Understanding and Mitigating Memory Interference in FPGA-based HeSoCs

Gianluca Brilli, Alessandro Capotondi, Paolo Burgio, Andrea Marongiu
University of Modena and Reggio Emilia, Italy
name.surname@unimore.it

***Abstract*—Like most high-end embedded systems, FPGA-based systems-on-chip (SoC) are increasingly adopting heterogeneous designs, where CPU cores, the configurable logic and other ICs all share interconnect and main memory (DRAM) controller. This paradigm is scalable and reduces production costs and time-to-market, but creates resource contention issues, which ultimately affects the programs' timing. This problem has been widely studied on CPU- and GPU-based systems, along with strategies to mitigate such effects, but little has been done so far to systematically study the problem on FPGA-based SoCs. This work provides an in-depth analysis of memory interference on such systems, targeting two state-of-the-art commercial FPGA SoCs. We also discuss architectural support for *Controlled Memory Request Injection* (CMRI), a technique that has proven effective at reducing the bandwidth under-utilization implied by naive schemes that solve the interference problem by only allowing mutually exclusive access to the shared resources. Our experimental results show that: *i)* memory interference can slow down CPU tasks by up to 16× in the tested FPGA-based SoCs; *ii)* CMRI allows to exploit more than 40% of the memory bandwidth available to FPGA accelerators (normally completely unused in PREM-like schemes), keeping the slowdown due to interference below 10%.**

***Index Terms*—FPGA SoCs, Memory interference**

I. Introduction

Heterogeneous on-chip Systems (HeSoC) based on specialized acceleration logic like GP-GPU and FPGA have revolutionized the design of embedded systems, allowing the creation of increasingly sophisticated applications. In HeSoCs the very high peak performance and energy efficiency enabled by the accelerators is coupled to the flexibility of a multi-core, general-purpose *host* CPU. FPGA-based ICs further add to the flexibility by leveraging the concept of reconfigurability. Commercial-of-the-shelf (COTS) products based on this architectural template reduce production costs and time-to-market.

These systems typically rely on a shared-memory organization, where several distinct IPs are interconnected through a shared bus or network-on-chip (NoC) to the main system DRAM. While this solution simplifies the deployment of applications, and provides very good average-case performance, it does not cope well with those scenarios where timing guarantees must be provided. As the number of cores and other blocks sharing memory, interconnect and I/O resources grows, the effect of interference is more and more impactful on the worst-case latency observed from a task [1]–[3].

The Predictable Execution Model (PREM) [4] is a notable paradigm for the development of parallel software that is robust to memory interference, offering a task model where execution is structured as a repeating sequence of memory phases (that access the shared memory) and compute phases (that only use local caches and registers). By allowing only a single actor at a time to execute its memory phase, PREM avoids timing delays due to memory interference by construction.

PREM has been studied in the context of single-core CPU and I/O [4], multicore CPUs [5] and HeSoCs [6]. Although effective at guaranteeing predictable timing of memory accesses, PREM and PREM-like approaches [7], [8] greatly under-utilize memory bandwidth, which in HeSoCs is designed to concurrently serve multiple computing units. Recently, *Controlled Memory Request Injection* (CMRI) has been proposed as a technique that can operate on top of PREM-like schemes, enabling fine-grained control of the unexploited bandwidth [9].

Most papers studying this problem target multi-core CPUs [1], [10] or GPU-based HeSoCs [2], [6], [11], while not much has been done so far in the context of FPGA-based HeSoCs. This paper aims at filling this gap by means of: i) extensive memory interference characterization on two state-of-the-art FPGA-based HeSoCs for high-end embedded systems: the *Xilinx Zynq UltraScale+ ZU9EG* and the *Xilinx Versal VC1902 ACAP*; ii) an implementation of the CMRI technique [9] suitable for generic FPGA accelerators; iii) a thorough assessment of the resulting benefits on the memory bandwidth exploitation.

The paper is organized as follows: in Section II we discuss related work and motivate our own. In Section III we describe the architecture of the target platforms of this work. In Section IV we introduce the generic accelerator template considered and its extensions to support CMRI. The experimental setup and results are described in Section V. Section VI provides concluding remarks.

II. Related Work

Memory interference has been extensively studied in the context of multi-core CPU and GP-GPU HeSoCs [2] [11] but only very few papers have addressed the problem in FPGA-based HeSoCs. A. Bansal et al. [1] characterize the memory subsystem of the ZU9EG MPSoC, through micro-benchmarks. They also propose a hardware/software infrastructure capable of guaranteeing isolation. K. Manev et al. propose a characterization of the memory

978-3-9819263-6-1/DATE22/©2022 EDAA

subsystem of two SoCs belonging to the *Zynq UltraScale+* family: ZU9EG and ZU3EG [12]. Their experiments involve memory-mapped accelerators interconnected in DRAM through the *AXI HP* ports. Memory interference between Host CPUs and FPGA is not addressed. The closest exploration to ours is from M. Mattheeuws et al. [3], which conduct a similar analysis on the Xilinx ZU9EG SoC, evaluating the performance slowdown that a task running on the ARM cluster experiences when traffic generators instantiated on the FPGA contend for the DRAM bandwidth. While the aims are very similar to ours, here the analysis is overall less comprehensive, carried out on a single SoC and neglectful of solutions to mitigate the effects of interference.

Memory interference mitigation has been studied both at the hardware and software level. Concerning the former, F. Restuccia et al. [13], [14] have explored the fairness of the *AXI Interconnect* IP of the Xilinx ecosystem, showing that although the bus access mechanisms implement a *Round-Robin* arbitration, in certain conditions fairness may not be guaranteed. F. Farshchi et al. [15] try to mitigate memory contention exploiting bandwidth regulations on host cores via an FPGA-based component sitting between *host* and shared buses. The focus is on multi-core systems, not on heterogeneous, FPGA-based SoCs.

On the software side, both the Certification Authorities [7] and a number research groups [4]–[6], [8] have outlined mutually exclusive access to the DRAM as the practical solution to address the adverse effects of memory interference in shared-memory SoCs. These approaches are typically too pessimistic, as their *one-at-a-time* execution model causes severe under-utilization of the system DRAM bandwidth [5], [8], [16]. Attempts to overcome such pessimism have been made. G. Yao et al. [17] allow multiple tasks to access DRAM at the same time to increase the memory bandwidth utilization, but the granularity of the approach is too coarse, failing to provide timing guarantees. Controlled Memory Request Injection (CMRI) [9] also allows more than a single PREM task at a time to access DRAM. The memory phases of such tasks are implemented as fine-grained controllable duty cycles, where memory requests are interspersed with idle cycles (via compiler-level instrumentation or via dynamic task throttling [18]). CMRI has proven effective at increasing bandwidth utilization with little impact on the latency of the main PREM task in multi-cores CPUs.

In this paper we build upon these findings to study the problem of memory interference in FPGA-based SoCs. We propose a scheme to deploy CMRI on FPGAs, to target generic FPGA-based accelerators, and thoroughly study the benefits that can be reaped from such support. We also investigate applicability of this approach applied to real world use-cases.

III. Platform Description

FPGA-based HeSoCs feature a *host complex*, composed of several general-purpose CPUs coupled to an FPGA subsystem and other application-specific accelerators (e.g. GP-GPU, NN Inference Engines). As a representative embodiment of such a general template we study two *commercial-of-the-shelf (COTS)* platforms from Xilinx: the *Zynq UltraScale+ MPSoC* and the *Versal ACAP*.

Xilinx Zynq UltraScale+: it is composed of a heterogeneous *host* processor *(PSU)* alongside an FPGA for the acceleration of compute-intensive tasks. The PSU is composed of two processor islands, named *APU* and *RPU*. The *APU* consists of a quad-core *ARM Cortex A53* processor implementing the *ARMv8-A* micro-architecture, and constitutes the target choice for general-purpose computing. The *RPU* consists of a dual-core *ARM Cortex R5F*, which is used to deploy control tasks with real-time requirements. The FPGA fabric is interfaced with the system via different AXI4 ports, some of which allow for coherent transactions with the PSU through the CCI. In contrast, some others leverage a dedicated path to the DRAM controller. In this work, we focus on the *AXI High-Performance Ports (HP)*, which can sustain the maximum bandwidth to the DRAM [19]. Each of the HP ports has a width of *128 bits* and can reach a maximum frequency of *300MHz*. The maximum nominal DRAM bandwidth is 17 GB/s. Aa a representative implementation of Xilinx Zynq UltraScale+ we selected the ZU9EG.

Xilinx Versal ACAP: in this SoCs, the concept of heterogeneity is further developed by combining three main computational units: *host* cores, FPGA and *AI Engines (AIE)*. As *host* CPUs the VC1902 embeds an *ARM Cortex A72* dual-core processor for general-purpose computing and an *ARM Cortex R5F* dual-core processor for real-time workloads. The RPU is the same as the Zynq Ultrascale+ MPSoC. The AIE is composed of VLIW cores organized as a systolic two-dimensional array to accelerate Deep Learning workloads. The AIE subsystem is connected to the main memory through the FPGA. In our test environment, we joined the AIE and main memory through the *Deep Learning Processing Unit (DPU)* IP, which also implements some layers not supported yet by the AIE. The three computational units can access DRAM through a configurable *Network-on-Chip (NoC)*. Unlike on Zynq UltraScale+, it is possible to enable a greater number of ports to interconnect the FPGA to the NoC. The width of these AXI4 ports is 128 bit, and their maximum working frequency is 300MHz. The maximum nominal memory bandwidth is 25.6 GB/s. Aa a representative implementation of Xilinx Versal we selected the VC1902.

IV. Accelerator Template with CMRI Support

To conduct an in-depth analysis and characterization of the memory interference in a SoC a typical methodology is that of relying on synthetic workloads aimed at stressing corner cases [2], [9]. When focusing on a FPGA-based HeSoC, a typical way of generating configurable synthetic memory traffic is that of deploying some form of *traffic generators* [3]. More in general, full-custom acceleration logic is typically designed as shown in left part of Figure 1, where the core acceleration logic (datapath) is coupled to some sort of *data mover* or DMA engine and a local

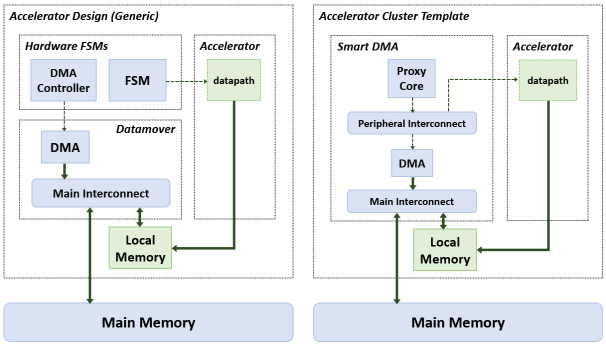

Fig. 1: Two architectural designs for accelerator deployment. Left: a traditional design, where the accelerator and DMA datapaths are controlled by an FSM. Right: architectural design used in this work, where the control is implemented by a soft-processor.

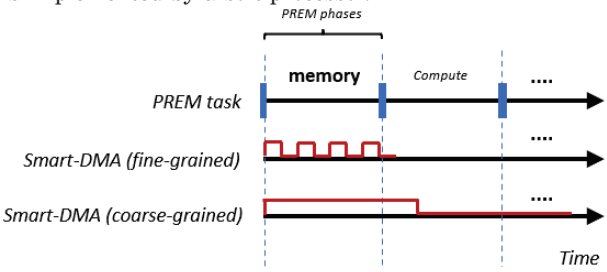

Fig. 2: Impact of adopting fine-grained *subcopies*.

memory. The *data mover* leverages a Finite State Machine (FSM) or DMA controller to supervise the flow of data in and out the local memory as the datapath executes.

We build on top of this general block diagram to design our extensions to support Controlled Memory Request Injection (CMRI). Our *Accelerator Cluster Template* is shown in Figure 1 *(right)*. Here, we consider the DMA and local memory as immutable parts of an IP, the *Smart DMA*, that can be interfaced to different datapaths. To flexibly support different types of control logic for the DMA (and the datapath itself) we rely on a programmable core – the *proxy core* – rather than on a FSM logic. This paradigm moves the control on the software side, allowing for higher flexibility and better data management [20], [21].

The *Smart DMA component* allows for controllable DMA operation, which can be used to ensure that the bandwidth request generated from the FPGA accelerators does not impact the performance of other active cores/tasks in the system beyond what can be tolerated.

As we have already explained in Section III, the Programmable Logic blocks can generate much higher sustained bandwidth request to the DRAM controller than what can be done on the ARM Application Processors. On the other hand, if we decided to stick to a PREM-like *one-at-a-time* arbitration of memory accesses and prevent completely the FPGA accelerators from accessing DRAM while the host CPU tasks are also accessing DRAM, we would end up severely under-utilizing the system bandwidth. Given this scenario, our goal is that of controlling DMA requests from FPGA accelerators in a way that at any moment we only allow up to the maximum bandwidth that does not degrade the execution latency of the CPU tasks beyond a determined threshold. This has been achieved on multi-core CPU SoCs via *throttling* of the memory requests, which we implement on our Smart DMA through *duty cycling*.

Listing 1 shows a small piece of code that highlights the main idea of the duty-cycled loop executed on the Smart-DMA component described above. Given a standard DMA transfer request to the Smart DMA, described as the triple $< destination, source, size >$, the Proxy Core splits up this request into smaller transfer requests interleaved with a configurable amount of *idle cycles*. This allows a fine-grained adjustment of the bandwidth requested to the shared memory subsystem. We define the notion of *Cluster Load Intensity* (CLI) to indicate in our experiments the amount of *throttling* (IDLE_CYCLES) that we apply to the system. Intuitively, when IDLE_CYCLES is zero we operate the accelerator clusters at full throttle, which corresponds to 100% *cluster load intensity* (CLI). Altering the IDLE_CYCLES parameter produces different, controllable amounts of *cluster load intensity*.

```
for(size_t i = 0; i < transactions; ++i){
    // a single transaction is split in
    // subcopies to interleave idleness
    for(size_t j = 0; j < subcopies; ++j) {
        dma_memcpy(&dma,
            dst_buffer+j*(SIZE/subcopies),
            src_buffer+j*(SIZE/subcopies),
            SIZE/subcopies
        );
        idle(IDLE_CYCLES);
    }
}
```

Listing 1: firmware running on the *Proxy Cores* to implement CMRI.

In Figure 2 we can see the effect of having a fine-grained smart-DMA, i.e. able to operate even on small PREM regions. We can see that in the *fine-grained* case, it is possible to interleave memory transactions with idleness, allowing for finer bandwidth adjustments. The *coarse-grained* version operates with larger copies, which does not allow for bandwidth adjustment within the memory-phase in the example considered.

V. Experimental Results

Our experimental evaluation is organized in three main sections. First (Section V-A), we study the self-interference that various processing elements from the same block can experience (multiple cores from the same APU or RPU, or multiple accelerator clusters from the FPGA). Second (Section V-B), we characterize the combined effect of inter- plus intra-block memory interference by measuring the worst-case delay a task running on one processing element under test (one of the APU/RPU cores or the AI Engine on VC1902) can experience when the rest of the cores and the FPGA accelerator *clusters* are generating traffic

TABLE I: Comparison of maximum measured DRAM bandwidth in GB/s, on *host* cores (varying memory access pattern) and *Accelerator Clusters* of the two systems.

SoC	*PE Type*	*Pattern*	#Active Cores 1	2	3	4
ZU9EG	*A53*	sequential	2.50	4.77	6.50	7.20
	A53	random	0.39	0.76	1.12	1.46
	R5F	sequential	0.27	0.52	-	-
	R5F	random	0.21	0.42	-	-
VC1902	*A72*	sequential	6.98	7.28	-	-
	A72	random	0.35	0.68	-	-
	R5F	sequential	0.22	0.41	-	-
	R5F	random	0.19	0.38	-	-
SoC	*PE Type*		#Active Clusters 1	2	3	4
ZU9EG	*Acc. Cluster*		8.93	13.6	13.2	14.0
VC1902	*Acc. Cluster*		8.68	17.3	21.6	24.0

at full throttle. Third (Section V-C), we study the benefits of CMRI by allowing the FPGA cluster(s) to inject memory requests at increasing CMRI rates and observing the effect that this has on the task running on one of processing elements under test.

Both for the ZU9EG and the VC1902 we instantiate four *clusters*, connecting them to each of the available four AXI HP ports. The accelerator clusters operate *@300MHz*. The small control firmware of the *Smart DMAs* runs on the *Proxy Core* of each accelerator *cluster*.

As reference tasks running on the ARM cores we consider both micro-benchmarks – capable of generating both *sequential* and *random* memory traffic patterns towards the DRAM and modeled after the `lmbench` test suite[1] – and real benchmarks from the *Polybench* suite[2]. In both cases the benchmarks are configured with a working-set-size that exceeds the L2-cache of the two SoCs. These benchmarks are executed on top of Linux for the ARM A53 and A72 and on bare metal for the Cortex R5F. As reference tasks running on the AI Engines of the VC1902 SoC we consider several popular CNNs for object detection and classification, widely adopted on embedded platforms. Pre-trained versions of these networks are available on Xilinx Model Zoo[3].

A. Self-Interference among CPU cores

Table I shows the maximum memory bandwidth generated by the various IPs (cores and FPGA accelerator *clusters*) when running at full throttle on the two SoCs.

The topmost part of the table shows the maximum DRAM bandwidth available on the RPU and APU of the two systems. In this case all the cores execute the micro-benchmarks simultaneously. Focusing on the sequential traffic pattern (the most memory-intensive) we observe a difference between the two SoCs. In the ZU9EG MPSoC a single core from both the APU or RPU complex only uses a fraction of the total bandwidth, and an almost additive behaviour is observed as more cores are used. On the VC1902 system the same applies to the RPU, but on the APU a single core can saturate the available memory bandwidth to the whole APU complex. When the second core is activated, this total bandwidth is evenly shared between the two cores.

The bottom part of the table highlights the memory bandwidth that the FPGA can achieve on the two reference platforms. The ZU9EG reaches $\approx 14GB/s$, which is 82% of its nominal maximum bandwidth (i.e., of the whole SoC). Similar to what happens on the APU, this suggests that static partitioning of the bandwidth is applied, corroborated by the fact that two clusters alone (i.e., two AXI HP ports at full throttle) can saturate the available bandwidth, and adding more does not increase it further. The VC1902 reaches $24GB/s$, which is 94% of its nominal maximum bandwidth. Again, similar to what happens on the APU in this case there are no evident effects of static bandwidth partitioning: it takes the whole four clusters to saturate the bandwidth of the VC1902.

B. Memory Interference Characterization

The experiments described in this section aim to quantify the impact of memory interference from the FPGA clusters on the workload *under test*.

As processing elements under test (UT) we consider three IPs: (i) RPU cores; (ii) APU cores (A53 and A72); (iii) AI Engines (AIE - on VC1902 only). As workload, we execute micro-benchmarks on the RPUs, Polybench benchmarks on the APUs, and various CNNs on the AIE. As we are interested in studying the worst-case scenario, we set the number of active FPGA accelerator clusters to the maximum (four) and we execute an instance of each benchmark on each IP *under test* (on the APUs and RPUs one instance on each core).

Table II shows the slowdown due to interference when the clusters are running with *cluster load intensity* CLI = 100%. The table is divided in three parts, one for each IP under test: APU, RPU and AI Engines.

Focusing on the RPUs, for sequential traffic – the most sensitive to interference – the four accelerator clusters can slow down the execution of the task *under test* by 3.97× on ZU9EG and by 5.58× on VC1902. Random patterns, which are usually less affected by interference, still suffer from up to 3.18× and 4.95× slowdowns on ZU9EG and VC1902, respectively. This finding is particularly relevant if we consider that RPU cores are typically used for latency-critical tasks, as they can exploit dedicated resources which are physically not shared with the rest of the system. Yet, if we do rely on those memory paths that are shared with the rest of the system the effects on timing can be severe.

Focusing on APUs, we can see different behaviours for each tests, as the real benchmarks exhibit very diverse computation-to-communication-ratios. On the ZU9EG we see up to 7.4× slowdown. On the VC1902 we see even greater interference, due to the larger fraction of memory bandwidth used by the FPGA and the effects of self interference (as we observed earlier the APU cores have

[1]http://lmbench.sourceforge.net/

[2]https://web.cse.ohio-state.edu/ pouchet.2/software/polybench/

[3]https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo

TABLE II: Effect of memory interference (slowdown) from four FPGA *clusters* on different *Under Test (UT)* IPs : APU cores (A53, A72), RPU cores (R5F) and AI Engines (AIE).

RPU			APU			AIE	
	ZU9EG	VC1902		A53	A72		VC1902
seq	3.97	5.58	adi	2.0	3.9	Inceptionv2	2.44
ran	3.18	4.95	atax	1.4	4.3	Refinedet	2.45
			bicg	1.3	2.4	Resnet50	1.12
			conv2d	1.5	2.0	Yolov3-Tiny	2.07
			covariance	1.9	2.2	Yolov2-pruned	1.48
			fdtd-2d	3.1	16	Yolov2	1.94
			cholesky	2.3	10	Yolov3	1.98
			mvt	3.9	5.1	Yolov4	2.02
			trisolv	1.7	11		
			durbin	7.4	8.6		
			gramschmidt	6.0	13		
			lu	3.2	13		

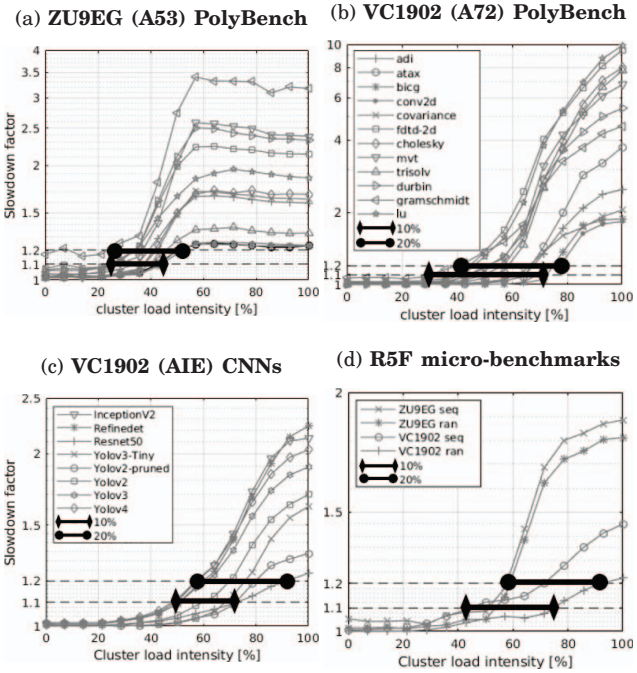

Fig. 3: *CMRI* on different Processing Elements and platforms. (3a and 3b): Polybench on APUs; (3c): CNNs on AI Engines; (3d): Micro-benchmarks on R5F.

non additive behavior in terms of bandwidth usage). In this case the slowdown can go as high as 16×.

Finally, focusing on the AI Engine of the VC1902, we study the effect of memory interference from the FPGA on the CNN workloads. Like in the previous case, to also consider self interference within the IP under test we execute two instances of each CNN in parallel.

Also in this case we see that different CNNs have different sensitivity to interference: while the most sensitive one, `refinedet`, can suffer up to ≈ 2.5× slowdown, the least sensitive ones (`yolov2_voc_pruned` and `resnet50`) experience less than 1.5× slowdown even in the worst case. This observation further consolidates the intuition that CMRI schemes should be beneficial in FPGA-based heterogeneous SoCs, too, as a conservative scheme like PREM would severely underutilize the system memory bandwidth for CNNs with poor communication-to-computation ratios.

C. Controlled Injection

The experiments presented in this section aim at studying to which extent it is possible to allow concurrent execution of FPGA accelerators and CPU or AIE workloads – provided that we can control and bound the effects of interference – as opposed to PREM-like DRAM-arbitration schemes, where only one task at a time is allowed to access the main memory. In particular, similar to the original CMRI setup [9], we measure the memory bandwidth that the throttled entity (here, the FPGA accelerators) can utilize while maintaining the slowdown imposed on the tasks *under test* below a given threshold. We consider two thresholds, at 1.1× and 1.2× slowdown, and we measure the *Cluster Load Intensity* (CLI), i.e., the maximum injection rate that does not exceed such thresholds. Clearly these threshold values are only for illustration purposes: different applications/systems will have different maximum tolerated latency increase. Note that in this context CLI acts as an indicator of FPGA bandwidth usage efficiency (BWE), as it represents the bandwidth injected by the FPGA cluster(s) normalized to the maximum bandwidth they can request (see Table I).

Like in the previous section, we consider both the case where the task *under test* runs on the RPU/APU and the case where the task *under test* runs on the AIE. The setup in terms of benchmarks and configurations is the same.

Figure 3 shows the results for this experiment, where a single FPGA cluster is injecting traffic at various CLI rates. Due to the lack of space we don't report plots for the experiment with four clusters, but we will report the key findings in the discussion. The plots in this figure show workload slowdown with increasing CLI. The markers with circular and rhomboidal ends highlight the ranges in which CMRI operates within the tolerated 10% and 20% thresholds, respectively.

Focusing on the APUs as units under test (Figures 3a and 3b), we see that CMRI allows to exploit 22-to-43% of the maximum bandwidth that a single cluster can generate on the ZU9EG (8.93GB/s) and 28-to-72% on the VC1902 (8.68GB/s), while guaranteeing a maximum latency increase of the workload under test of up to 10%. If the tolerance threshold is increased to 20% CMRI can exploit 23-to-53% of the maximum cluster bandwidth on the ZU9EG and 40-to-78% on the VC1902. Note that if the same experiment is repeated on four clusters we only observe a difference in the minimum CLI allowed for the most conservative threshold on ZU9EG, which is reduced to 0.2%. In all the other cases the CLI values remain nearly identical, but the maximum bandwidth to which

they refer is in this case much higher (see Table I): 14 GB/s for ZU9EG and 24 GB/s for VC1902.

On the RPUs (Figure 3d) CMRI allows to exploit 41-to-78% of the maximum bandwidth for a single cluster on both SoCs for the 10% threshold, and 59-to-93% for the 20% threshold. If we consider four clusters CMRI allows to exploit 8-to-49% of their maximum bandwidth for the 10% threshold, and 29-to-54% for the 20% threshold.

On the AIE (Figure 3c) CMRI allows to exploit 50-to-72% of the maximum bandwidth that a single cluster can generate on VC1902 for the 10% threshold, and 59-to-93% for the 20% threshold. Even when considering four active clusters the CLI values remain nearly identical, which means there are cases in which we can exploit nearly the whole bandwidth generated by the FPGA without significantly impacting the timing of the CNN workloads.

VI. Conclusion

In this work we presented an in-depth memory interference analysis for two last-generation FPGA-based HeSoCs belonging to the *Zynq UltraScale+* and *Versal ACAP* families. This fills a gap in the literature, which mostly focuses on multi-core CPUs and GPU-based HeSoCs, and confirms the fact that memory interference strongly affects also FPGA-based devices. When all the CPU cores and the FPGA logic are executing in parallel, memory interference can slow down APU tasks by more than 16×, RPU cores by up to 5.58× and the AI Engines on the VC1902 by up to ≈ 2.5×. Our analysis highlights thus that Real-Time RPU cores are potentially impacted by memory interference, which must be taken into account. Globally, the analysis points out that also in FPGA-based HeSoCs PREM-like approaches that arbitrate accesses to DRAM in a mutually exclusive manner are bound to severely underutilize the available memory bandwidth. We propose an approach to support state-of-the-art CMRI technique on FPGA-bases HeSoCs, integrated in a generic template for FPGA accelerator clusters based on a smart DMA.

Our experiments show that CMRI, previously only applied to multi-core CPUs, is very effective also in FPGA-based HeSoCs, allowing in the best case to exploit over 90% of the available bandwidth to the FPGA accelerators, while providing analogous timing guarantees to what is offered by PREM-like schemes [4], [7].

VII. Acknowledgements

The authors have received funding from the ECSEL-JU projects COMP4DRONES (No. 826610) and FRACTAL (No. 877056).

References

[1] A. Bansal, R. Tabish, G. Gracioli, R. Mancuso, R. Pellizzoni, and M. Caccamo, "Evaluating the memory subsystem of a configurable heterogeneous mpsoc," in *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 07 2018, p. 55.

[2] N. Capodieci, R. Cavicchioli, I. S. Olmedo, M. Solieri, and M. Bertogna, "Contending memory in heterogeneous socs: Evolution in nvidia tegra embedded platforms," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2020, pp. 1–10.

[3] M. Mattheeuws, B. Forsberg, A. Kurth, and L. Benini, "Analyzing memory interference of fpga accelerators on multicore hosts in heterogeneous reconfigurable socs," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1152–1155.

[4] 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*, 2011, pp. 269–279.

[5] A. Alhammad and R. Pellizzoni, "Time-predictable execution of multithreaded applications on multicore systems," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6.

[6] B. Forsberg, L. Benini, and A. Marongiu, "Heprem: Enabling predictable gpu execution on heterogeneous soc," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 539–544.

[7] C. A. S. T. (CAST), "Multi-core processors," *Position Paper CAST-32A*, 2016.

[8] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.

[9] R. Cavicchioli, N. Capodieci, M. Solieri, M. Bertogna, P. Valente, and A. Marongiu, *Evaluating Controlled Memory Request Injection to Counter PREM Memory Underutilization*, 11 2020, pp. 85–105.

[10] R. Pellizzoni, A. Schranzhofer, Jian-Jia Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, pp. 741–746.

[11] H. Wen and W. Zhang, "Interference evaluation in cpu-gpu heterogeneous computing," *IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

[12] K. Manev, A. Vaishnav, and D. Koch, "Unexpected diversity: Quantitative memory analysis for zynq ultrascale+ systems," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 179–187.

[13] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? restoring fairness in axi interconnects for fpga socs," *ACM Transactions on Embedded Computing Systems*, vol. 18, pp. 1–22, 10 2019.

[14] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[15] F. Farshchi, Q. Huang, and H. Yun, "Bru: Bandwidth regulation unit for real-time multicore processors," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 364–375.

[16] M. R. Soliman and R. Pellizzoni, "PREM-Based Optimal Task Segmentation Under Fixed Priority Scheduling," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Quinton, Ed., vol. 133. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 4:1–4:23. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/10741

[17] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo, "Global real-time memory-centric scheduling for multicore systems," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2739–2751, 2016.

[18] 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 *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 55–64.

[19] S. W. Min, S. Huang, M. El-Hadedy, J. Xiong, D. Chen, and W.-m. Hwu, "Analysis and optimization of I/O cache coherency strategies for SoC-FPGA device," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019.

[20] G. Bellocchi, A. Capotondi, F. Conti, and A. Marongiu, "A risc-v-based fpga overlay to simplify embedded accelerator deployment," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, 2021, pp. 9–17.

[21] A. Kurth, A. Capotondi, P. Vogel, L. Benini, and A. Marongiu, "HERO: An open-source research platform for HW/SW exploration of heterogeneous manycore systems," in *Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems*, 2018, pp. 1–6.