Invited Paper: On the Granularity of Bandwidth Regulation in FPGA-Based Heterogeneous Systems on Chip

Gianluca Brilli
University of Modena and Reggio Emilia, Italy

Giacomo Valente
University of L'Aquila, Italy

Alessandro Capotondi
University of Modena and Reggio Emilia, Italy

Tania Di Mascio
University of L'Aquila, Italy

Andrea Marongiu
University of Modena and Reggio Emilia, Italy

Abstract

Main memory sharing in commercial, FPGA-based Heterogeneous System on Chips (HeSoCs) can cause significant interference, and ultimately severe slowdown of the executing workload, which bars the adoption of such systems in the context of time-critical applications. *Bandwidth regulation* approaches based on *monitoring* and *throttling* are widely adopted also in commercial hardware to improve the system quality of service (QoS), and previous work has shown that the finer the granularity of the mechanism, the more effective the QoS control. Different mechanisms, however, might exploit more or less effectively the available residual memory bandwidth, provided that the QoS requirement is satisfied. In this paper we present an exhaustive experimental evaluation of how three bandwidth regulation mechanisms with coarse, fine and ultra-fine granularity compare in terms of exploitation of the system memory bandwidth. Our results show that a very fine-grained regulation mechanism might experience worse system-level memory bandwidth exploitation compared to a coarser-grained approach.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Bandwidth Regulation, System-on-Chip, FPGA

Digital Object Identifier 10.4230/OASIcs.WCET.2024.5

Funding *Andrea Marongiu*: This work was supported by project AI4CSM, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007326. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Belgium, Czech Republic, Italy, Netherlands, Lithuania, Latvia, Norway. The work was also supported by the European Union under the NextGenerationEU Programme within the Plan "PNRR - Missione 4 "Istruzione e Ricerca" - Componente C2 Investimento 1.1 "Fondo per il Programma Nazionale di Ricerca e Progetti di Rilevante Interesse Nazionale (PRIN)" by the Italian Ministry of University and Research (MUR)". Project Title: "Simplifying Predictable and energy-efficient Acceleration from Cloud to Edge (SPACE)", Project code: E53D23007800006. MUR D.D. financing decree n. 959, 30th June 2023.

1 Introduction

Heterogeneous Systems-on-Chip (HeSoCs) coupling general purpose multi-cores and accelerators of various types are widely adopted across several application domains. Commercial off-the-shelf HeSoCs constitute a convenient and cheap solution to providing the necessary computing power to run modern software applications, but also pose novel challenges. Main interconnect and memory sharing, which is a key architectural trait of such products, causes a significant slowdown of the application tasks [6], ultimately barring their adoption in time-critical domains. Main memory bandwidth regulation strategies are being increasingly adopted in commercial products, to provide some degree of QoS control. Several research approaches have also been proposed in this area to improve the effectiveness of such techniques [8, 15, 20].

Focusing on FPGA-based based HeSoCs, previous work has shown that the combination of bandwidth *monitoring* and *throttling* mechanisms is key not only to providing QoS guarantees to the CPU workloads – in terms of maximum slowdown experienced – but also to allowing FPGA accelerators to effectively use the residual bandwidth (without slowing down the CPU beyond the tolerated QoS thresholds) [3]. The degree of coupling between *monitoring* and *throttling* is pivotal to achieving fine-grained QoS control, suggesting that the finer the granularity, the better the results. We observed that while this is certainly true in terms of control capability (how fast the mechanism can adapt to varying workload characteristics due to memory interference), a very fine-grained bandwidth regulation mechanism might adversely impact the behavior of the memory controller, ultimately resulting in worse system-level memory bandwidth exploitation.

In this paper we study and compare three state-of-the-art mechanisms for joint bandwidth monitoring and throttling: (i) a coarse-grained, software-based approach that can be implemented on top of widely available HW performance counters; (ii) a fine-grained, FPGA-based, integrated runtime bandwidth regulator (RBR [3]); (iii) an ultra fine-grained, interconnect-level hardware approach, available in a number of commercial HeSoCs (ARM QoS-400 [1]). We conduct an extensive experimental evaluation on a representative FPGA-based HeSoCs, the AMD/Xilinx Zynq Ultrascale+, using real benchmarks: the Polybench benchmark suite [18]. Our setup measures the amount of residual bandwidth that various bandwidth regulation mechanisms allow the FPGA accelerators to exploit, while maintaining the slowdown of CPU programs within the tolerated QoS requirements. Experimental results show that the best results are achieved for the medium-granularity approach, while the ultra fine-grained one is often surpassed even by the coarse-grained one in terms of exploitation of the system bandwidth. We thus present a more in-depth investigation of the problem, that highlights how idleness insertion at a very fine-grained level[1] triggers worse DRAM controller behavior than a slower regulation mechanism.

2 Related Work

Memory interference significantly impacts the performance of modern HeSoCs. This has driven extensive research in recent years, examining its effects on various components such as the main CPU [7], GPGPU accelerators [5, 19], and FPGA accelerators [4, 12].

Memory bandwidth regulation is a practical and effective technique typically adopted on Commercial Off-the-shelf (COTS) HeSoCs. This method is essential for providing guarantees on application Quality of Service (QoS) and formitigating issues related to memory contention. Providing an accurate memory bandwidth regulation reduces contention and enforces execution time predictability, particularly in scenarios where multiple applications with diverse and competing bandwidth demands are executed simultaneously [17, 20, 22].

[1] QoS-400 operates at the granularity of a small number of back-to-back transactions (or *beats*), 16 on the target hardware, equivalent to 256 bytes only.

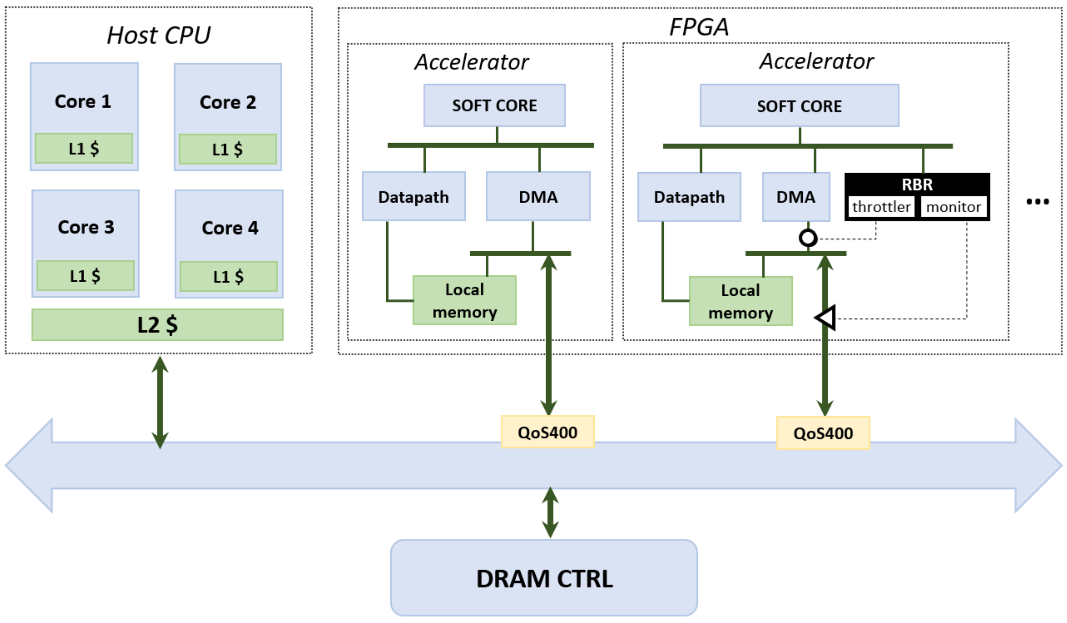


Figure 1 Architectural template of the target HeSoC.

Several bandwidth regulation techniques exist both from hardware and software perspectives. Considering software-based memory bandwidth regulators, Yun et al. proposed *MemGuard* [20] a memory bandwidth throttler that is based on a joint action between bandwidth monitoring (using core performance counters) and a software throttling mechanism based on interrupts. Controlled Memory Request Injection (CMRI) has been originally proposed as a software-based bandwidth regulation technique, to regulate CPU-based workloads [7] and in a more complex setup where also FPGA-based accelerators are involved [4]. All these software-based bandwidth regulators suffer of non-negligible overheads due to software interactions between loosely-coupled monitoring and throttling components.

Finer bandwidth regulation mechanisms could be designed by leveraging tighter hardware components to implement bandwidth regulation. For example, Zuepke et al. proposed *MemPol* [22] a hardware memory bandwidth regulator that can regulate application cores with $6.25\mu s$ granularity, where the throttling phase is implemented using a hardware debugging interface. Similarly, Farshchi et al. proposed a hardware-based fine-grained regulator for application cores evaluated on the *FireSim* simulator [8]. A complementary hardware approach has been proposed to regulate FPGA-based accelerators with a similar granularity [3]. Several other works try to understand and effectively exploit the available hardware knobs for controlling QoS and regulating memory bandwidth at different interconnect levels and memory hierarchy. As it was shown in recent works, using these knobs is typically not straightforward, due to the varying degrees of support on different products and to the many different (and in some cases obscure) configurations available [9, 16, 21]. Furthermore, these mechanisms lack generality, as they are typically closed solutions specific to a given vendor or hardware platform (e.g., ARM MPAM [14], ARM QoS-400 [1]).

Although it is known that a fine-grained mechanism adapts better to bandwidth variations, our work proposes a preliminary investigation that highlights that, in some conditions, using a coarser-grained bandwidth regulator (i.e. even software-based), could better utilize the bandwidth from the FPGA, given a QoS requirement on a task running on CPU cores.

3 Target Architecture and Bandwidth Regulation Schemes

We consider the architectural template of an FPGA-based HeSoC shown in Figure 1, which is composed of a *host* multi-core CPU coupled to an FPGA subsystem. The two subsystems communicate via the main DRAM. This template captures the main traits of several existing commercial products. Within the FPGA, one or more *accelerators* are deployed. A generic template for an *accelerator* includes a *datapath*, namely the core logic that performs the computation, and an efficient *DMA engine*, used to facilitate the staging of data from the DRAM into a fast, local memory. To simplify the development of FPGA applications it is common to also enrich the accelerator template with a *soft core* for local control of the *datapath* and *DMA* operation, without the need for the costly intervention of the main CPU [2, 10, 11, 13].

In modern HeSoCs, the DMAs inside FPGA accelerators generate much higher DRAM bandwidth request than what happens on the CPU cores, and more than a single master port is typically available to individually attach accelerators to the main interconnect fabric (for example, the AMD/Xilinx Ultrascale+ device that we use for our experimental setup features three independent ports). If CPU cores and FPGA accelerators run in parallel without DRAM access control, the execution time of the CPU tasks can slow down by over 10× [12]. On the other hand, enforcing mutually exclusive CPU/FPGA DRAM accesses causes severe under-utilization of the available memory bandwidth. Since the main interface of an *accelerator* to the DRAM is the *DMA*, previous work has shown that bandwidth monitoring and throttling can efficiently happen at this level [3].

3.1 Bandwidth Regulation Schemes

In this section we describe three bandwidth regulation mechanisms that rely on integrated monitoring and throttling cycles, ranging from coarse-grained, full-SW solutions to ultra-fine-grained, full-HW solutions.

3.1.1 SW-DMA

Previous research has investigated the throttling of FPGA *accelerators* by utilizing *soft cores* to program the DMA in a duty-cycled loop [4]. Each DMA transfer request is divided into multiple smaller transfers, which can be interspersed with a programmable amount of $idle_{cycles}$, computed as shown in Eq. (1):

$$idle_{cycles} = \frac{100 - THR_{\%}}{THR_{\%}} * copy_{cycles} \tag{1}$$

Here, $THR_{\%} \in [1, 100]$ denotes the throttling factor[2] applied to the transfer, and $copy_{cycles}$ is the number of clock cycles required to complete the smaller transfer. A disadvantage of throttling accelerators via software (even when executed on the local *soft core*) is the substantial programming overhead, which prevents very fine-grained operation.

3.1.2 Runtime Bandwidth Regulator

The Runtime Bandwidth Regulator (RBR) [3] has been proposed as a non-intrusive component introduced in the the accelerator template, as shown in Figure 1. It contains two main blocks: (i) a monitor that probes the outgoing channel to the DRAM to unobtrusively measure

[2] THR=100 corresponds to 100% bandwidth, while THR=1 corresponds to 1% bandwidth.

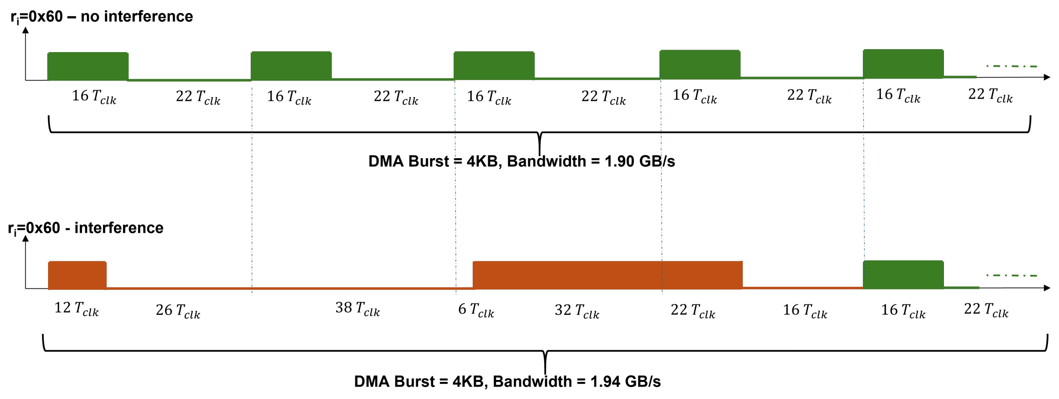


Figure 2 Waveforms of a memory transaction generated from an accelerator, composed of 256 beats and regulated using QoS-400 with r_i = 0x60. The red and green waveforms represent the case with and without memory interference.

the time (copy cycles from Eq. (1)) to transfer a given amount of bytes. The size of this transfer defines the *granularity* of the technique; (ii) a throttler that computes the idle cycles from Eq. (1) and stops DMA operation for that amount of time. The *granularity* and the throttling factor (THR) can be dynamically (re)configured at any time by the software.

The tight coupling between the monitor and the throttler in the RBR guarantees very fast QoS regulation, which is convenient in presence of dynamically varying QoS requirements.

3.2 ARM QoS-400 regulator

The ARM CoreLink QoS-400 regulator [1] is a hardware component designed to manage bus traffic generated from various actors sharing main memory on the HeSoC. The official ARM documentation does not provide detailed descriptions of the QoS-400 regulator behavior, nor does it specify the granularity of the regulation. By means of a thorough experimental characterization of the QoS-400 behavior, it is possible to observe that the regulation operates with a fixed, very fine granularity.

Fig. 2 shows an example of how the QoS-400 manages long outstanding transactions from a DMA in absence of interference (top plot) and in presence of interference (bottom plot). The example DMA transfer is a burst of 4096 bytes. The burst is split in 256 *beats*, as the physical size of the channel is 16 bytes. In absence of interference, the QoS-400 steadily fragments the 256-beat transfer into blocks of 16 beats each, followed by a idle period whose length is determined according to Eq. (1). In presence of interference, it is evident that the QoS-400 is capable of adapting, employing bandwidth monitoring and adjusting the throttling. From the plot on the bottom we can see that the QoS-400 detects the bandwidth drop due to interference and maintains the desired QoS level by adjusting the throttling . By allowing twice the number of beats to be transmitted in the third sub-transaction, the bandwidth is maintained at the target value.

4 Experimental Results

This experimental section aims to analytically study the behavior of the different bandwidth controllers under investigation. Previous work has highlighted already that finer-grained approaches are more effective at guaranteeing the desired QoS levels in presence of fast dynamic changes in the QoS requirements or/and in the traffic characteristics [3]. However, different approaches are more or less effective at redistributing the bandwidth unused by the actors executing the QoS-constrained workload to the remaining (best-effort) actors. Our experiments focus on analyzing the capability of each method of redistributing the unused bandwidth, provided that they all are configured to always meet the QoS requirements. Further, the experiments examine how the different architectural designs of the controllers and their inherent operational granularities impact their performance and the level of interference they generate in the system.

4.1 Experimental Setup

The experimental evaluation is conducted on an AMD/Xilinx Zynq Ultrascale+, *XCZU9EG* HeSoC. The accelerator template for the SW-DMA and RBR mechanisms was modeled after the setup described in [3] [4], using AMD/Xilinx IPs for the DMA, soft-core and interconnects. To capture worst-case interference effects, we instantiate three accelerators (one per DRAM controller port), each of which is configured to operate as a generator of steady R/W traffic[3] [12]. The resulting design was synthesized with a target frequency of 300 MHz using AMD/Xilinx Vivado 2020.2.

In our setup, the CPU is considered the actor with a QoS constraint, while the FPGA accelerators are the best-effort actors. As it is often done in the literature [16, 17, 22], we consider two fixed thresholds to the maximum tolerated QoS degradation: 10% and 20% slowdown with respect to non-interfered execution. We measure the maximum amount of memory bandwidth that FPGA accelerators can utilize without causing performance degradation in software applications that surpass the aforementioned QoS thresholds. As a target workload executing under the described QoS constraints, we execute 31 different applications from the Polybench benchmark suite [18].

4.2 Exploitable Residual Bandwidth Evaluation

Figure 3 describes how the experimental evaluation is conducted. One such plot is derived for every benchmark. The X-axis represents the overall throttling factor $\mathrm{THR}_{\%}$ applied to the three FPGA accelerators. Note that the percentage shown on the X-axis refers to the cumulative bandwidth used by the three accelerators. Thus, a $\mathrm{THR}_{\%} \leftarrow 100\%$ implies that the *i*-th accelerator is configured with one-third of the total $\mathrm{THR}_{\%}$ (i.e., $ACT_i = 33.3\%$ $\mathrm{THR}_{\%}$). A $\mathrm{THR}_{\%} \leftarrow 100\%$ corresponds to utilizing the entire memory bandwidth of the FPGA, as denoted by a vertical dotted line. The red area depicts the bandwidth used by the FPGA accelerators and refers to the left Y-axis of the plot. The horizontal black curves mark the two tolerated CPU slowdowns (10% and 20%) and refer to the right Y axis of the plot. Two black arrows originate at the points where the measured CPU slowdown curve (the one with red markers) intersects the horizontal black curves and are projected up vertically to the point where they intersect the red area. The latter intersection points indicate the memory bandwidth used by the FPGA for the target regulation mechanism under both QoS constraints (10% and 20%). Intuitively, a higher FPGA bandwidth means a better capability of redistributing the unused bandwidth by the CPU to the FPGA accelerators.

These memory bandwidth values are plotted in Figure 4 for all the PolyBench benchmarks. Subplots a) and b) refer to 10% and 20% max QoS degradation, respectively. The RBR regulator generally allows for greater bandwidth usage compared to other regulators, both at

[3] Note that this is without loss of generality, as well-designed accelerators overlap memory transactions with computation.

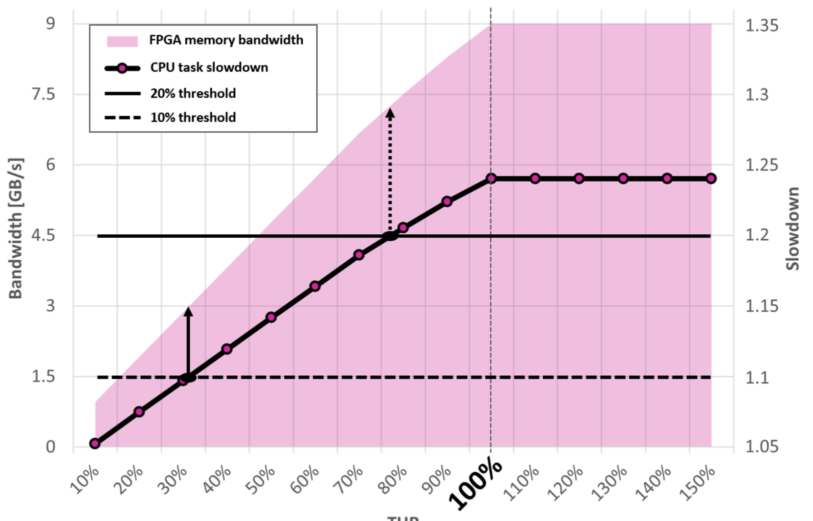


Figure 3 Application slowdown and FPGA memory bandwidth varying the THR$_{\%}$ parameter. The intersections between the slowdown curve and the two horizontal lines determine the maximum tolerated FPGA bandwidth.

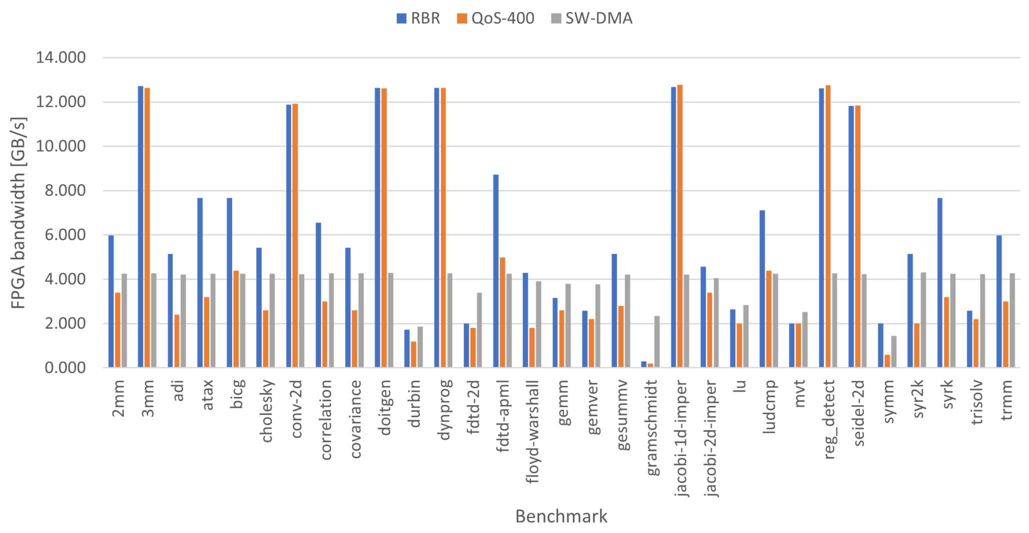
10% and 20% QoS thresholds. This is evident in Figure 5, where we summarize the average memory bandwidth utilization of the 31 benchmark kernels, normalized to the memory bandwidth achieved by the RBR mechanisms. Medium-grained bandwidth regulation (RBR) allows the exploitation of +37.57% and +16.08% more of the memory bandwidth compared to SW-DMA and QoS-400, respectively, for the 10% QoS requirement. A similar improvement is also seen when relaxing the QoS to 20%. In that case, the RBR can allow +35.45% and +8.54% higher memory bandwidth utilization.

This advantage is particularly pronounced when stringent regulation is necessary, such as when FPGA bandwidth falls below 10 GB/s. Looking at individual benchmarks, however, it might come as a surprise that the SW-DMA sometimes offer better residual bandwidth exploitation than QoS-400 or even RBR. This seem to happen in scenarios where the residual bandwidth is low (below 6 GB/s), indicating that the CPU benchmark is very sensitive to interference. It appears that a slower idleness insertion mechanism in these situations can better exploit the residual bandwidth.

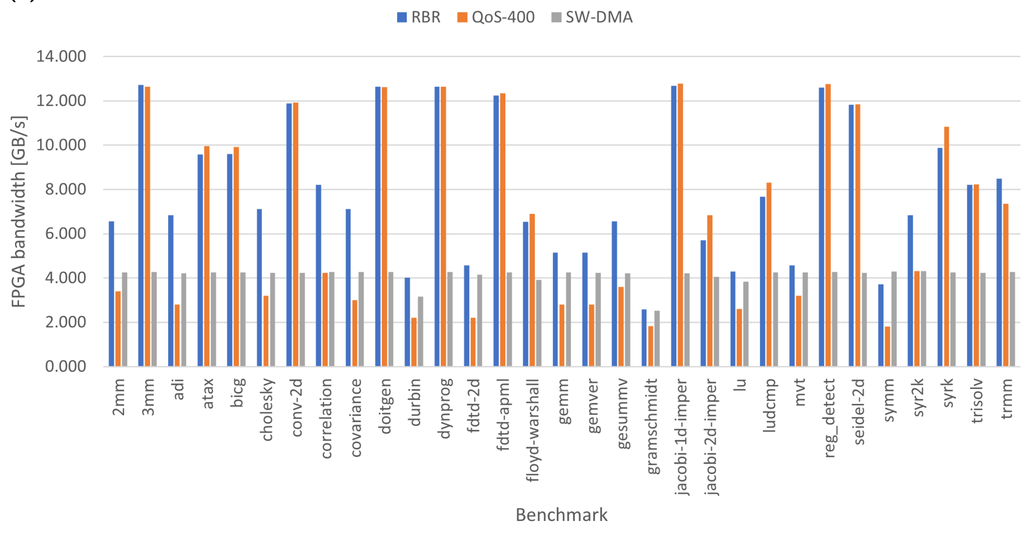
To confirm this intuition we conduct a new experiment, where we focus on a synthetic benchmark, running on the CPU, which performs only DRAM read accesses with a 100% L2 cache miss rate (maximum DRAM bandwidth requirement).

Figure 6 illustrates the usable bandwidth for FPGA accelerators (colored area, left Y-axis) and the respective slowdown experienced by the application under test (black line, right Y-axis) for each regulator type: *fine-grain* QoS-400, *medium-grain* RBR, and *coarse-grain* SW-DMA, as the THR varies (X-axis). Analyzing the FPGA bandwidths at different THR levels, it is evident that the QoS-400 and RBR controllers can both precisely control the amount of DRAM bandwidth used by the FPGA accelerators under identical THR configurations, peaking at 8.6 GB/s. In contrast, the SW-DMA controller can only utilize half of that bandwidth (4.2 GB/s) due to the overhead introduced by the software-based monitoring and control loop between different data transactions.

If we now focus on the impact on the application under test, we can notice that given any THR configuration, the CPU traffic suffers a significantly higher slowdown under QoS-400 regulation compared to RBR. Using RBR the FPGA can exploit up to 1.73 GB/s and 3.16



(a) 10% slowdown increase.



(b) 20% slowdown increase.

Figure 4 FPGA memory bandwidth usage given a maximum tolerated slowdown (X% increase) of each computational kernel of the PolyBench benchmarking suite.

GB/s of DRAM bandwidth at QoS levels of 10% and 20%, respectively. In contrast, the more fine-grained QoS-400, only exploits residual bandwidth of 0.40 GB/s and 1.40 GB/s for the same QoS settings (over 50% worse than RBR). Surprisingly, the coarse-grain controller SW-DMA allows for a higher bandwidth exploitation than the other two controllers, meeting the QoS requirements at 2.35GB/s and 3.81GB/s. This preliminary investigation thus indicates that using the finest bandwidth regulation component (e.g., the QoS-400) is not always beneficial for maximizing memory bandwidth utilization, especially when the CPU has a strict slowdown requirement.

The intuition is that a very fine-grained memory bandwidth regulator imposes smaller regulation intervals on the memory buses, resulting in reduced opportunities for the CPU to interleave software-generated memory transactions on the memory controller. The intuition seems to be confirmed by simply doubling the regulation period of the RBR mechanism and observing that it can use higher residual bandwidth.

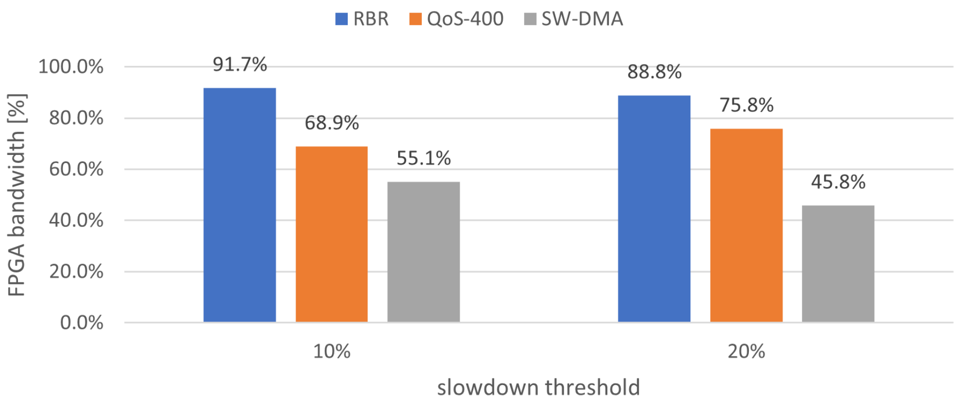


Figure 5 Average FPGA memory bandwidth utilization while the PolyBench applications are subject to 10 and 20% of slowdown increase.

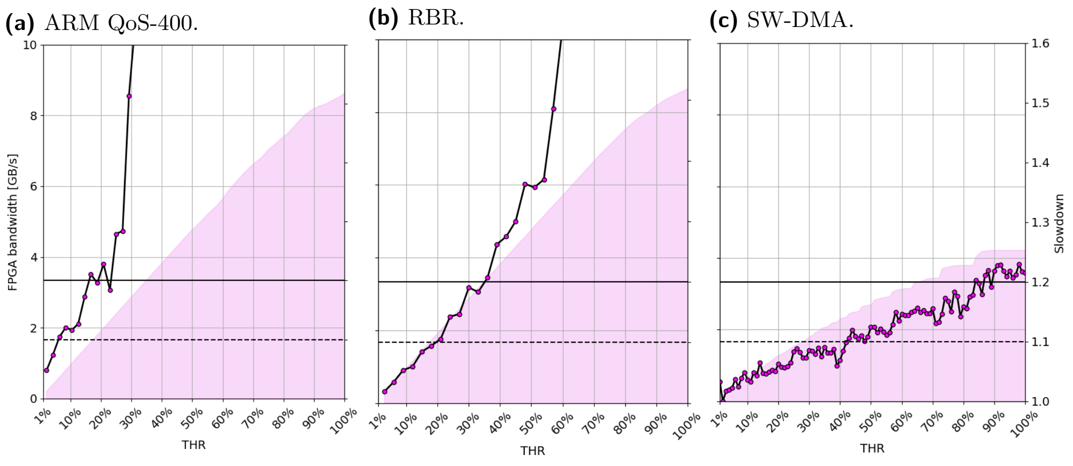


Figure 6 Synthetic benchmark that executes 100% cache read-miss operations, while the FPGA accelerators are regulated using the three different bandwidth regulators.

A coarser granularity has clearly the downside of making the approach slower to adapt to dynamically varying QoS requirements or traffic characteristics [3], so the sweet spot changes across different applications and hardware platforms. This insight opens the door to studying novel, potentially dynamic solutions that exploit different regulation granulates optimized for allowing the system to use the maximum performance from the HeSoC in all operational scenarios (e.g., traffic patterns, QoS levels, etc.), which is the focus of our ongoing work.

5 Conclusion

Bandwidth regulation mechanisms based on integrated monitoring and throttling are being increasingly used both in commercial products and in research, as they allow to mitigate the unpredictable behavior of HeSoCs where several CPUs and accelerators share the main memory. Although previous work has already highlighted that fine-grained bandwidth regulation allows for very precise QoS control and fast adaptation to varying QoS requirements and traffic characteristics, our preliminary investigation indicates that using the finest bandwidth regulation is not always beneficial for maximizing residual memory bandwidth utilization, especially when the CPU has a strict slowdown requirement. Configurable-granularity approaches like RBR can be exploited to devise software policies to exploit the sweet spot between fast control and maximal residual bandwidth exploitation, dynamically adapting at runtime to the characteristics of the workload and target hardware.

References

1 ARM. *CoreLink QoS-400 network interconnect advanced quality of service*, 2016. Accessed: November 5th, 2024. URL: `https://developer.arm.com/documentation/dsu0026/latest/`.

2 Gianluca Bellocchi, Alessandro Capotondi, Francesco Conti, and Andrea Marongiu. A RISC-V-based FPGA Overlay to Simplify Embedded Accelerator Deployment. In *24th Euromicro Conf. on Digital System Design*, pages 9–17, 2021. `doi:10.1109/DSD53832.2021.00011`.

3 G. Brilli, G. Valente, A. Capotondi, P. Burgio, T. Di Masciov, P. Valente, and A. Marongiu. Fine-grained qos control via tightly-coupled bandwidth monitoring and regulation for fpga-based heterogeneous socs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. `doi:10.1109/DAC56929.2023.10247840`.

4 Gianluca Brilli, Alessandro Capotondi, Paolo Burgio, and Andrea Marongiu. Understanding and Mitigating Memory Interference in FPGA-based HeSoCs. In *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1335–1340, 2022. `doi:10.23919/DATE54114.2022.9774768`.

5 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)*, pages 1–10, 2020. `doi:10.1109/RTCSA50079.2020.9203722`.

6 CAST. *Position Paper CAST-32A Multi-core Processors*, 2016. Accessed: November 21st, 2021. URL: `https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/media/cast-32A.pdf`.

7 Roberto Cavicchioli, Nicola Capodieci, Marco Solieri, Marko Bertogna, Paolo Valente, and Andrea Marongiu. Evaluating Controlled Memory Request Injection to Counter PREM Memory Underutilization. In *Workshop on Job Scheduling Strategies for Parallel Processing*, page 85. Springer, 2020.

8 Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. In *2020 IEEE Real-Time and Emb. Tech. and Applications Symposium (RTAS)*, pages 364–375, 2020. `doi:10.1109/RTAS48715.2020.00011`.

9 Sergio Garcia-Esteban, Alejandro Serrano-Cases, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. Quasi Isolation QoS Setups to Control MPSoC Contention in Integrated Software Architectures. In Alessandro V. Papadopoulos, editor, *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:25, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ECRTS.2023.5`.

10 Xiaoyi Ling, Takahiro Notsu, and Jason Anderson. An Open-Source Framework for the Generation of RISC-V Processor + CGRA Accelerator Systems. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pages 35–42, 2021. `doi:10.1109/DSD53832.2021.00015`.

11 Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. Agile SoC Development with Open ESP: Invited Paper. In *2020 IEEE/ACM Inter. Conf. On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

12 Maxim Mattheeuws, Björn Forsberg, Andreas Kurth, and Luca Benini. Analyzing Memory Interference of FPGA Accelerators on Multicore Hosts in Heterogeneous Reconfigurable SoCs. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1152–1155. IEEE, 2021.

13 Hossein Omidian, Nick Ivanov, and Guy G.F. Lemieux. An Accelerated OpenVX Overlay for Pure Software Programmers. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 290–293, 2018. doi:10.1109/FPT.2018.00056.

14 Andrea Pellegrini. Arm Neoverse N2: Arm's 2 nd generation high performance infrastructure CPUs and system IPs. In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–27. IEEE, 2021.

15 Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, Giorgiomaria Cicero, and Giorgio Buttazzo. AXI HyperConnect: A Predictable, Hypervisor-level Interconnect for Hardware Accelerators in FPGA SoC. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi:10.1109/DAC18072.2020.9218652.

16 Alejandro Serrano-Cases, Juan M Reina, Jaume Abella, Enrico Mezzetti, and Francisco J Cazorla. Leveraging hardware QoS to control contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

17 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. E-WarP: A System-wide Framework for Memory Bandwidth Profiling and Management. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 345–357, 2020. doi:10.1109/RTSS49844.2020.00039.

18 The Ohio State University. PolyBench-ACC/C the Polyhedral Benchmark suite. https://github.com/cavazos-lab/PolyBench-ACC, 2011.

19 H. Wen and W. Zhang. Interference Evaluation In CPU-GPU Heterogeneous Computing. *IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

20 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Emb. Tech. and Applications Symposium (RTAS)*, pages 55–64, 2013. doi:10.1109/RTAS.2013.6531079.

21 Matteo Zini, Giorgiomaria Cicero, Daniel Casini, and Alessandro Biondi. Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms. *Software: Practice and Experience*, November 2021. doi:10.1002/spe.3053.

22 Alexander Zuepke, Andrea Bastoni, Weifan Chen, Marco Caccamo, and Renato Mancuso. MemPol: Policing Core Memory Bandwidth from Outside of the Cores. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 235–248, 2023. doi:10.1109/RTAS58335.2023.00026.