Memory Interference Characterization between CPU cores and integrated GPUs in Mixed-Criticality Platforms

Roberto Cavicchioli, Nicola Capodieci and Marko Bertogna
University of Modena And Reggio Emilia, Department of Physics, Informatics and Mathematics, Modena, Italy
{name.surname}@unimore.it

***Abstract*—Most of today's mixed criticality platforms feature Systems on Chip (SoC) where a multi-core CPU complex (the host) competes with an integrated Graphic Processor Unit (iGPU, the device) for accessing central memory. The multi-core host and the iGPU share the same memory controller, which has to arbitrate data access to both clients through often undisclosed or non-priority driven mechanisms. Such aspect becomes critical when the iGPU is a high performance massively parallel computing complex potentially able to saturate the available DRAM bandwidth of the considered SoC. The contribution of this paper is to qualitatively analyze and characterize the conflicts due to parallel accesses to main memory by both CPU cores and iGPU, so to motivate the need of novel paradigms for memory centric scheduling mechanisms. We analyzed different well known and commercially available platforms in order to estimate variations in throughput and latencies within various memory access patterns, both at host and device side.**

I. Introduction

Modern Systems on Chips (SoCs) integrate within a single chip substrate many functionalities that are usually fabricated as distinct entities on more traditional designs, such as laptops or desktop computers. Examples of these integrated functionalities commonly featured in embedded boards are represented by the CPU complex (i.e Multi-Core processors), the integrated GPU and their respective memory interfaces. Each core of the CPU complex and the iGPU can process tasks in parallel as they are independent compute units. However, contention may occur at the memory interface level. More specifically, CPU cores and the iGPU might share common cache levels, hence experiencing self-eviction phenomena. In addition, the system memory (usually DRAM) also represents a contented resource for all memory controller clients experiencing cache misses at their Last Level Cache (LLC). It is mandatory to accurately measure the impact of such contention in mixed-criticality platforms, as memory contention poses a significant threat to Worst Case Execution Times (WCETs) of memory bounded applications, as will be shown in this study. The contribution of this paper is to provide accurate measurements of both intra CPU complex memory interference and iGPU activity. We will show how such interference impacts throughput and latency both on the host side and iGPU device.

The ultimate purpose of this analysis is to highlight the need for accurate memory-centric scheduling mechanisms to be set up for guaranteeing prioritized memory accesses to Real-Time critical parts of the system. Special emphasis will be put on the memory traffic originated by the iGPU, as it represents a very popular architectural paradigm for computing massively parallel workloads at impressive performance per Watt ratios [1]. This architectural choice (commonly referred to as General Purpose GPU computing, GPGPU) is one of the reference architectures for future embedded mixed-criticality applications, such as autonomous driving and unmanned aerial vehicle control.

978-1-5090-6505-9/17/$31.00 © 2017 IEEE

In order to maximize the validity of the presented results, we considered different platforms featuring different memory controllers, instruction sets, data bus width, cache hierarchy configurations and programming models:

i NVIDIA Tegra K1 SoC (TK1), using CUDA 6.5 [2] for GPGPU applications;

ii NVIDIA Tegra X1 SoC (TX1), using CUDA 8.0; and

iii Intel i7-6700 SoC featuring HD 530 Integrated GPU, using OpenCL 2.0 [3].

This paper is organized as follows: section II presents an up-to-date brief review on previous studies regarding memory contention in integrated devices. Section III includes a thorough description of the platforms that are being characterized. Section IV describes the experimental framework and the obtained results. Section VI concludes the paper.

II. Related Work

As soon as the processors industry introduced the concept of Multi-Core CPUs, memory contention was observed to become a potential bottleneck, mostly due to bus contention and cache pollution phenomena [4]. Later studies [5], [6], [7] successfully identified methodologies to bound the delay times of Real Time tasks due to memory access contention. Memory arbitration mechanisms have been recently proposed to decrease the impact of memory contention in the design of critical applications, implementing co-scheduling mechanism of memory and processing bandwidth by multiple host cores [8]. Examples of such memory arbitration mechanisms are represented by MEMGUARD [9], BWLOCK [10] and the PREM execution model [11]. The previously cited contributions are instrumental to our work, because they aim at understanding the location of the contention points within the memory hierarchy with respect to the platforms we used in our tests. Moreover, such contention points might differ according to the analyzed COTS (Commercial Off The Shelf)

systems, hence the need to qualitatively characterize the recently commercialized platforms we used in our experiments (detailed in the next section). Since the integrated GPU of the observed SoCs are most likely used to perform computations with strict Real Time requirements, it is important to estimate the impact of unregulated CPU memory accesses during the execution of Real Time GPU applications. It is also trivial to understand that the iGPU, in a mixed-criticality system can execute non critical applications during the same time windows in which one or more CPU cores are executing Real Time tasks, hence the need to observe the impact of unregulated GPU activity towards CPU memory access latencies. CPU and GPU co-run interference is a topic that was not treated in the previously cited works, but was briefly explored in [12]. In a simulated environment, the authors of this latter contribution highlighted a decrease of instruction per cycle w.r.t. CPU activity together with a decrease in memory bandwidth on the GPU side. In our contribution, we therefore promise more accurate measurements on commercially available SoCs.

III. SoCs specifications and contention points

In this section, we provide the necessary details on the platforms we selected for our analysis. This is instrumental for understanding where memory access contention might happen for each SoC. The preliminary investigation of plausible contention points will help us better understanding the experimental results to identify future solutions [13]. The delays due to the private memory of the iGPU is not considered in this study, focusing only on the memory shared with the CPU complex. Private memory contention may be a significant issue for discrete GPUs that feature a significant number of execution units (i.e., Streaming Multiprocessors in NVIDIA GPUs). Today's iGPUs, however, are much smaller than their discrete counterparts. As a consequence, parallel kernels are typically executed one at a time, without generating contention at private memory level.

Another important abstract component at iGPU side is the Copy Engine (CE), i.e., a DMA used to bring data from CPU to GPU memory space. In discrete devices, this basically translates into copying memory from system DRAM through PCIe towards the on-board RAM of the graphics adapter (Video RAM, VRAM). In case of embedded platforms with shared system DRAM, using the CE basically means duplicating the same buffer twice on the same memory device. Both CUDA and OpenCL programming models specify alternatives to the CE approach to avoid explicit memory transfers and unnecessary buffer replications, such as CUDA UVM (Unified Virtual Memory [14]) and OpenCL 2.0 SVM (Shared Virtual Memory [15]). However, these approaches introduce CPU-iGPU memory coherency problems when accessing the same shared memory buffer, so that avoiding copy engines does not necessarily lead to performance improvements[1] For this reason, we will characterize the contention originated in both CE- and non-CE-based models.

A. NVIDIA Tegra K1

The NVIDIA Tegra K1 [16] is an hybrid SoC featured in the NVIDIA Jetson Development board. It is the first mobile processor to have the same advanced features and architecture as a modern desktop GPU while still using the low power draw of a mobile chip (365 GFlops single precision peak performance at $<$ 11 W). The most relevant parts of this platform and notable contention points are visible in Figure 1(a).

The K1 SoC consists of a quad-core 2.3GHz ARM Cortex-A15 CPU (32kb I-cache + 32kb D-cache L1 per core, 2MB L2 cache common to all cores); ARM A15 belongs to the ARMv7-A family of RISC processors and features a 32bit architecture. Although not shown in the picture, an ARM Cortex A15 shadow-core is also present for power saving policies. We consider all cores be clocked at their maximum operative frequencies. A single CPU core can utilize the maximum bandwidth available for the whole CPU complex which amounts to almost 3.5 GB/s for sequential reading operations. The iGPU is a Kepler generation "GK20a" with 192 CUDA cores grouped in a single Streaming Multi-processor (SM). As visible from Figure 1(a), the compute pipeline of an NVIDIA GPU includes engines responsible for computations (Execution Engine, EE) and engines responsible for high bandwidth DMA memory transfers (Copy Engine, CE), as explained in [2] and [17]. In the TK1 case, the EE is composed of a single SM which is able to access system memory in case of L2 cache misses. The CE is meant to replicate CPU visible buffers to another area of the same system DRAM that is only visible to the GPU device. It does that by exploiting high bandwidth DMA transfers that can reach up to 12 GB/s out of the 14 GB/s theoretical bandwidth available by the system DRAM. Therefore, the GPU alone is able to saturate the available system DRAM bandwidth. Specifically regarding the system DRAM, the K1 features 2GB of LPDDR3 64bit SDRAM working at (maximum) 933MHz.

The first contention point is represented by the LLC for the CPU complex. This cache is a 16-way associative cache with a line size of 64B. Contention may happen when more than one core fills the L2 cache evicting pre-existing cache lines used by other cores. Another contention point for LLC is indicated as point 3 in Figure 1(a), and it is due to coherency mechanisms between CPU and iGPU when they share the same address space. In the GK20a iGPU, such coherence is taken care in a unclear and undisclosed software mechanisms by the (NVIDIA proprietary) GPU driver. Hardware cache coherence mechanisms take place only at CPU complex level.

The remaining contention points (2 and 4 in Figure 1(a)) are represented by memory bus and EMC (Embedded Memory Controller) for accessing the underlying DRAM banks. Such contention is of uttermost importance as it is caused by parallel memory accesses of single CPU cores and the iGPU. We refer to [18] for finer grained discussions regarding the effect of bank parallelism in DRAM devices.

B. NVIDIA Tegra X1

The NVIDIA Tegra X1 [19] is a hybrid System on Module (SoM) featured in the newest NVIDIA Jetson Development board. It is the first mobile processor to feature a chip powerful enough to sustain the visual computing load for autonomous and assisted driving applications, still presenting a contained power consumption (1 TFlops single precision peak performance drawing from 6 to 15 W). Also for this platform, the

[1] Visible as results of these experiments https://github.com/Sarahild/CudaMemoryExperiments/tree/master/MemCpyExperiments

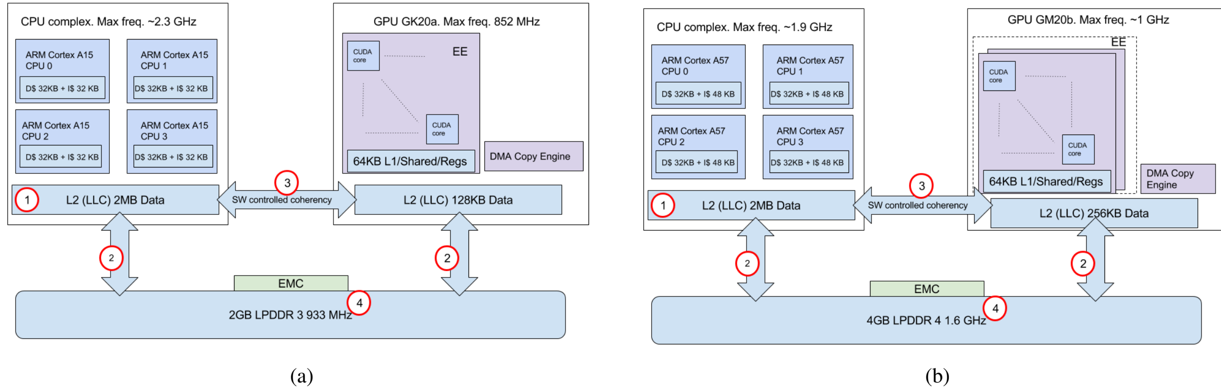


Fig. 1: A simplified overview of the Tegra K1 (a) and X1 (b) SoCs, with notable memory access contention points numbered from 1 to 4: (1) contention on L2 cache shared by the 4 cores; (2) contention on bus to central memory from different clients; (3) coherency protocol on LLC; (4) access arbitration on main memory controller.

most relevant components and notable contention points are shown in Figure 1(b).

The X1 CPU and GPU complex consists of a quad-core 1.9GHz ARM Cortex-A57 CPU (48kb I-cache + 32kb D-cache L1 per core, 2MB L2 cache common to all cores); ARM A57 belongs to the ARMv8-A family of RISC processors and features a 64bit architecture. Even if not visible in the Figure, the CPU complex features a big.LITTLE architecture, with also four ARM Cortex A53 little cores for power saving purposes. As for the K1, we will not analyze the performance of this board under power saving regimes. A single CPU core can utilize the maximum bandwidth available for the whole CPU complex, which amounts to almost 4.5 GB/s for sequential reading operations. The iGPU is a Maxwell second generation "GM20b" with 256 CUDA cores grouped in two Streaming Multi-processors (SMs). The L2 is twice the size of its Kepler based predecessor. The EE and CE can access central memory with a maximum bandwidth close to 20 GB/s. As with the K1, also this high performance iGPU can saturate the whole DRAM bandwidth. The system DRAM consists of 4GB of LPDDR4 64bit SDRAM working at (maximum) 1.6GHz, reaching a peak ideal bandwidth of 25.6 GB/s. With relation to the contention points, there are no substantial differences with the K1.

C. Intel i7-6700

The Intel i7-6700 processor presents noticeable differences between the two boards described in the previous paragraphs. This SoC features a quad-core CPU complex with Hyper-Threading (HT) technology built above the well known x86 64 bit CISC architecture. Technical points of interests for this platform are depicted in Figure 2.

This specific processor belongs to the Skylake (6th generation) of Intel CPUs. Such processors are common for desktop or laptop configurations, with a SoC power consumption around 65W, higher than in the previously described boards. Another peculiar difference refers to the cache hierarchy, where, unlike the ARM based design, L2 cache is not shared among physical cores. This is an architectural choice that Intel adopts since the Nehalem generation (released in late 2008). L2 cache is shared between two logical cores of the same physical computing unit. According to Intel HyperThreading's proprietary design [20], a number of logical cores that is twice the number of the physical cores is made available to the operating system. Hence, 8 threads can be scheduled to run concurrently. This is allowed by exploiting aggressive out of order execution policies and other optimization techniques aimed at improving the instruction level parallelism of a single physical core. However, HyperThreading is also known to increase the competition for shared memory resources [21]. Figure 2 shows the interconnections between the iGPU, each CPU core, and the interface to the Memory Controller (MC), Display Controller (DC) and PCI express bus (PCIe). In our experimental setup, the i7 interfaces with 16GB of 64bit DDR4 DRAM clocked at 2133 MHz, allowing the CPU complex to reach a theoretical bandwidth of 34 GB/s, resulting in a measured bandwidth of 29 GB/s using Intel MLC[2] for sequential reads. Both the CPU complex and the iGPU reach the memory controller through the SoC ring interconnection fabric depicted in Figure 2. The iGPU in this SoC is an Intel HD 530 belonging to the 9Gen Intel graphic architecture [22]. Differently from the NVIDIA architectures, the compute pipeline is composed of an execution engine composed of a *slice*, divided into three sub-slices. Each of these partitions relies on 8 Execution Units (EU from 0 to 7 as shown in Figure 2). Another difference w.r.t. the previous solutions is represented by the cache hierarchy, where the LLC is shared between the CPU and the GPU, with related coherency HW mechanisms. A cache miss in the GPU L3 will imply using the SoC ring interconnect to access CPU L3, and, if needed, the external memory controller. Moreover, as detailed in [22], L1 and L2 cache are read-only caches only used for the graphics pipeline, not representing a significant contention point. The HD 530 also has an abstraction of a Copy Engine used to replicate data from a CPU-only visible address space to a space accessible by the iGPU. However, thanks to the CPU-GPU shared cache level, the best practices of using the iGPU in such designs is to exploit the unified memory architecture (SVM model in OpenCL 2.0).

[2]Intel Memory Latency Checker v. 3.1 available at https://software.intel.com/en-us/articles/intelr-memory-latency-checker

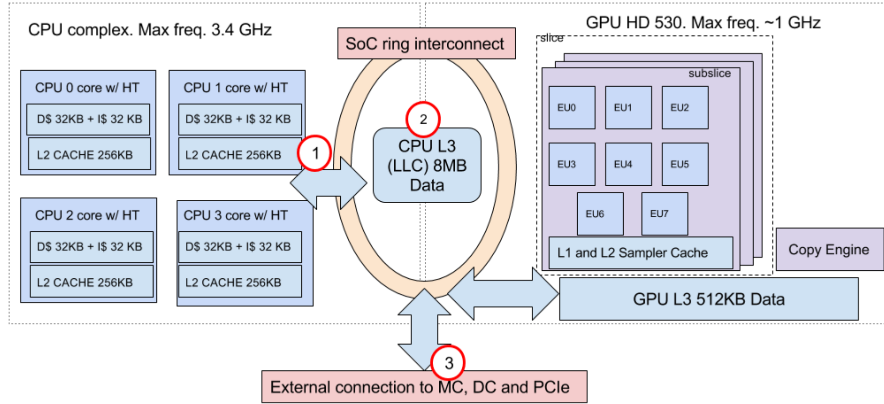


Fig. 2: A simplified overview of the Intel i7-6700 Skylake with notable contention points: (1) represents contention on shared L3 cache among all CPU cores; (2) contention on the shared CPU-iGPU LLC; (3) parallel access to system DRAM by all the presented actors. (1) and (2) also include coherency overhead for CPU and iGPU traffic.

IV. EXPERIMENTAL SETTING

We are interested in measuring the effect of memory interference as indicated by the previously detected contention points, both at CPU and iGPU side. Latencies variations of single memory accesses are highlighted. Idle latencies of single CPU cores are first measured, before adding the interference from the other cores, and then also from the GPU (CE, EE and unified memory models). On the iGPU side, we measure the variation of execution times of tasks using the CE, EE and unified memory models. Once again, the case with no interference is first measured, before then adding the interference from the different cores within the CPU complex. Another key parameter to vary was the memory access pattern for the CPU complex. For both the measured core and the interfering cores, we considered either sequential memory accesses, exploiting the hardware pre-fetching abilities of the CPU, or random accesses. On the GPU side, only sequential accesses were profiled, since (i) CE memory transfers are inherently sequential, and (ii) typical memory access patterns on the EE side are also sequential to allow the lock-step mechanism to work at the maximum efficiency. Different measuring tools have been used in our experiments. The Intel MLC (Memory Latency Checker v3.1) was used for the i7-6700 to calculate the available theoretical and practical bandwidths. For all platforms, we used LMBench [23] and a custom-made program to measure latencies with (i) sequential, (ii) variable, and (iii) random stride reads, with all possible interfering memory access patterns, both in read and write.

Summarizing, the following tests have been performed:

Test Case A: intra CPU complex interference.

A1: the observed core reads sequentially within a variable sized working set (henceforth *sequential read*), while the other cores are interfering sequentially (henceforth, *sequential interference*). Each iteration performs a *memcpy* of 100MB, so to involve every element within the CPU complex memory hierarchy.

A2: the observed core reads with a random stride within a variable sized working set (henceforth *random reads*), while the interfering cores performs sequential interference.

A3: the observed core reads sequentially, while the other cores iteratively read 64B with a random stride within a 128MB array (*random interference*). The array size has been chosen to statistically prevent fetching already cached data.

A4: the observed core performs random reads, while the interfering cores performs random interference.

Test Case B: iGPU interference to CPU.

B1: the observed core reads sequentially, while the GPU accesses memory according to different paradigms:

- launching a copy kernel[3] between GPU buffers;

- launching a copy kernel involving unified memory located buffers (CUDA UVM for X1 and K1, OpenCL SVM for the i7);

- copying a buffer by means of the copy engine, using pinned memory; and

- zeroing a device buffer.

B2: the observed core reads randomly, while the GPU activity is the same as B1.

Test Case C (CPU interference to iGPU).

C1: the GPU accesses memory according to the different paradigms detailed in test case B, while the host cores perform the interfering patterns described in A1.

C2: same as above, but host cores perform the interfering patterns described in A3.

V. EXPERIMENTAL RESULTS

In this section, we present the results of the test cases identified above for each considered platform.

A. *Latencies on Tegra K1*

Test Case A measures the impact on latencies due to shared L2 and shared memory bus to system DRAM (points 1, 2 and 4

[3] A copy kernel is a GPU program that performs an element-wise data copy between two buffers.

in Figure 1(a)), without accounting for GPU activity. Working Set Size (WSS) goes from 1kB to 25MB.

Figure 3 (a) and (b) show the average latency for sequentially accessing one word (32 bit) varying the WSS, for subtests A1 and A2, i.e., with sequential interference. In all test cases, memory latencies with a random access pattern are only slightly higher than with a sequential access, showing limited burst capabilities. A significant performance degradation is noticeable for WSS larger than L1 size in all interfered cases (*Interf 1* to *3*, in both graphs). The maximum gap between the interfered and non-interfered cases can be noticed for WSS slightly comparable to L2 size. This is explained by the cache evictions performed by interfering cores, reducing the useful cache blocks available for the measured core. After the L2 boundary, the measured latency converges to a delay proportional to the number of interfering cores, both for sequential and random reads. With three interfering cores, a performance loss of about 72% and 84% is measured with respect to the non-interfered setting, for sequential and random reads respectively.

A similar behavior is obtained for random interfering patterns, as shown in Figure 3 (d) and (e) for subtests A3 and A4. In this case, delays are more weakly proportional to the number of interfering cores. Both for sequential and random reads, the latency introduced by interfering cores is smaller than in the previous cases: with three interfering cores, a performance loss of about 60% and 75% is measured with respect to the non-interfered setting, for sequential and random reads respectively.

Summarizing the results of Test Case A, the interfering sources are perfectly consistent with the contention points identified in Figure 1(a). The bandwidth available to each core of the CPU complex when accessing system DRAM is fairly distributed among cores in the sequentially interfered scenario, while a less fair distribution happens in the randomly interfered one. The largest delay is measured for random reads and sequentially interfering tasks. This can be motivated by the re-ordering mechanism implemented at EMC level, which tends to favor sequential accesses with respect to random ones.

Results for Test Case B are shown in Figure 3 (c) and (f), measuring the interference from the iGPU to the CPU complex (points 2,3 and 4 in Figure 1(a)). Only one core is observed, while the remaining ones are inactive. Memory access patterns for the observed CPU core are sequential (subtest B1) and random (subtest B2), while only sequential interference is considered from the iGPU.

In Figure 3 (c) and (f), we notice that differently from Test Case A, performance degradation mostly takes place after the CPU L2 boundaries. This happens because separate caches are adopted at CPU and iGPU level. The highest performance degradation is observed with the *memset* operation, where the iGPU causes the observed core to experience a 426% higher delay. Smaller degradations are measured for the *memcpy* (377%) and UVM/CUDA kernel operations (220%). This shows that *memset* operations saturate to a larger extent the memory bandwidth, while UVM and CUDA kernel operations are not able to fully exploit the available memory bandwidth from the iGPU side. Interestingly, UVM and CUDA kernel memory transfers impose the same interference to the observed core. The cache coherency traffic due to UVM is not noticeable in this setting. This is because the CPU accesses different memory region than those accessed at iGPU side, without triggering coherency traffic. Implementing a test case that highlights the effect of coherency traffic between CPU and iGPU would require using recent versions of the NVIDIA CUDA proprietary profiler (nvprof)[4], which allows a more accurate analysis of page faults handling with UVM. For space constraints, we postpone UVM related tests to a future work.

Results of Test Case C are shown in Figure 3 (g) and (h), measuring the interference from CPU cores to iGPU tasks. As can be noticed comparing the histograms, sequentially interfering tasks cause more visible delays to iGPU activities than with random interfering patterns. Performance deterioration ranges from a minimum of 7% (CUDA UVM with one interfering core) to 35% (CUDA *memset* with 2 or 3 interfering sequential cores) in the sequential case, and from 3 to 12% in the random case. This is consistent with the observation made about the re-ordering features of the EMC engine, prioritizing sequential accesses over random strides.

Only a limited increase in the execution times of GPU tasks is measured when increasing the number of interfering CPU cores, showing that a single CPU core is able to almost fully utilize the available memory bandwidth. A larger increase with the number of cores is noticeable in the *memset* and *memcpy* cases, because these commands exploit the memory bus in both read and write directions, allowing a higher bandwidth utilization.

B. Latencies on Tegra X1

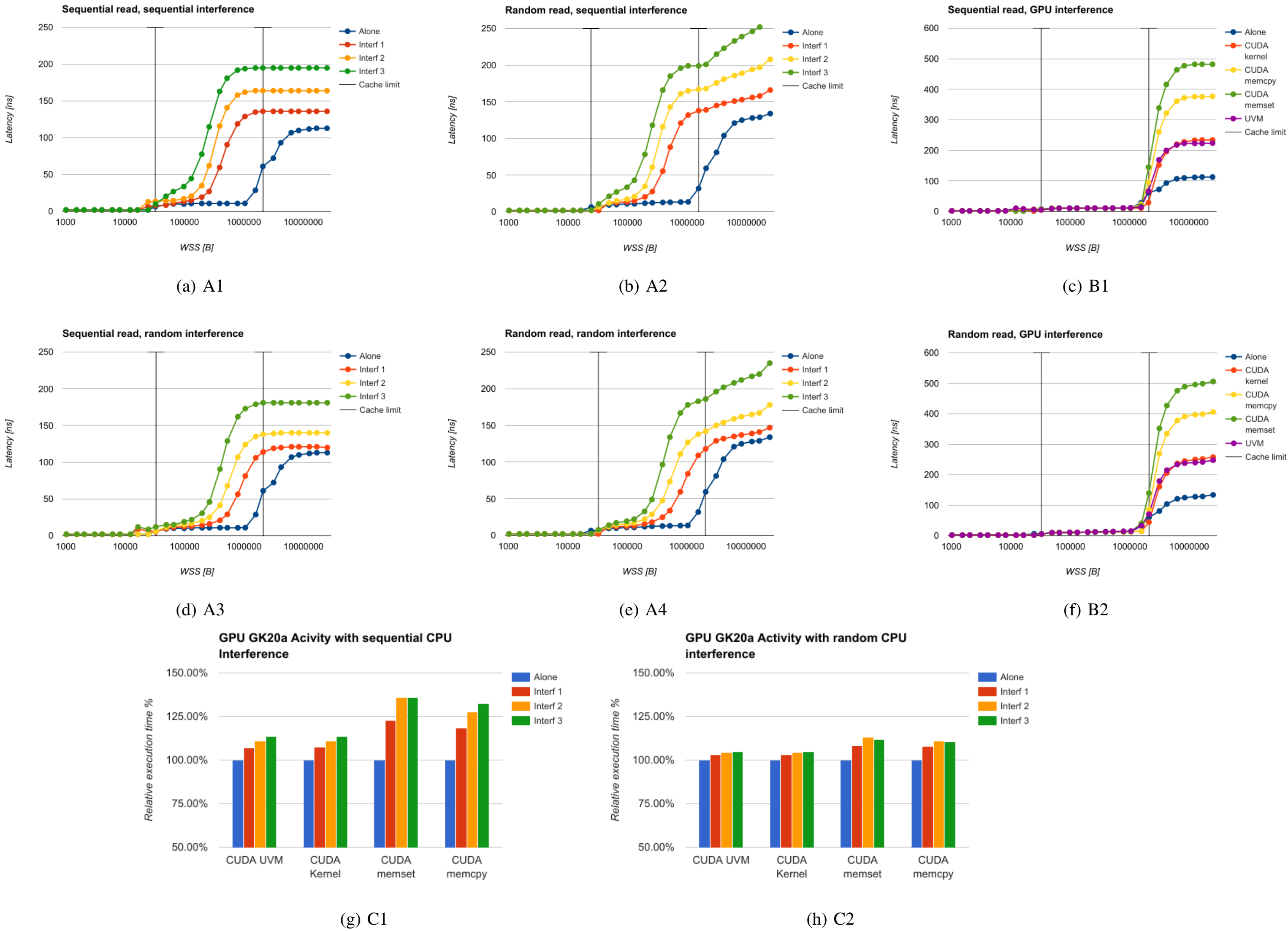
The second NVIDIA-based platform share the same architectural paradigm of the K1 platform. Thus, the same memory contention points highlighted in Test Cases A, B and C are involved. Results are shown in in Figure 4.

The first noticeable difference w.r.t. Tegra K1 is related to the much better performance for sequential reads (up to 15ns and 150ns for the X1 and K1, respectively). Random read delays are instead comparable for both platforms. This shows significantly improved burst transfer features in the X1 architecture, allowing a tenfold improvement with respect to random access patterns (as we will detail also in Section V-D).

The contention points identified in section III-B are confirmed in the experiments. Results for Test Case A shown in insets (a) and (b) present a trend comparable to that shown for the K1, with large performance deterioration occurring with working set sizes between L1 and L2, and a linear degradation increasing the number of cores. Latency spikes are visible in subtests A1 and A3 (insets (a) and (d)) around L2 boundaries. The origin of such spikes, that were not observed in the K1 case, is not clear, and it will be investigated in future works.

Results for Test Case B are shown in Figure 4 (c) and (f). The effect of unregulated iGPU activity is similar to the K1 case, with two notable differences: (i) the presence of the spikes in the sequential case, as in the previous experiment; and (ii) a much larger performance gap between the two extreme

[4]For more info see https://devblogs.nvidia.com/parallelforall/beyond-gpu-memory-limits-unified-memory-pascal/



(a) A1 (b) A2 (c) B1

(d) A3 (e) A4 (f) B2

(g) C1 (h) C2

Fig. 3: Test results for Tegra K1: WSS [Byte] (log) vs. Latencies [ns]. Vertical lines in (a)-(f) correspond to L1 and L2 size.

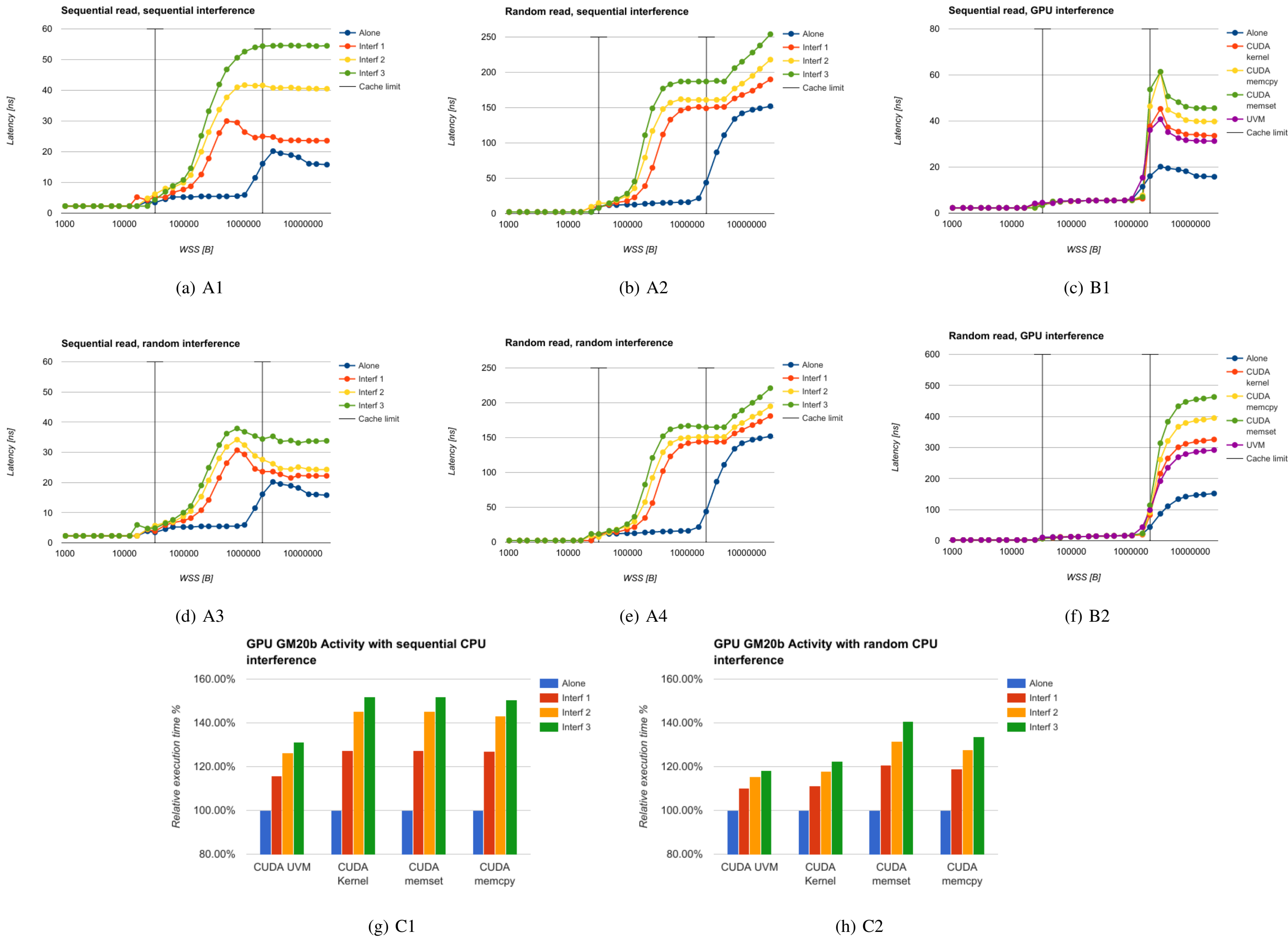
situations, i.e., sequential non-interfered reads (15-18 ns) and random reads with CUDA memset interference (463 ns).

In Figure 4 (g) and (h), we see the effect of CPU activity over iGPU related tasks (Test Case C). The relative impact on iGPU execution times is larger than the one observed with the K1. In case of three sequentially interfering cores, CUDA memset and memcpy execution times increase by 52% (it was 30% in the K1 case). This is mostly due to the enhanced memory bandwidth of the CPU cluster in the X1 platform, together with the more aggressive prefetching mechanisms in the A57's, as will be shown in section V-D. With random interference, the deterioration is somewhat smaller (12%), although not as small as it was in the K1 case (24%), confirming the largest memory bandwidth utilized by the CPU cluster in the X1.

C. Latencies on i7-6700

Interpreting the results on the i7 is much more difficult than the previous solutions: the x86 architecture is substantially more complex than the RISC based architectures analyzed so far. This implies having much more aggressive and counter intuitive optimization strategies for prefetching, speculative execution and other mechanisms (e.g., System Mode Management [24] and non trivial cache mapping policies [25]) that are not analyzed in this paper but that might affect the results.

Still, some meaningful aspects can be identified. Experiments for Test Case A are depicted in Figure 5. The latencies with no interference are significantly smaller than in the previous SoCs. For sequential reads, increasing the working set size beyond L3 only adds a few ns with respect to the latency measured within L1 (from 1 to 4/5 ns). For random reads, a significant performance decrease (about 11x) can be noticed for WSS beyond L3. Increasing the number of



(a) A1 (b) A2 (c) B1

(d) A3 (e) A4 (f) B2

(g) C1 (h) C2

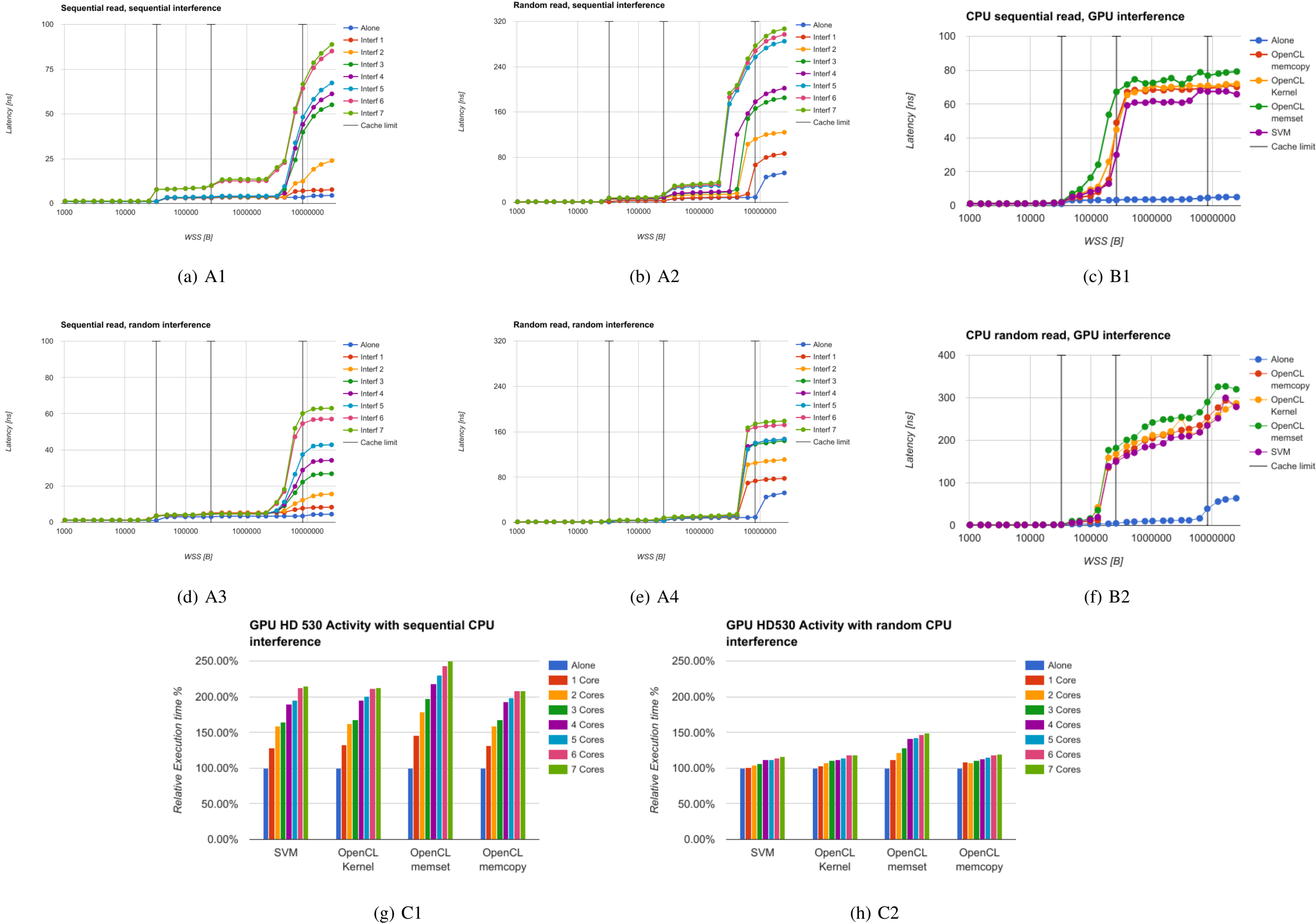
Fig. 4: Test results for Tegra X1: WSS [Byte] (log) vs. Latencies [ns]. Vertical lines in (a)-(f) correspond to L1 and L2 size.

interfering cores, contention on L1 and L2 is almost absent in case of random interference (A3 and A4), while it is more noticeable in case of sequential interference (A1 and A2). When reaching L3 boundaries, a dramatic performance deterioration takes place, with latencies increasing up to 17x for sequential reads, and up to 6x for random reads, both with sequentially interfering cores. A smaller impact on relative performance deterioration is noticed when the interfering cores performs random reads.

The uneven spacing between the lines in the graph is due to the fact that half of the cores are not physical cores, but logical cores enabled by Intel HyperThreading technology. These logical cores share resources with a corresponding physical core, competing for L1 and L2 access. Therefore, logical cores do not add significant contention in system DRAM w.r.t. their physical counterparts [20], [21].

Experiments for Test Case B related to iGPU interference are shown in Figure 5 (c) and (f). As already pointed out in section III-C, a larger impact on latencies is expected due to the cache shared between the CPU complex and the iGPU. Somewhat unexpectedly, a performance deterioration is observed already with WSS smaller than L2. However, as pointed out in III-C, L2 is not shared with the GPU. The observed deterioration is instead due to the HW coherency mechanisms between L2 and L3, where this latter cache level is shared with the interfering GPU. Such effect was not observed on Tegra based platforms, as NVIDIA solutions rely on different SW controlled cache coherency mechanisms, and there are no shared caches between the CPU complex and the iGPU.

Test Case B shows that the interference at L2 boundaries is already so severe that it almost matches the interference experienced when accessing system DRAM, especially for sequential reads (B1). Differently from the previous boards,



(a) A1 (b) A2 (c) B1

(d) A3 (e) A4 (f) B2

(g) C1 (h) C2

Fig. 5: Test results for I7-6700. Black vertical lines in (a)-(f) correspond to L1, L2 and L3 CPU cache sizes

there is no big difference in performance deterioration when changing the nature of the interfering programs running on the iGPU, with the OpenCL equivalent of the memset (*clEnqueueFillBuffer*) slightly dominating the interference caused by other operations. Even in this case, it is difficult to isolate the effect of cache coherency caused by SVM for the same reasons discussed when treating CUDA UVM in Tegra-based architectures.

Figure 5 (g) and (h) show the behavior of the HD530 iGPU in case of unregulated CPU complex memory operations (Test Case C). In case of sequentially interfering cores, relative execution times of iGPU tasks may increase up to more than 200% for all GPU programs. A much smaller interference is instead observed in case of randomly interfering cores. Similarly to NVIDIA architectures, the most affected iGPU activities are OpenCL memset and memcpy. As observed in Test Case A, the height of adjacent bars are pairwise similar, due to the fact that half of the interfering cores are logical cores.

D. Prefetching mechanisms

The results from the previous test cases motivated us to analyze the prefetching mechanisms exploited in the three different SoCs. The substantially different behaviors expressed when the CPU was interfering or being interfered highlighted that prefetching plays a key role in performance degradation in memory contention scenarios. We stress that in all the tested environments, we left the HW prefetching mechanisms to the default SoC value[5]. We therefore tried to infer the prefetching mechanisms of the analyzed platforms by having a single core reading data from system DRAM at an increasing stride to identify read latency variations that depend on the number of pre-loaded LLC lines. Stride values for this test do not exceed

[5]According to ARM a15 and a57 reference manual, L2 cache prefetching can pre-load 0, 2, 4, 6 or 8 cache line after a LLC miss.

the page size value (4096B) as LLC prefetchers work between memory page boundaries.

The experimental results are shown in Figure 6 (a). Note that the leftmost point in the graph corresponds to the latency measured in the previous experiments with a non-interfered sequential read with a large WSS, while the rightmost points converge to the latency measured for random accesses. For the Tegra K1, varying the strides leads to no latency variation, showing very limited pre-fetching features. This is in accordance with the small performance gap observed between sequential and random reads in the experiments described in Subsection V-A.

A specular situation is observed with the X1, where a latency degradation occurs increasing the strides, especially beyond 512B. This is consistent with an L2 prefetch size of 8 lines (8*64B = 512B, with 64B being the cache line size). An even higher degradation is observed for larger strides, until reaching the size of DRAM row buffers (2 kB). No further increase in the latency is observed for strides beyond the row buffer size.

The Intel platform is again the most difficult to interpret, due to the presence of multiple prefetching mechanisms at different memory hierarchy levels, which are only partially disclosed in the related documentation [26]. These aggressive prefetching mechanisms allow lower latencies at CPU side in case of sequential reads (around 5ns). Increasing the strides, a 10x latency increase is observed with a stride of 256B, after that no further increase is observed.

E. Combined interference

In the previous experiments, we separately analyzed the interfering impact of CPU cores and iGPU. We conducted further experiments to analyze whether the above described interfering effects are additive when a CPU task may be simultaneously interfered by CPU cores and iGPU. We measured the delay experienced by a CPU task that is sequentially interfered by two cores, while another core iteratively sends memset commands to the iGPU (see Figure 6 (b)).

We hereafter outline only the results for the X1 platform. The observed latencies were up to 115 ns in case of sequential reads, and up to 600 ns in case of random reads. These numbers are in line with the results expected for an additive contribution of (i) the A57 sequential read idle latency of 15-18 ns, plus (ii) the delay of 40 ns due to two sequentially interfering cores (see Figure 4 (a)), plus (iii) the delay of 50-60 ns due to the iGPU memset (see Figure 4 (c)). A similar calculation leads to 600 ns in case the observed core performs random reads. Another interesting effect is observed in both the random and sequential case, where the latency delay caused by the iGPU memset is experienced already within L2 boundaries. This was not observed in Test Case B because CPU L2 is not shared with the iGPU. By adding two CPU interfering cores, evictions at L2 level caused by the CPU interference cause the observed core to resort to main memory sooner than the L2 cache size, while DRAM bandwidth is already saturated by iGPU activity.

VI. Conclusion and Future Work

In this paper, we presented a fairly complete overview on memory interference effects of Multi-Core CPUs and iGPU. We identified the main contention points in three different commercially available SoCs, taking accurate measurements of CPU read latencies and iGPU task execution times under various stress situations. Our contribution makes it evident that the detrimental effect on latencies escalates even more in presence of iGPU activity. We also showed how CPU activities may increase iGPU task execution times. We discovered how hardware prefetching, related memory access patterns and different iGPU activities play an important role in performance degradation. Also, we showed how unified CPU-iGPU cache levels have a larger impact in the measured delays than with separated caches. As a future work, We plan to implement a memory server that acts as a memory access arbiter to avoid the dramatic latencies reported in this paper. Such a server, likely to be implemented at hypervisor level, takes inspiration from recently introduced mechanisms, such as MEMGUARD and PREM, but revisited in order to take into account heterogeneous architectures featuring high-performance iGPUs.

Acknowledgment

This work is part of the Hercules and OPEN-NEXT projects, which are respectively funded by the EU Commission under the HORIZON 2020 framework programme (GA-688860) and ERDF-OP CUP E32I16000040007.

References

[1] N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovic, and A. Ramirez, "Experiences with mobile processors for energy efficient hpc," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 464–468.

[2] C. Nvidia, "Programming guide version 8.0," *Nvidia Corporation*, 2016. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[3] O. W. G. Khronos, "The opencl specification version 2.0," *Khronos Group*, 2015. [Online]. Available: https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf

[4] L. Chai, Q. Gao, and D. K. Panda, "Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. IEEE, 2007, pp. 471–478.

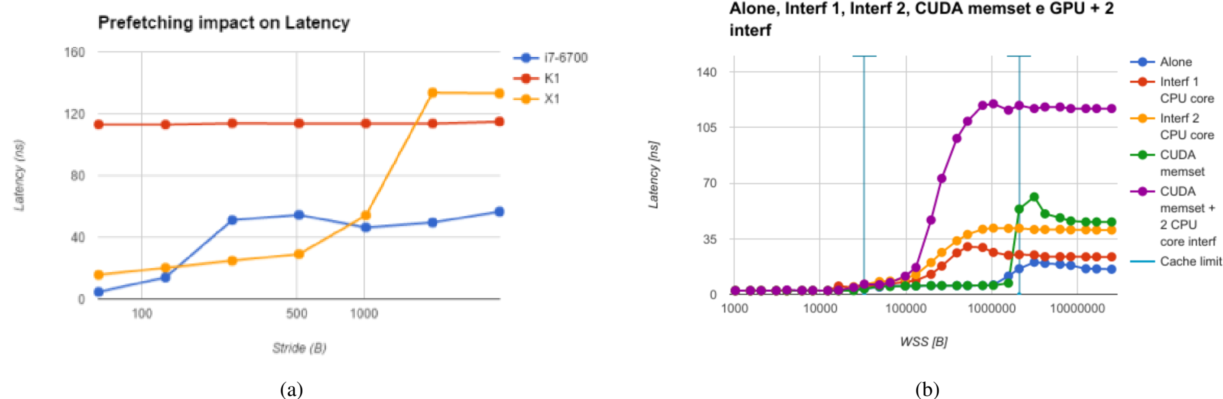
[5] H. Kim, D. De Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in cots-based multi-core systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*. IEEE, 2014, pp. 145–154.

[6] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 741–746.

[7] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee, "Response time analysis of cots-based multicores considering the contention on the shared memory bus," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 1068–1075.

[8] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo, "Global real-time memory-centric scheduling for multicore systems," 2015.

[9] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*. IEEE, 2013, pp. 55–64.



(a) (b)

Fig. 6: (a) HW prefetch impact for the tested platforms. (b) Combined interference in X1

[10] H. Yun, S. Gondi, and S. Biswas, "Protecting memory-performance critical sections in soft real-time applications," *arXiv preprint arXiv:1502.02287*, 2015.

[11] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 269–279.

[12] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A qos-aware memory controller for dynamically balancing gpu and cpu bandwidth use in an mpsoc," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 850–855.

[13] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale *et al.*, "Single core equivalent virtual machines for hard realtime computing on multicore processors," Tech. Rep., 2014.

[14] A. Rao, A. Srivastava, K. Yogesh, A. Douillet, G. Gerfin, M. Kaushik, N. Shulga, V. Venkataraman, D. Fontaine, M. Hairgrove *et al.*, "Unified memory systems and methods," Jan. 20 2015, uS Patent App. 14/601,223.

[15] B. A. Hechtman and D. J. Sorin, "Evaluating cache coherent shared virtual memory for heterogeneous multicore chips," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 118–119.

[16] NVIDIA, "Nvidia tegra k1 white paper, a new era in mobile computing," *NVIDIA Corporation*, 2014. [Online]. Available: http://www.nvidia.com/content/pdf/tegra_white_papers/tegra_k1_whitepaper_v1.0.pdf

[17] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*. IEEE, 2013, pp. 33–44.

[18] S. Goossens, B. Akesson, K. Goossens, and K. Chandrasekar, *Memory Controllers for Mixed-Time-Criticality Systems*. Springer, 2016.

[19] NVIDIA, "Nvidia tegra x1 white paper, nvidiaś new mobile superchip," *NVIDIA Corporation*, 2015. [Online]. Available: http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf

[20] D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty *et al.*, "Hyper-threading technology in the netburst® microarchitecture," *14th Hot Chips*, 2002.

[21] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra, and R. Biswas, "The impact of hyper-threading on processor resource utilization in production applications," in *High Performance Computing (HiPC), 2011 18th International Conference on*. IEEE, 2011, pp. 1–10.

[22] Intel, "The compute architecture of intel processor graphics gen9, v. 1.0," *Intel White Paper*, 2015. [Online]. Available: https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

[23] L. W. McVoy, C. Staelin *et al.*, "lmbench: Portable tools for performance analysis." in *USENIX annual technical conference*. San Diego, CA, USA, 1996, pp. 279–294.

[24] R. A. Starke and R. S. de Oliveira, "Impact of the x86 system management mode in real-time systems," in *Computing System Engineering (SBESC), 2011 Brazilian Symposium on*. IEEE, 2011, pp. 151–157.

[25] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 48–65.

[26] Intel, "Intel 64 and ia-32 architectures. optimization reference manual," *Intel Corporation*, 2016. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf