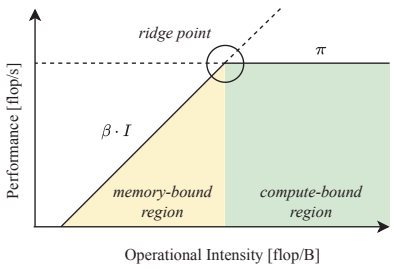

Fig. 1. Illustrative roofline plot.

system, as its real-time properties may be negatively altered. Our extended roofline model presented in Section IV-C provides an intuitive way to see the impact of memory interference, providing a useful tool for real-time system designers.

B. The ZU9EG SoC

The ZU9EG is a popular Xilinx Ultrascale+ SoC. It is a multiprocessor system on chip (MPSoC), integrating an ARM Cortex-A53 quad core and programmable logic (PL), see Figure 2. Every core possesses a private instruction and data cache, each of 32 KiB size. All cores share a 1 MiB L2 cache. Cache lines are 64 B wide. The SoC's operating system is Xilinx' PetaLinux 2017.2 based on Linux 4.9.0.

The PL can access the DRAM through four AXI4 ports. Each port has a data width of 128 bit and can be clocked up to 300 MHz. Shown in Figure 2, the four ports are connected to three DRAM controller ports, where crossbars join two ports to one. If multiple ports are active concurrently, the DRAM controller has to arbitrate. We do not use the DisplayPort and the FPD DMA connected to the other DRAM controller ports. The DRAM throughput is maximum 2400 MT/s at 64 bit width and its total capacity is 4ĠiB.

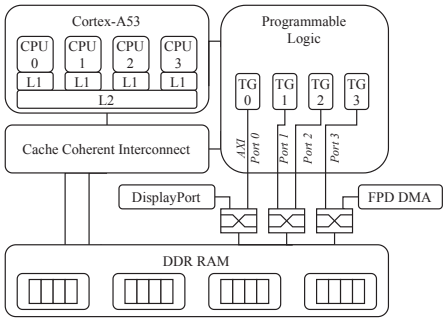

Fig. 2. Abstracted ZU9EG block diagram with TGs in the PL.

III. Platform Characterization Methodology

To characterize the impact of memory interference, we set up a profiling infrastructure to analyze the response time of benchmarks executing on an A53 core while the PL injects traffic into main memory. We instantiate access-generating modules on the PL, as well as CPU benchmarks upon which the interference effect of DRAM traffic can be measured.

A. Hardware

We set up one Xilinx AXI4 traffic generator (TG) [9] per AXI4 port in the PL, resulting in four TGs as depicted in Figure 2. They have a common frequency but can be enabled individually, allowing up to 4.8 GB/s throughput per port. The available frequencies are 250 MHz, 166.67 MHz, 125 MHz, and 93.75 MHz, below which every integer division of 1.5 GHz can be set. A cumulative bandwidth of up to 8 GB/s can be injected by the TGs before the ZU9EG SoC becomes unresponsive due to the processing system (PS) being starved for memory.

B. Software

We consider well-known benchmarks, as well as a customized one that repeatedly causes cache misses with the lowest possible intensity. They are compiled using LLVM 9 [10] with the `-O3` option. We use matrices and vectors of the shape $n \times n$ and n, respectively, with a total memory footprint of 2 MiB, which is twice the size L2 cache (see Table I).

TABLE I
Benchmark intensities and parameters.

	`axpy`	`conv2d`	`bicg`	`3mm`	`2mm`	`gemm`
I	0.125	0.5	0.994	58.21	80.36	104.6
n	262144	512	722	273	323	418

1) Benchmarks: We consider PolyBench [11] benchmarks with various intensities. Their intensities are obtained by counting the floating-point operations in their assembly codes. For `axpy` and `conv2d`, the compiler introduces vector floating-point (VFP) instructions. We compute the flop per VFP instruction by normalizing to the number of vectors it operates on; e.g., a VFP instruction operating on four elements counts as 1/4 flop.

2) Memory-Trashing with `stride`: We also consider `stride`, a synthetic benchmark that accesses the next cache line with 64 B long strides to continually cause L1 and L2 cache misses, see Algorithm 1. It is therefore highly sensitive to memory interference.

Algorithm 1: `stride` with intensity control.

```
   Data: vectors X,Y of length n, and a scalar k
   Result: Y=X+Y
1  stride s=16;
2  for i=0;i<n;i+=s do
3  |  for j=0;j<k;j++ do
4  |  |  Y[i]+=X[i];
5  |  end
6  end
```

Additionally, we add a nested loop on line 3 to repeat the computation. As data is reused immediately, this allows to modulate `stride`'s the computational intensity. We set `k=1` in Section IV-A to evaluate several traffic configurations and set various `k`s in Section IV-C to measure a roofline under different interference levels.

IV. Memory Interference and the Roofline Model

We use two components to measure the impact of memory interference: First, we instantiate one TG per AXI port in the PL. Second, we measure a set of commonly-used software benchmarks covering a wide range of intensities on the host CPU. In this section, we employ this methodology to evaluate traffic patterns and then generate a measured roofline under increasing DRAM traffic injection on the ZU9EG.

We begin by finding the parameters that generate the most interference. As described in Section III-A, the PL contains TGs we can enable individually. Each active TG injects the same amount of bandwidth. As two TGs share a common port to the DRAM controller, that port needs to handle twice as much traffic if both its TGs are active.

Unless stated otherwise, all results are the WCETs from 128 measurements. We time the execution of the kernel only with the `clock_gettime(CLOCK_MONOTONIC)` function. We also use the `taskset` command to assign the benchmarks to a core, set a minimum `nice`ness to avoid preemptions by the OS scheduler, and disable swapping.

A. AXI Ports and Interference

For each possible combination of active TGs running at every frequency we can set on the PL, we quantify the memory interference slowdown H of `stride` as the interfered divided by the non-interfered WCET.

We found differences in slowdowns depending on the shared DRAM port usage, as presented in Table II. The superscript differentiates between shared (*shrd*) or individual (*indv*) port usage, depending if TG1 and TG2 are simultaneously injecting traffic on the common port or not. The subscript indicates how many TGs are active. The table excludes slowdowns using a single TG, which increases linearly with the injected traffic bandwidth and is only $H_{1\text{TG}} = 1.21$ at most.

TABLE II
Slowdowns caused by varying amounts of traffic generators.

Injected BW	$H^{\text{indv}}_{2\text{TG}}$	$H^{\text{shrd}}_{2\text{TG}}$	$H^{\text{indv}}_{3\text{TG}}$	$H^{\text{shrd}}_{3\text{TG}}$	$H_{4\text{TG}}$
320 MB/s	1.17	1.12	1.11	1.12	1.12
1600 MB/s	1.18	1.28	1.21	1.20	1.19
3200 MB/s	1.26	1.29	1.33	1.21	1.31
5984 MB/s	2.51	3.01	2.09	2.03	1.62
8000 MB/s	4.37	6.51	11.44	6.22	25.99

It is not relevant which TGs are active for lower bandwidths. Pushing the traffic injection to the maximum (see Section III-A), we measure the highest slowdown using 4 TGs. Maximum interference with only 2 TGs is generated when they are placed on the shared DRAM port. In this setting, the DRAM controller sees the highest single-port load we can generate. Interestingly, when using 3 TGs, most interference is generated when they are placed on independent ports. Furthermore, having them in that setting exhibited the most stable behaviour w.r.t. interference created, while running all 4 TGs displayed the most variance. From this we conclude that the DRAM controller works best with traffic evenly distributed among the ports, but that the magnitude of the slowdown is dependent on the amount of interfering TGs. As expected, running all 4 TGs at once causes the worst interference.

TABLE III
Worst-case execution time slowdown factors.

Injected BW	`2mm`	`3mm`	`axpy`	`bicg`	`conv2d`	`gemm`
320 MB/s	1.00	1.00	0.98	1.00	1.00	1.01
1600 MB/s	1.01	1.00	1.15	1.00	0.97	1.04
3200 MB/s	0.99	1.00	1.02	1.00	0.98	1.05
5984 MB/s	1.00	1.00	1.27	1.01	0.99	1.04
8000 MB/s	1.00	1.00	19.00	1.30	7.91	1.00

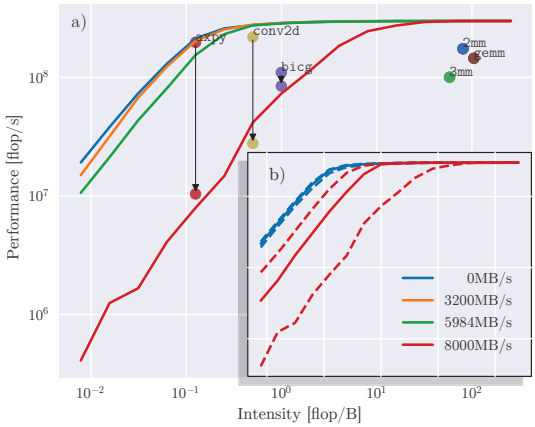

Fig. 3. Roofline degradation under increasing memory traffic bandwidths.

B. CPU Benchmarks under Interference

Based on last section's findings, we explore the benchmark behaviour under maximum interference by activating all four TGs and modulating their frequency to vary the injected bandwidth. The results can be seen in Table III, showing the slowdown for each benchmark. `axpy` evolves in a similar fashion to `stride` in Table II in the previous section. This is due to the fact that they have a similar structure, linearly accessing elements to perform computation. For benchmarks with $\mathcal{O}(n^2)$ complexity, we see that `conv2d` is more susceptible to interference than `bicg`. This is because it has a different access pattern and lower intensity (0.5 flop/B vs. 0.994 flop/B). Benchmarks with higher intensity are compute-bound enough to not suffer from significant slowdown: the execution time increases by 19× for the heavily memory-bound `axpy` but doesn't change significantly for compute-bound matrix multiplication.

C. Interference Roofline

We move on to extend the roofline model by measuring its behaviour under various injected bandwidths. As before, all TGs are running and we evaluate several frequencies. First, we illustrate the roofline changes under increasing PL traffic. Second, we highlight the execution time jitter impacting real-time correctness. The rooflines are found empirically by varying

the intensity of `stride` as outlined in Section III-B1. Here, we measured 1024 times to get stable results.

Figure 3a) displays the measured WCET of `stride`, thus showing the empirical lowest-performing roofline. The ridge point is shifted to the right when the injected bandwidth is increased. This means that some compute-bound programs become memory-bound and already memory-bound programs are subject to a large performance decrease. The numerical values are listed in Table III. Using the original roofline model, we determine the ridge point of the uninterfered roofline to be at an intensity of 0.172 flop/B, while in the interfered case it is at 5.09 flop/B. Looking at Table I, we see that `axpy` is memory-bound and suffers the most. `conv2d` and `bicg`, slip from the compute- into the memory-bound region. For all benchmarks, the measured WCET align well with the interference roofline.

In Figure 3b), we additionally display the best, median and worst performance for the rooflines with no and maximal interference. The increased jitter translates to system idle time, as task dimensioning considers the worst case only. Furthermore, the larger the increase in jitter, the more it affects the composability of the system, as discussed in Section II-A. Dividing the `max` roofline by the `min` in the memory-bound region, we obtain that jitter increases from 20% in the non-interfered case to $10\times$. With the use of this extended roofline, this effect can be easily compared between multiple platforms, assessing their suitability for the real-time domain.

D. Characterization Outcome

In the previous sections, we showed how to systematically characterize computing platforms for interference using the extended roofline model. This model gives insights about the performance degradation of a CPU program while accelerators access shared memory. Depending on the arithmetic intensity of a program, a slowdown of up to an order of magnitude is to be expected and mitigation techniques are required.

Such techniques involve code refactoring, like the Predictable Execution Model [6], [7], to perform batch-wise main memory accesses phases. Memory interference is eliminated altogether and the extended roofline model becomes irrelevant at the cost of scheduling to avoid phase overlaps.

However, our model can also be used with other techniques that limit the interference low-priority tasks inflict to high-priority ones, e.g. MemGuard [8], a mechanism that guarantees memory bandwidth reservation. More importantly, our approach can be used to understand which, if any, is the most suitable technique.

V. Related Work

Manev et al. [4] have characterized DRAM access from the PL side on the ZU9EG. The highest throughput from DRAM they could achieve was 75 % of the theoretical maximum using three out of four AXI ports, avoiding the usage of ports 1 and 2 simultaneously. We saw in Section IV-A that this was the most stable setup. Their focus is on benchmarking and they do not address interference in real-time contexts. Bansal et al. [3] have characterized the memory subsystem of the ZU9EG with interference generated on the PS. In contrast, our work generates interference coming from the PL, thus addressing the inverse part of the problem while keeping the perspective of the PS. Furthermore, we connect our findings to a model.

Restuccia et al. [5] observed that round-robin bus arbitration of transactions with heterogeneous sizes can result in unfair bandwidth distribution. They propose a module named *AXI burst equalizer* to address this problem and evaluate it on three different Xilinx boards, including the ZU9EG. While their work investigates the arbiter of a generic AXI interconnect, where interference is not necessarily a concern, our work focuses on the DRAM controller, which is a crucial point of interference.

Lee et al. [12] proposed a degradation predictor for heterogeneous systems based on regression models. Their approach is model-based, whereas ours is a measurement-based extension of the intuitive roofline model.

VI. Conclusion

To address the issue of real-time correctness in heterogeneous reconfigurable SoCs, interference needs to be understood and controlled to enable timing predictability.

Applying our proposed methodology to the ZU9EG, we measure a slowdown of up to $19\times$ on several real-world benchmarks depending on their complexities and memory access patterns. Moreover, we measure the degrading rooflines (up to $26\times$ slowdown) and their growing jitter (from 20 % to $10\times$) due to interference. Tracking the behaviour of the ridge point, we provide a performance degradation estimation guideline.

With our new interference roofline model, we can efficiently determine when countermeasures are necessary to mitigate performance degradation due to interference. The timing predictability can thus be controlled, paving the way for the use of powerful cost-efficient SoCs in real-time applications.

References

[1] H. Kopetz, "Real-Time Systems - Design Principles for Distributed Embedded Applications," in *Real-Time Systems Series*, 1997.

[2] S. Williams *et al.*, "Roofline: an Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, 2009.

[3] A. Bansal *et al.*, "Evaluating Memory Subsystem of Configurable Heterogeneous MPSoC," in *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2018.

[4] K. Manev, A. Vaishnav, and D. Koch, "Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems," in *International Conference on Field-Programmable Technology*, 2019.

[5] F. Restuccia *et al.*, "Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs," *ACM Transactions on Embedded Computing Systems*, 2019.

[6] R. Pellizzoni *et al.*, "A Predictable Execution Model for COTS-Based Embedded Systems," *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011.

[7] B. Forsberg, L. Benini, and A. Marongiu, "HePREM: A Predictable Execution Model for GPU-based Heterogeneous SoCs," *IEEE Transactions on Computers*, 2020.

[8] H. Yun *et al.*, "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms," *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2013.

[9] Xilinx, "AXI Traffic Generator v3.0 LogiCORE IP Product Guide," 2019.

[10] C. Lattner and V. S. Adve, "LLVM: a Compilation Framework for Lifelong Program Analysis and Transformation," *International Symposium on Code Generation and Optimization*, 2004.

[11] S. Grauer-Gray *et al.*, "Auto-Tuning a High-Level Language Targeted to GPU Codes," *Innovative Parallel Computing*, 2012.

[12] S. Lee and C. Wu, "Performance Characterization, Prediction, and Optimization for Heterogeneous Systems with Multi-Level Memory Interference," in *IEEE Internat. Symp. on Workload Characterization*, 2017.

Analyzing Memory Interference of FPGA Accelerators on Multicore Hosts in Heterogeneous Reconfigurable SoCs

Maxim Mattheeuws
IIS at D-ITET
ETH Zürich
Switzerland
mmaxim@iis.ee.ethz.ch

Björn Forsberg
IIS at D-ITET
ETH Zürich
Switzerland
bjoernf@iis.ee.ethz.ch

Andreas Kurth
IIS at D-ITET
ETH Zürich
Switzerland
akurth@iis.ee.ethz.ch

Luca Benini
IIS at D-ITET
ETH Zürich
Switzerland
lbenini@iis.ee.ethz.ch

***Abstract*—Reconfigurable heterogeneous systems-on-chips (SoCs) integrating multiple accelerators are cost-effective and feature the processing power required for complex embedded applications. However, to enable their usage in real-time settings, it is crucial to control interference on the shared main memory for reliable performance. Interference causes performance degradation due to simultaneous memory requests by components such as CPUs, caches, accelerators, and DMAs.**

We propose a methodology to characterize the interference to multicore host processors caused by accelerators implemented in the FPGA fabric of reconfigurable heterogeneous SoCs. Based on it, we extend the roofline model to account for performance degradation of the computing platform. The extended model allows to determine in an efficient way at which point memory interference becomes critical for a given platform and workload.

We apply our methodology to a modern Xilinx UltraScale+ SoC integrating a multicore ARM Cortex-A CPU and a Kintex-grade FPGA. To the best of our knowledge, our results experimentally show for the first time that programs with intensities below 5 flop/byte – workloads with low cache locality – can suffer from slowdowns of up to an order of magnitude.

***Index Terms*—interference, heterogeneous, SoC, CPU, FPGA, benchmarks, model**

I. Introduction

Modern FPGA-based SoCs feature the processing power to run embedded applications such as autonomous driving. However, these applications have real-time requirements that necessitate timing predictability. *Real-time correctness* is achieved only if the computational outputs are correct *and* delivered before their *deadlines* [1]. Thus, real-time systems are designed in accordance to their Worst-Case Execution Time (WCET) bounds. Deriving them tightly is problematic, as main memory is shared among computing units, leading to memory interference. To understand when this becomes a threat to real-time correctness is thus the main challenge to achieve predictability.

Modern reconfigurable heterogeneous SoCs, such as Xilinx UltraScale+ MPSoCs, are advertised with hardware support to increase memory arbitration control. We demonstrate that when DRAM is stressed by the FPGA, CPU workloads on such devices suffer considerable performance degradation and thus their WCETs increase. We develop a novel methodology extending the roofline model [2] to incorporate these results. Our approach can be used to determine when memory interference threatens real-time correctness. While memory behaviour on heterogeneous reconfigurable SoCs has been explored [3]–[5], our work is the first to tie it to a formal model.

Techniques to address the interference problem have been proposed [5]–[8]. Using our methodology, we can identify the point at which they become indispensable. Summarizing, this paper makes the following contributions:

1) We present a methodology to profile accelerator traffic injection into the memory system (Section III).
2) We observe up to $19\times$ slowdown of CPU programs on a FPGA-based SoC, which motivates the need for interference modeling and countermeasures (Section IV-B).
3) We develop a roofline-based model to characterize the impact of accelerator traffic on CPU execution, parameterizing the critical point for interference (Section IV-C).

II. Prerequisites

Before detailing our contributions, we present the roofline model as well as the ZU9EG SoC used for our experiments.

A. The Roofline Model

The roofline model provides the maximal attainable performance P [flop/s] of a program with arithmetic intensity I [flop/B] executing on a computing platform described by its peak performance π [flop/s] and bandwidth β [B/s]:

$$P = \min\left(\pi, \beta \cdot I\right) \tag{1}$$

By plotting the intensity vs. maximal attainable performance, this model describes the limitations either given by memory bandwidth or CPU performance. Equation 1 can thus be represented as in Figure 1. According to the model, a program is either *memory-bound* or *compute-bound*, depending on whether it is restricted by bandwidth (yellow in Figure 1) or peak performance (green). The *ridge point* determines the transition. The strength of this model is its unifying approach. However, it neglects memory interference of concurrent executions.

As we show in Section IV-C, interference can expand the memory-bound region. This affects the *composability* of the