Bounding inter-core interference with a hardware quota mechanism

Pablo Andreu*, Carles Hernandez*,
Pedro Lopez*

* *Universitat Politecnica de Valencia (UPV). Departamento de Informatica y sistemas de computadores, Camino de Vera, S/N 46022, Valencia, Valencia Spain*

ABSTRACT

Since the beginning of computing, there has been a race to obtain the most performant systems. This race led the industry into multithreaded applications and multicore systems. Unfortunately, for safety-critical systems, the required functional certification guarantees limited the usage of multicore chips due to the vast amount of unpredictable shared resources that these chips require.

However, there has recently been an increased need for safety-certifiable high-performant systems with the rise of AI and autonomous driving features. To obtain a safe and performant system, we need to partition safe and performant worlds into different cores with low resource sharing.

This work aims to ease the control of contention that different cores generate on a shared bus by providing a mechanism to stop offending cores from accessing the bus once they reach a certain contention threshold. Thus, guaranteeing that the system will meet all calculated timing constraints.

KEYWORDS: QoS; Mixed criticality; Shared buses; Computer architecture

1 Introduction and motivation

In the safety-critical world, the usage of multicore chips has been limited due to the complexity of limiting the impact of multicore interference on shared resources. However, in the embedded world, AI computations that stress the shared resource system are needed in increasingly more common systems, such as cars or robot vacuums. Creating a high-performance system where one can mix dependable critical tasks with non-critical or best-effort tasks without compromising the guarantees of such system is the purpose of this work.

Static cache partitioning and strict bus arbitration policies are classically implemented to have a performant system while maintaining guarantees on the critical cores. The most predictable bus design is time division multiplexing (TDM), where the bus is statically split into time partitions and granted to each master as if it was the only one present on each time partition. However, with TDM systems, there is a lack of flexibility, and maximum throughput suffers from fulfilling the need for predictability.

To mend this lack of total system throughput, less strict approaches such as the BRU [FHY20] have been proposed. The BRU presents a bandwidth regulation unit that pretends to increase timing guarantees for the critical cores by throttling secondary core bandwidth to the shared resources. The downfall of this approach is that conservative throttling might lead

[1]E-mail: {pabance,carherlu,plopez}@upv.es

to unused resources to increase the predictability of the system, thus, potentially wasting system resources.

The MCCU [ea19] monitors inter-core contention and only takes corrective action once secondary cores infringe a configurable bound. This approach results in the system achieving its maximum throughput until safety guarantees are about to be lost. Then, the MCCU raises an interrupt and the operating system takes the appropriate corrective action. This approach presents several problems, such as deciding which actions to perform once an interrupt is raised and, if offending core stopping is needed, how to prevent core starvation.

In this paper, we developed the hardware quota mechanism to solve these issues. With this mechanism, once a given contention threshold on the shared bus is reached, we stop offending cores' access to the bus.

2 Proposal

The hardware quota mechanism enforces interference quotas over the shared bus. These quotas act over secondary core interference to a primary core executing a critical workload. Contention quotas are much more efficient at sharing resources than bus access quotas, as they take into account only the requests that slow down the primary core workload.

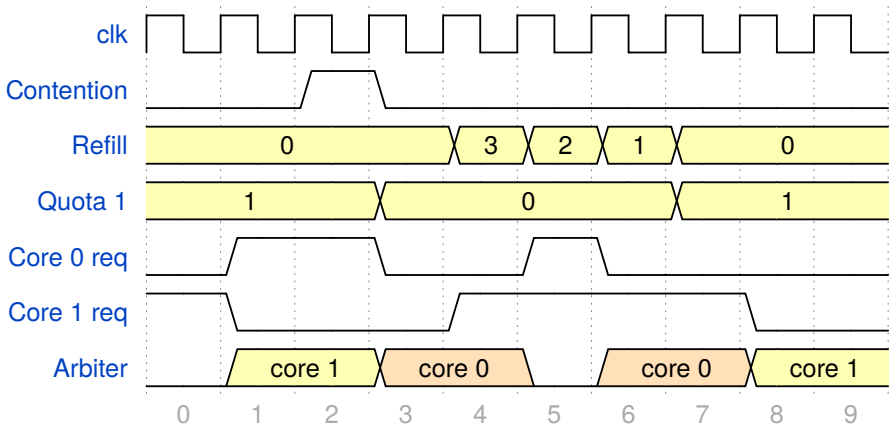


Figure 1: Timing diagram of hardware quota and hardware refill mechanisms

The basic quota mechanism can be seen on Figure 1 cycles 0-6. Core 1 blocks core 0 from accessing the bus on cycles 1 and 2, thus, depleting core 1 contention quota and starving core 1 from accessing the shared bus on cycles 4-8. On cycle 7, quota of core 1 is restored so it can re-access the bus. This restoration can happen in two ways: a software checkpoint in the program, where the critical core sets the allowable contention for the next period, or via the implemented hardware quota refill mechanism.

This mechanism refills the secondary core quota after some software-configurable cycles from its depletion. The amount of quota restored is also software-configurable. With this hardware refill mechanism, we can completely avoid having any software measures to enforce the timely execution of primary tasks and prevent the secondary tasks from starving on the bus.

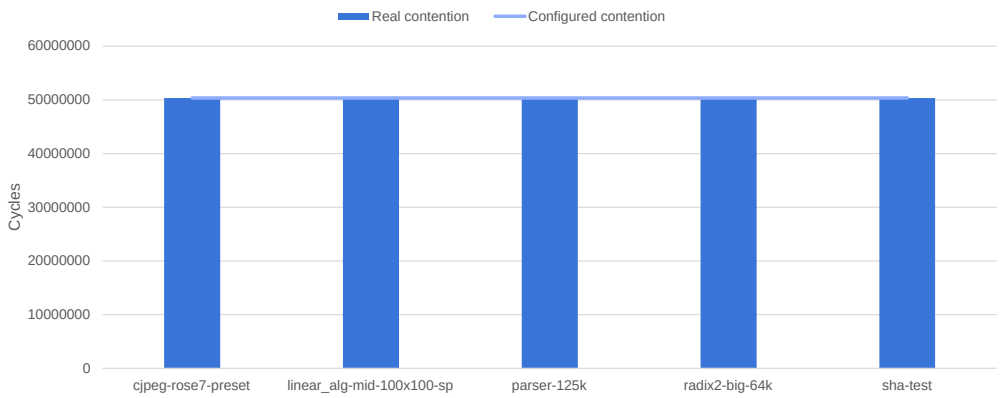


Figure 2: Contention caused in the bus for a quota of 50331645 (0x00ffffff per core)

3 Results

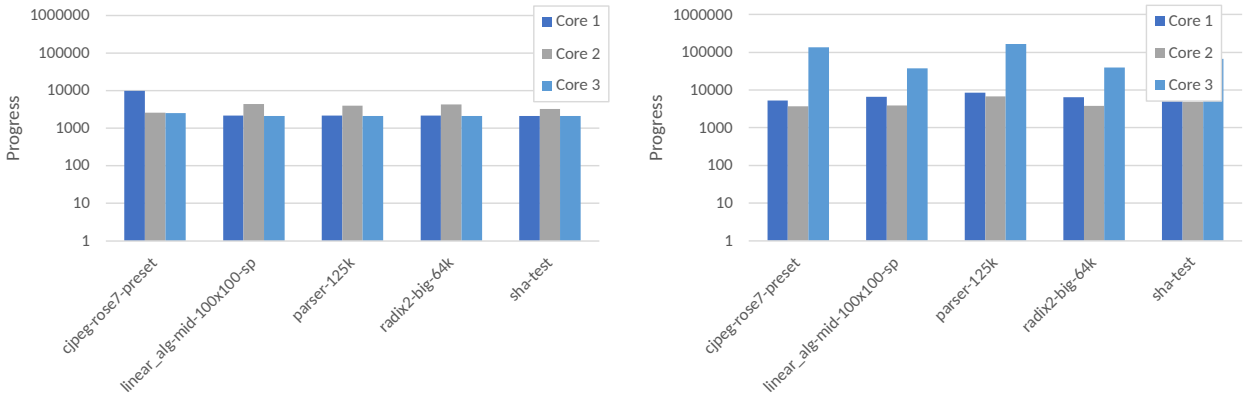
In this section we evaluate the results of hardware enforced contention quotas on a four core system. This system is implemented on an FPGA with a 4-core configuration of the GAISLER NOEL-V processors. In this system, the proposed mechanism is located on a AHB bus on the back-end of per-core private L1 caches and is used to access the shared L2 cache. Thus, once each core is prohibited from accessing the shared bus, it can still work with its private L1 data until access to L2 is necessary. Coherency is still maintained with this mechanism as bus snooping is used for L1 caches.

Figure 2 shows the contention caused in the bus for a given quota value for several applications. As it can be seen, the actual contention on the bus is effectively bound to the chosen quota. In fact, there is a negligible quota excess of 37 cycles on average and 68 cycles at max. This quota excess is due to our hardware enforcement mechanism acting on the bus arbitration phase; thus, we need to wait until the next arbitration phase for it to take effect.

Thus, with our proposed quota mechanism, we can effectively bound the contention that primary cores can experience on the shared bus. Once that primary core interference is bound, we can analyse the impact of the different bus arbitration policies and quota spreading mechanisms on the throughput of secondary cores to achieve the most work for the same quota. To do so, we use a metric called progress. The progress metric is the result of secondary cores reading a value from memory, incrementing it, and storing it back in memory, this causes read and write contention on the shared bus.

Figure 3 plots the progress of secondary cores for Round Robin and priority-based bus arbitration policies for the test system with the proposed quota implementation and without the hardware refill extension. This figure determines that the static priorities bus arbitration policy, which gives more priority to the lower index core, is the one which achieves a higher system throughput for the test system.

Figure 4 uses the progress metric to analyse the impact of the hardware refill mechanism on the progress of the hardware-quota limited cores running the selected application. It shows that, for the same amount of quota, the hardware refill mechanism obtains an order of magnitude more throughput than the simple hardware quota mechanism with both arbitration policies. It achieves this increase in throughput without decreasing the functional properties of the system.



(a) Round Robin

(b) Priorities

Figure 3: Progress of secondary cores in a quota mechanism with quota set to 0xFFFF

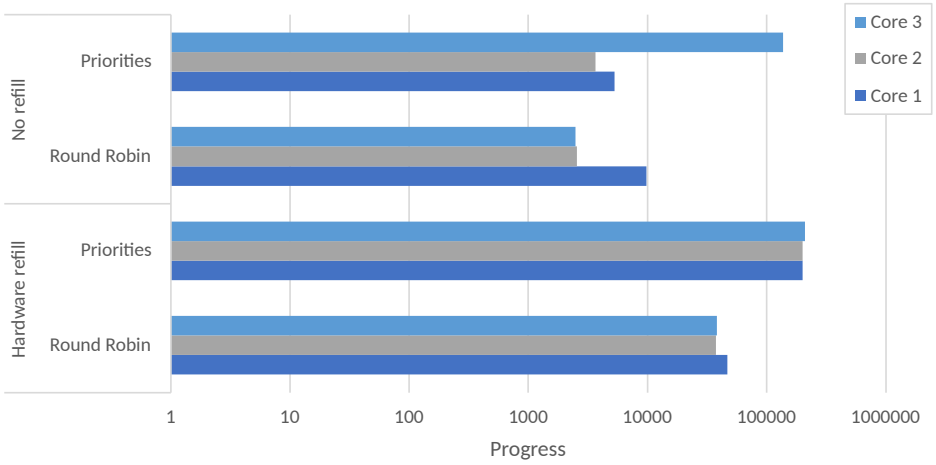


Figure 4: Progress with hardware refill and without refill for cjpeg-rose7-preset

4 Conclusions

The proposed hardware-enforced contention quota mechanism provides functional guarantees to critical cores in a mixed-criticality system while maintaining high system throughput, when hardware refill is implemented, throughput increases without guarantee degradation.

References

[ea19] Jordi Cardona et al. Maximum-contention control unit (MCCU): resource access count and contention time enforcement. In Jürgen Teich and Franco Fummi, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 710–715. IEEE, 2019.

[FHY20] Farzad Farshchi, Qijing Huang, and Heechul Yun. Bru: Bandwidth regulation unit for real-time multicore processors. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 364–375, 2020.