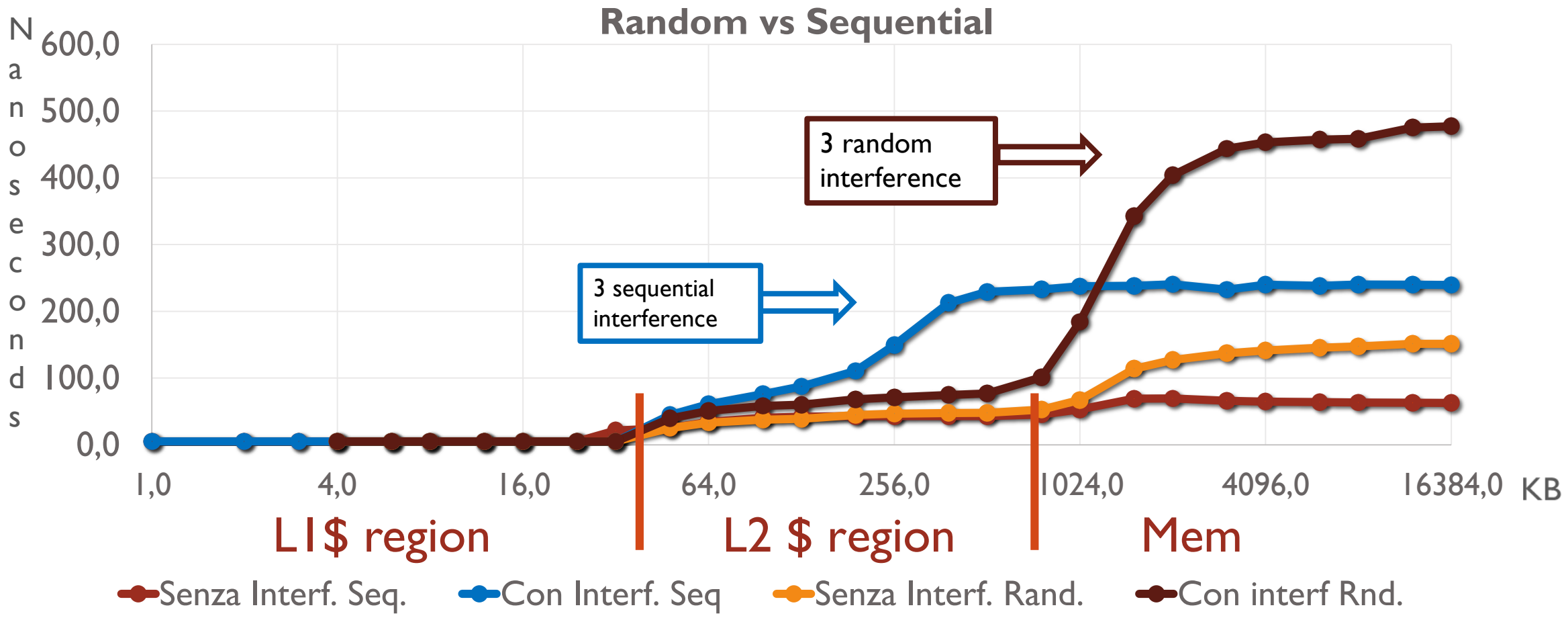


Memory latency - sequential (ns)



$$\text{Max BW} = \frac{\text{Cache Line Size}}{\text{Lat}} = \frac{\text{32 byte}}{\text{62,8 ns}} \cong \text{0,5 GB/s}$$

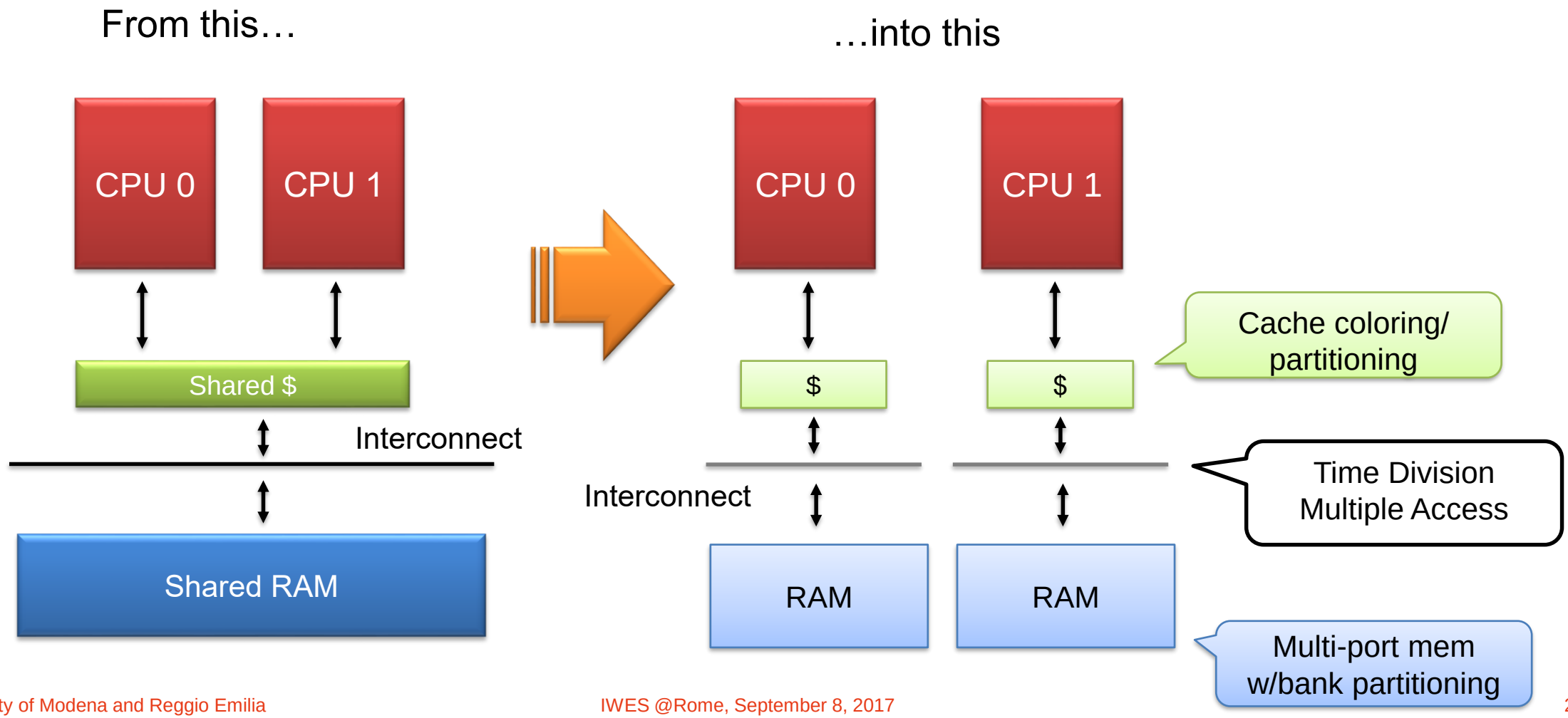
Memory interference impact



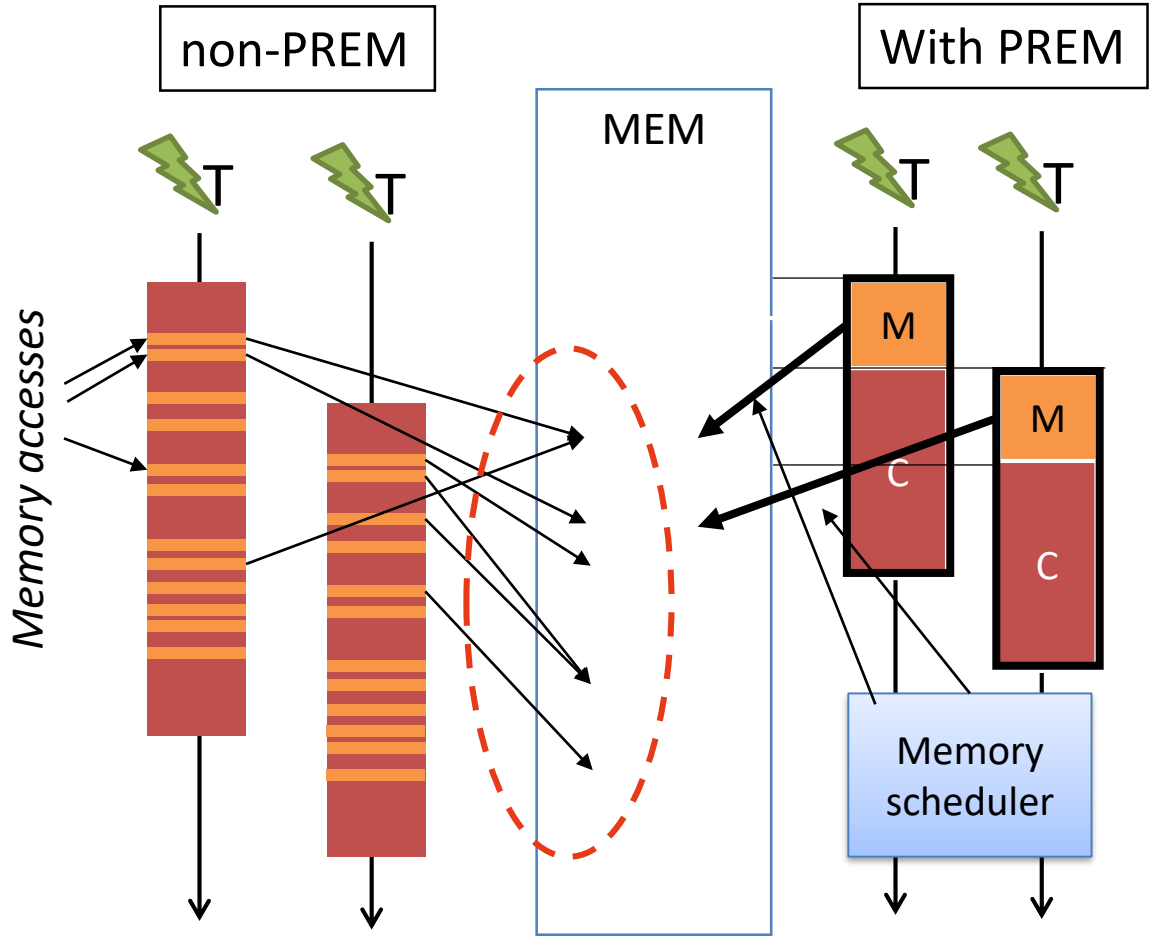
What do we do with this knowledge?

Single-core equivalence

- ✓ A set of techniques to turn the view of the system that software has..



PREM - PRedictable Execution Models



- ✓ Group memory access at the beginning of every software task
- ✓ Co-schedule memory accesses and tasks-to-cores
- ✓ Greatly reduces the complexity of the scheduling problem

…and increases performance

Up to 4x predictable performance on a many-core platform

2015 paper @ RTEST

# Cores/threads	1	2	4	8
No-PREM – Worst (Analytical)	0.026	0.047	0.088	0.170
PREM – Worst (Analytical)	0.010	0.014	0.022	0.038
Speedup	2.6×	3.4×	4.0×	4.5×

OPEN-NEXT

Thank you!

Paolo Burgio
paolo.burgio@unimore.it

HiPeRT Lab High-Performance Real-Time Lab

http://hipert.unimore.it

Backup

Test case A – intra-CPU interference

1. One observed core reads sequentially within a variable sized working set, while other cores are interfering sequentially

2. One observed core reads randomly within a variable sized working set, while other cores are interfering sequentially

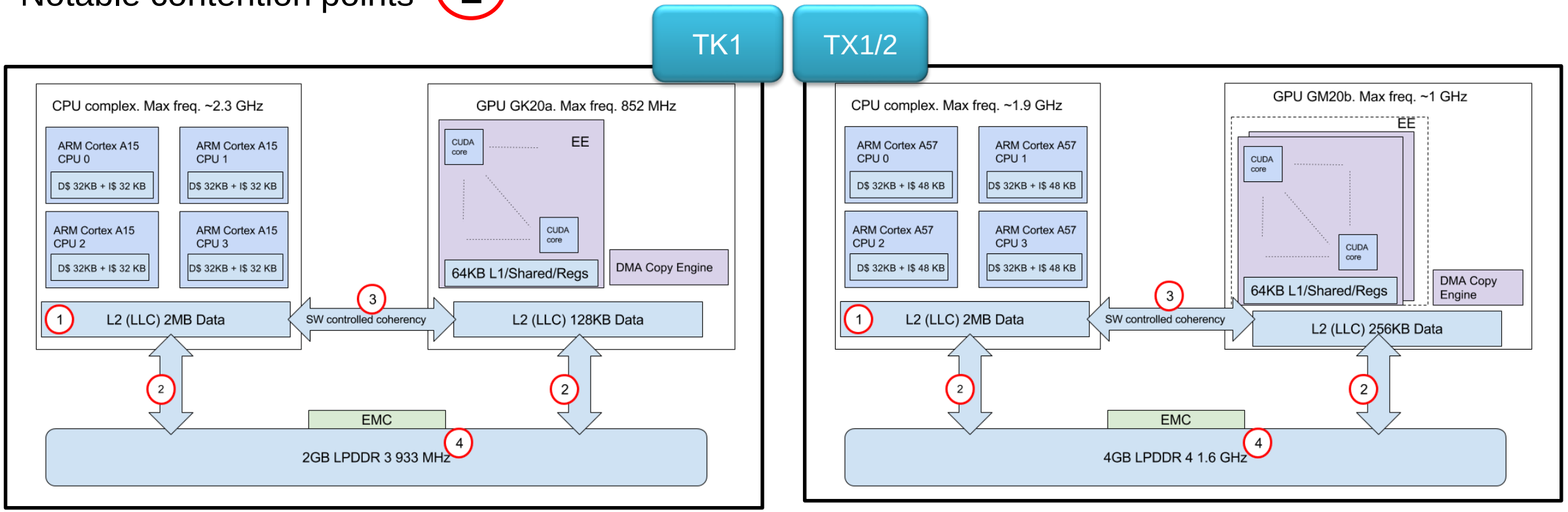
3. One observed core reads sequentially within a variable sized working set, while other cores are interfering randomly

4. One observed core reads randomly within a variable sized working set, while other cores are interfering randomly

NVIDIA Tegra family

- ✓ Shared memory between CPU/GPU complex
  - "Unified Virtual Memory"

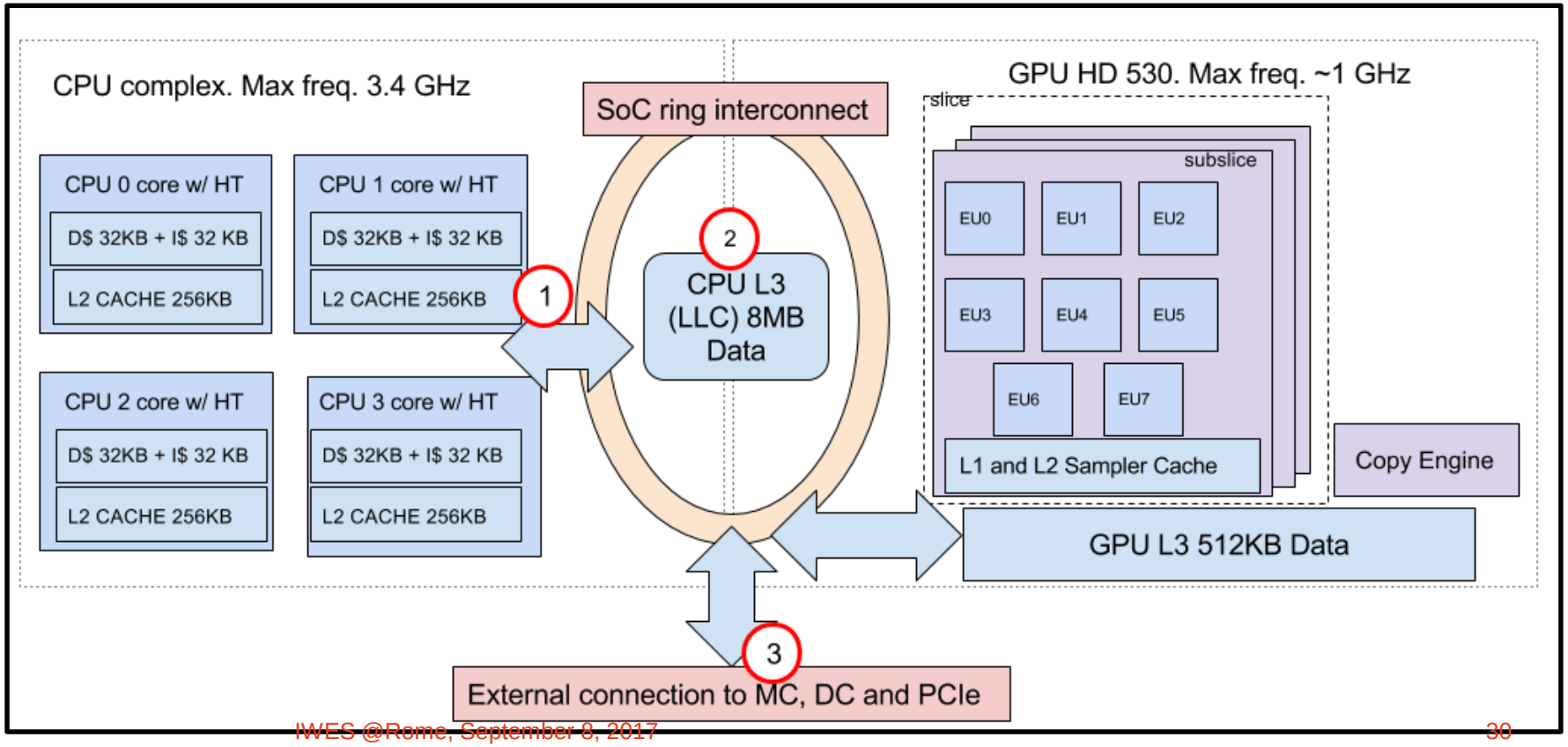
Notable contention points 1



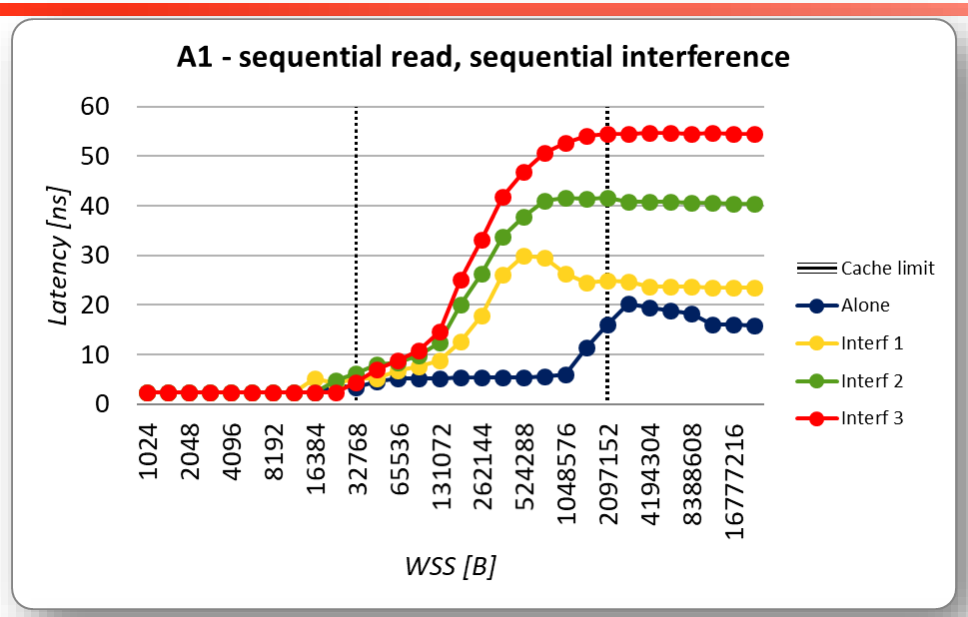
Intel i7-6700 Skylake

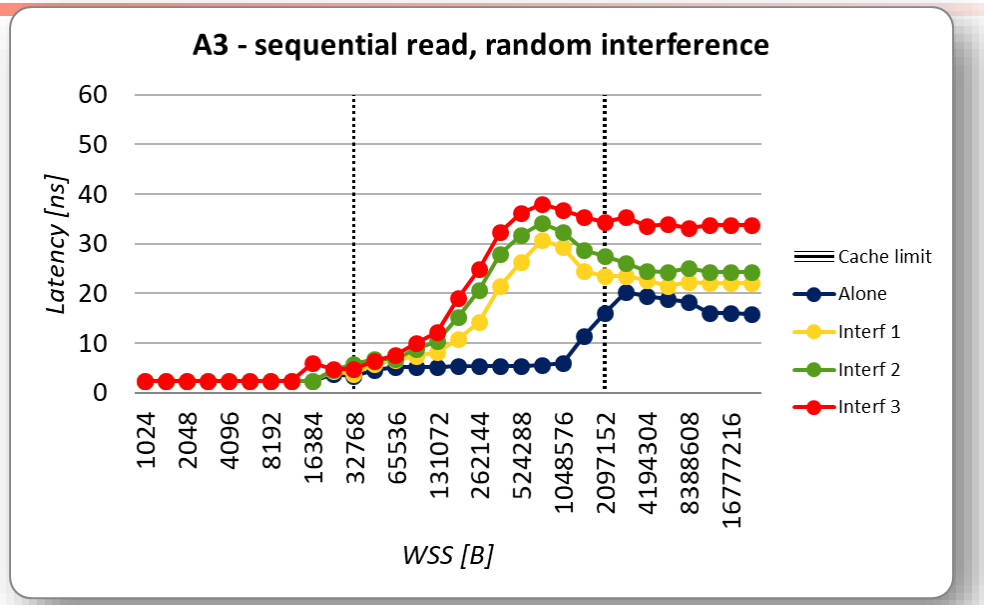
- ✓ x86_64 powerful host + iGPU
  - Sharing L3$, External DRAM…

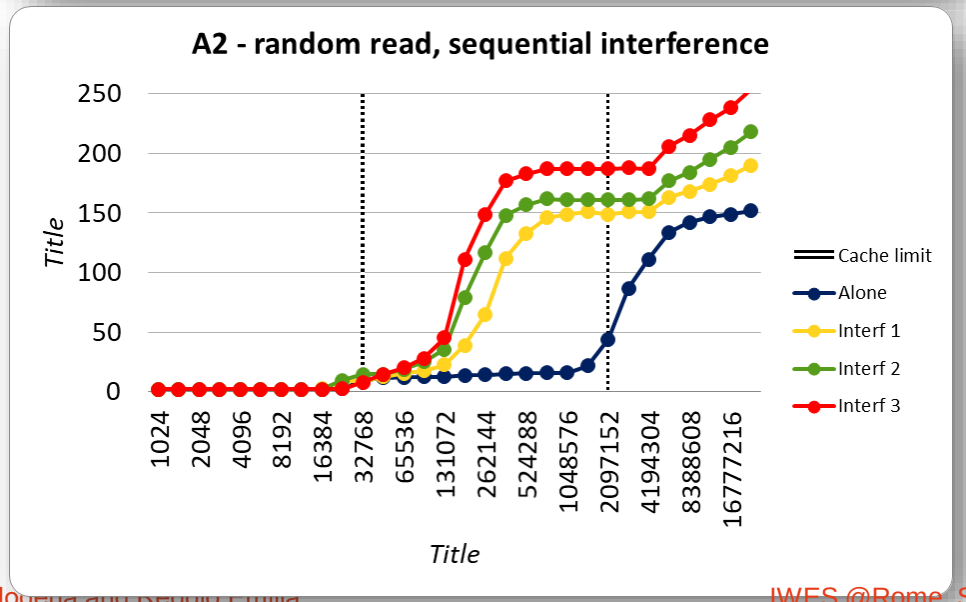
Notable contention points (1)

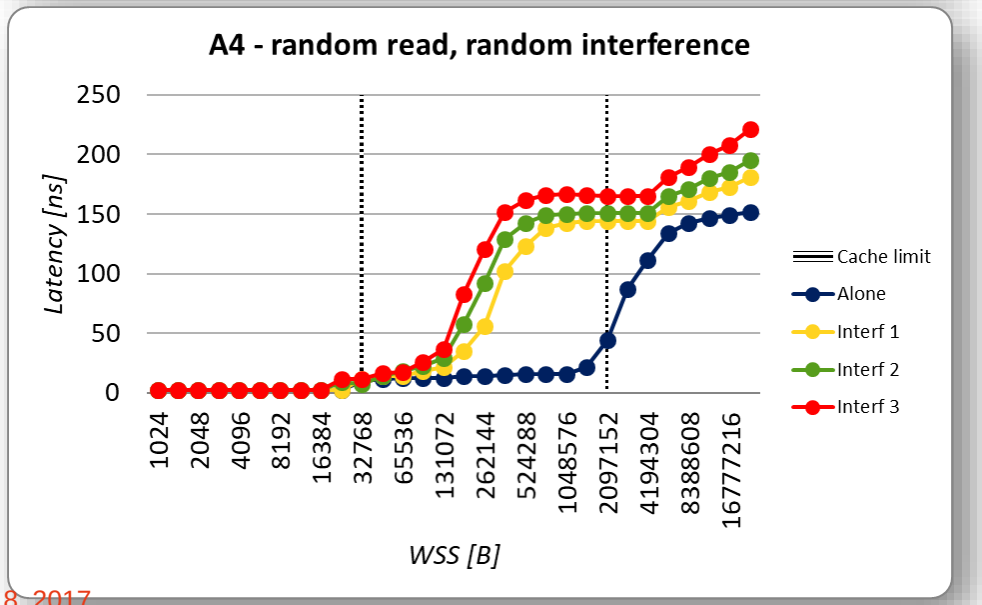


Tegra X1

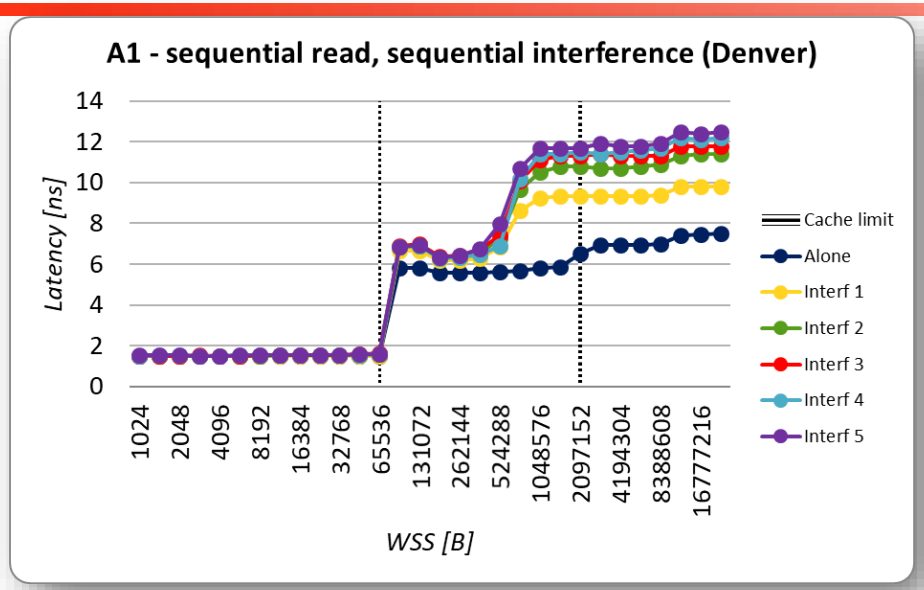

A1 - sequential read, sequential interference
Latency [ns]
60
50
40
30
20
10
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
WSS [B]
Cache limit
Alone
Interf 1
Interf 2
Interf 3

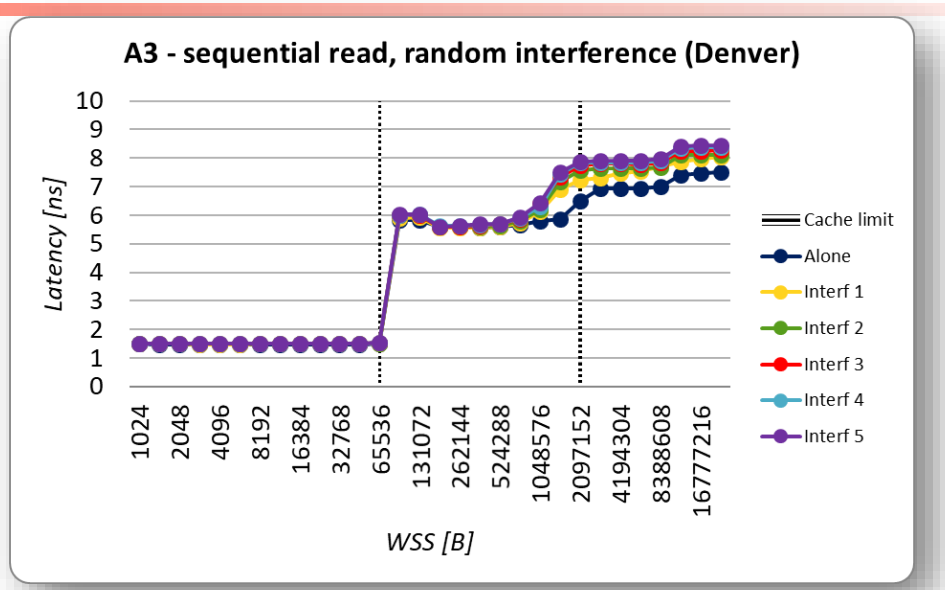

A3 - sequential read, random interference
Latency [ns]
60
50
40
30
20
10
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
WSS [B]
Cache limit
Alone
Interf 1
Interf 2
Interf 3

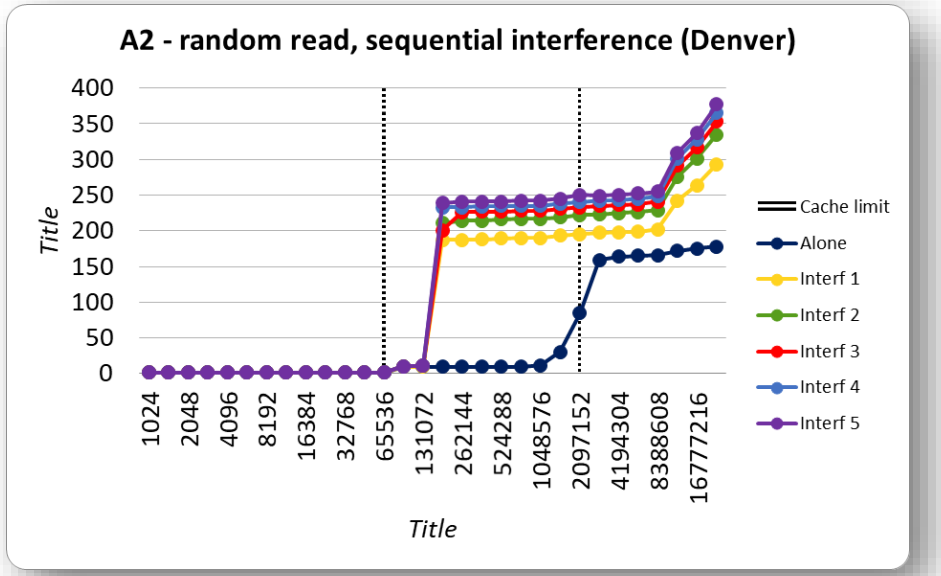

A2 - random read, sequential interference
Title
250
200
150
100
50
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
Title
Cache limit
Alone
Interf 1
Interf 2
Interf 3

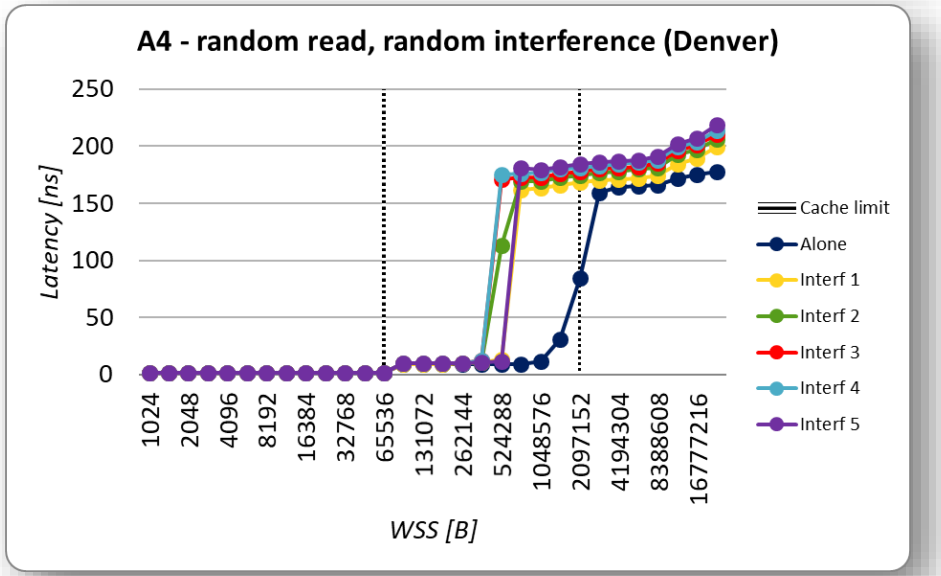

A4 - random read, random interference
Latency [ns]
250
200
150
100
50
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
WSS [B]
Cache limit
Alone
Interf 1
Interf 2
Interf 3

Tegra X2 - Denver







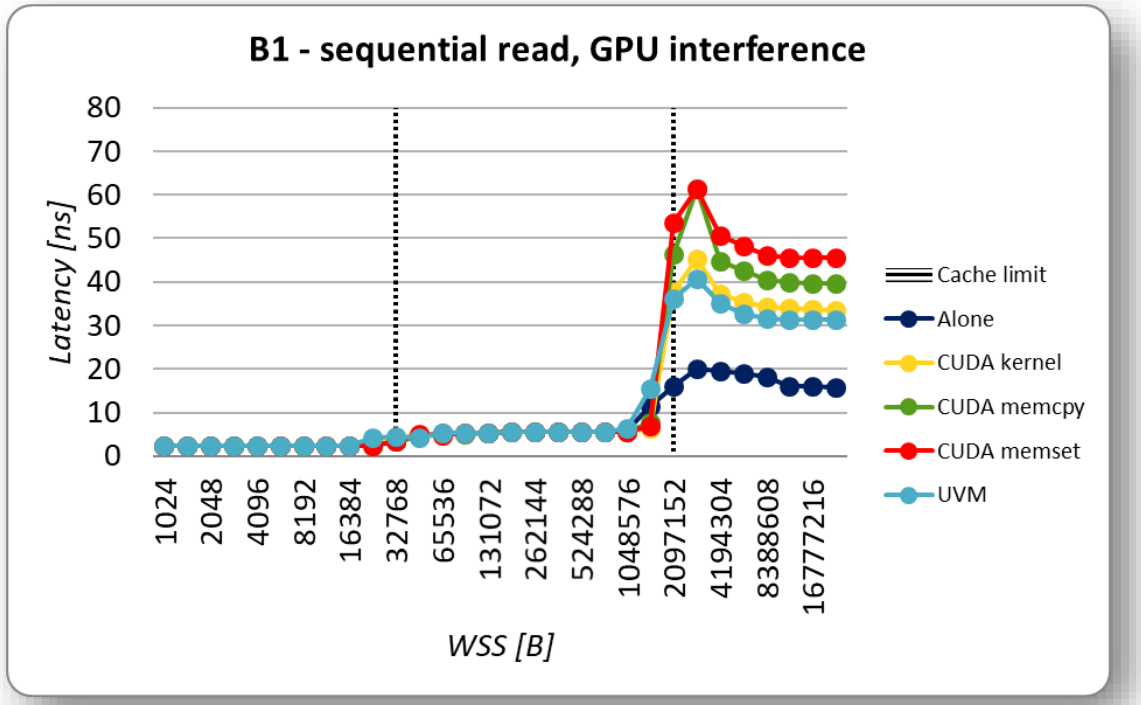


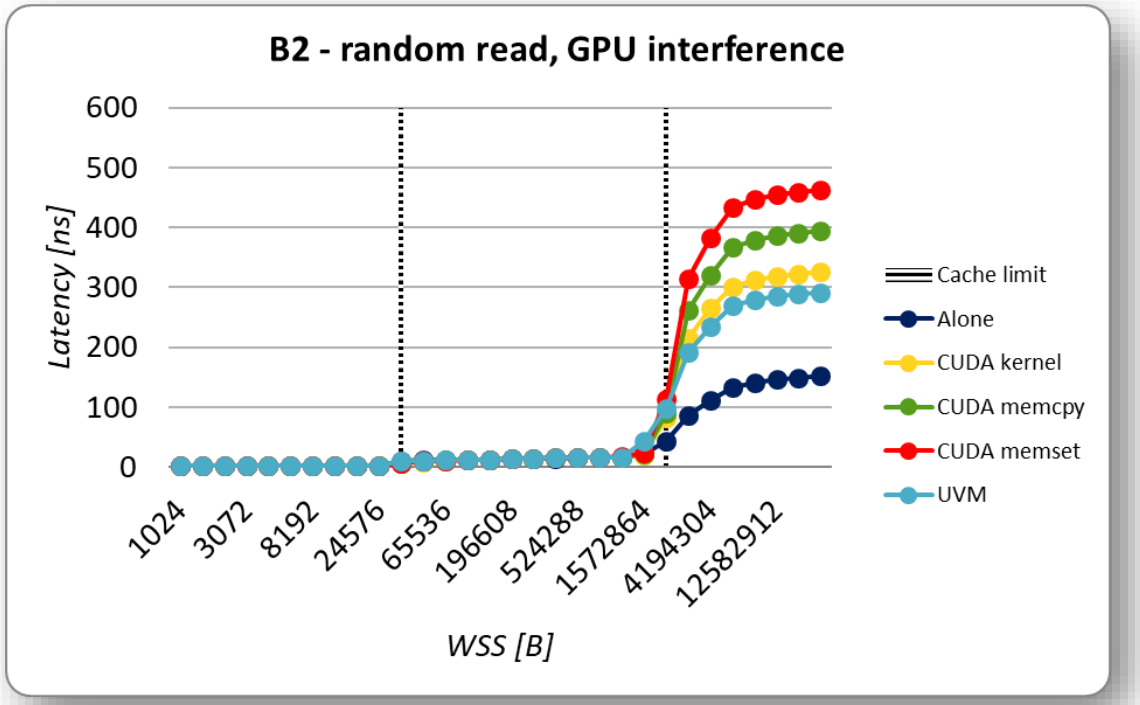
Test case B – iGPU interference on CPU

1. One CPU core reads sequentially within a variable working set, while the GPU accesses memory according to different paradigms:
   - CUDA memcpy
   - CUDA memcpy on UVM
   - CUDA memcpy on pinned mem
   - CUDA memset (0)

2. Same, but CPU core reads randomly

Tegra X1


B1 - sequential read, GPU interference
Latency [ns]
80
70
60
50
40
30
20
10
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
WSS [B]
Cache limit
Alone
CUDA kernel
CUDA memcpy
CUDA memset
UVM

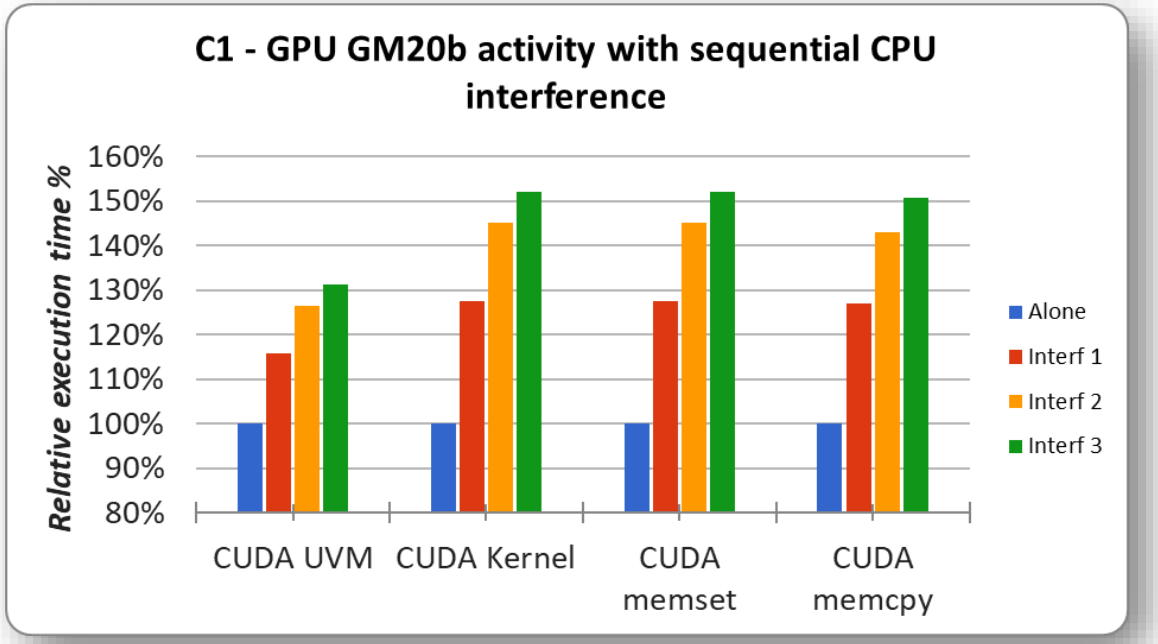

B2 - random read, GPU interference
Latency [ns]
600
500
400
300
200
100
0
1024
3072
8192
24576
65536
196608
524288
1572864
4194304
12582912
WSS [B]
Cache limit
Alone
CUDA kernel
CUDA memcpy
CUDA memset
UVM

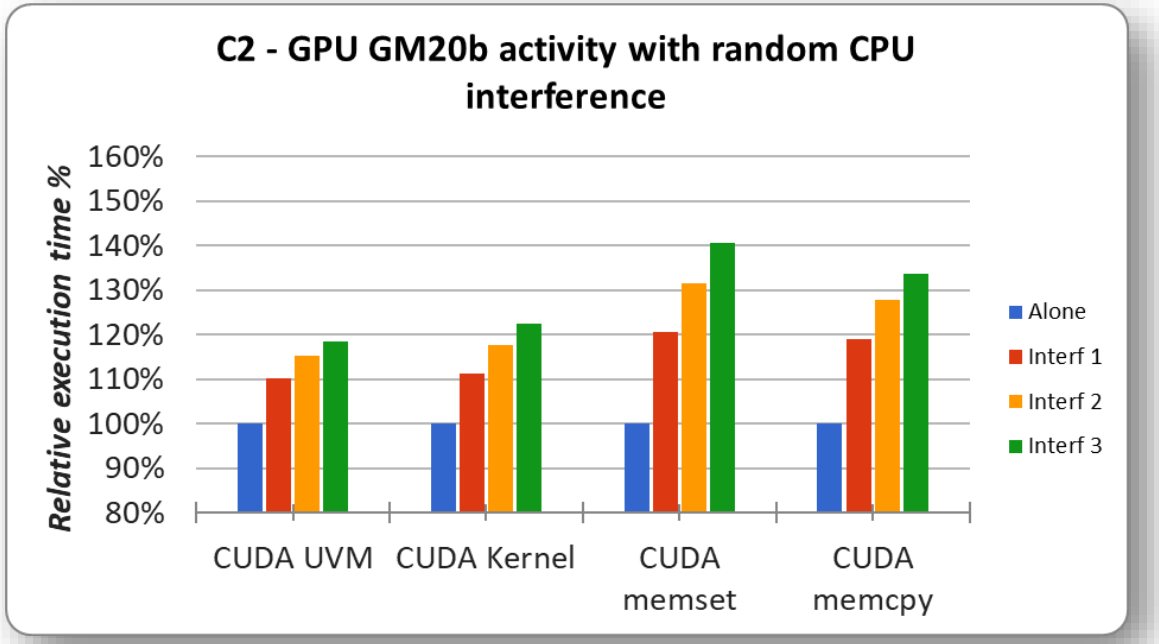
Test case C – CPU interference on iGPU

1. CPU generates sequential interfering mem accesses, while GPU accesses memory according to different paradigms:
   - CUDA memcpy
   - CUDA memcpy on UVM
   - CUDA memcpy on pinned mem
   - CUDA memset (0)

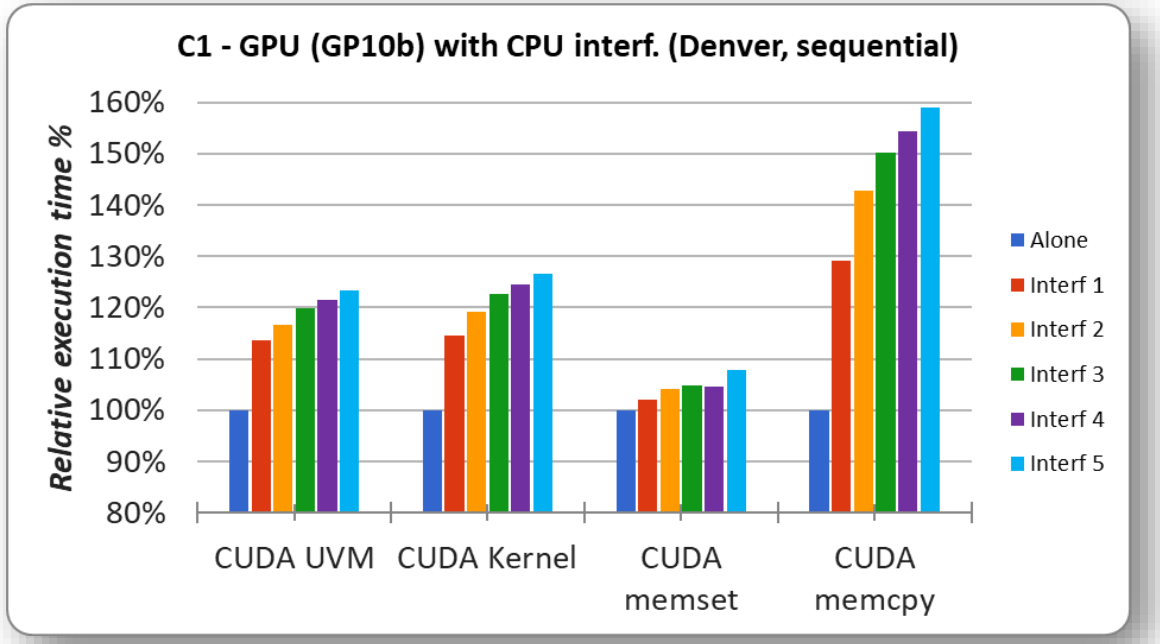
2. Same, but CPU core interference is random

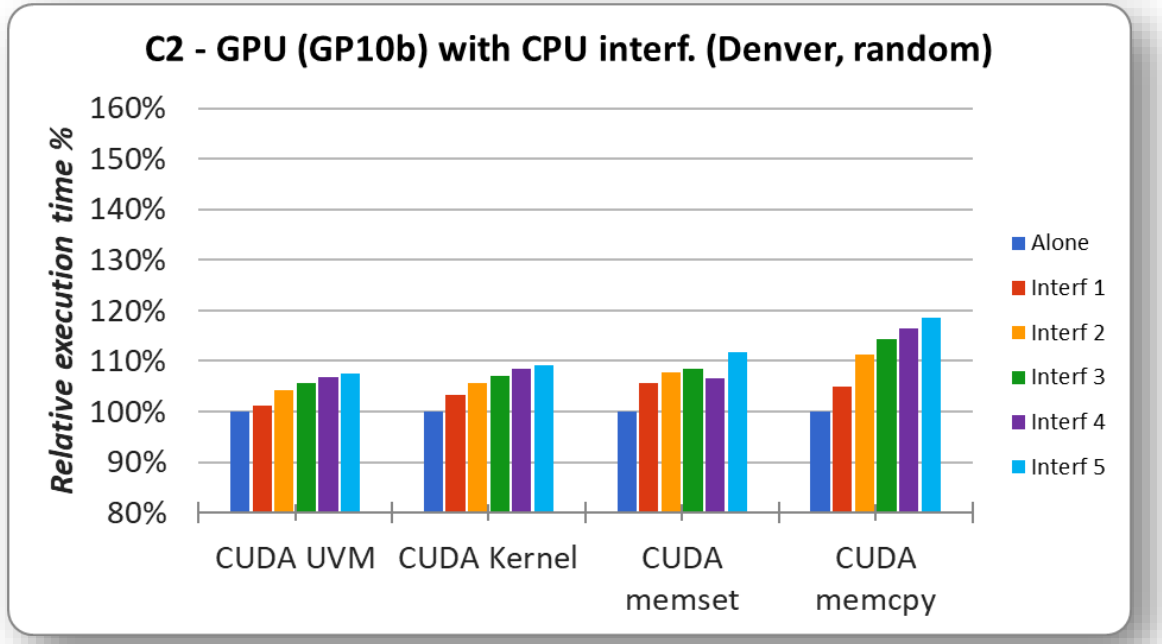
Tegra X1


C1 - GPU GM20b activity with sequential CPU interference
Relative execution time %
160%
150%
140%
130%
120%
110%
100%
90%
80%
CUDA UVM
CUDA Kernel
CUDA memset
CUDA memcpy
Alone
Interf 1
Interf 2
Interf 3


C2 - GPU GM20b activity with random CPU interference
Relative execution time %
160%
150%
140%
130%
120%
110%
100%
90%
80%
CUDA UVM
CUDA Kernel
CUDA memset
CUDA memcpy
Alone
Interf 1
Interf 2
Interf 3

Tegra X2 - Denver





Test 'C' - Intel i7-6700