Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 11, Issue 8, 2022 | ISSN: 1000-3428

Evaluating Consumer Electronics Processors in Critical Real-Time Systems: Lessons from a Practical Exercise

Leonardo Marquez, Sofia Rodriguez, and Julian Sanchez

1 Department of Computer Science, University of California, Berkeley, USA,

Abstract—The convergence between consumer electronics and critical real-time markets has increased the need for hardware platforms able to deliver high performance as well as high (sustainable) performance guarantees. Using the ARM big.LITTLE architecture as example of those platforms, in this paper we report our experience with one of its implementations (the Qualcomm SnapDragon 810 processor) to derive performance bounds with measurement-based techniques. Our theoretical and practical analysis reveals that some hardware features may not suit critical real-time needs, and restricted specifications and buggy documentation creates serious difficulties to derive WCET estimates for software running on this platform. From the lessons learned, we identify several main elements to consider to effectively consolidate the sustained performance needs between mainstream and critical real-time markets.

I. Introduction

Increasing time-predictability needs of the consumer electronics market and increasing (guaranteed) performance needs of the critical real-time market push toward their convergence [7], [8]. In particular, consumer electronics market, which includes mobile phones and tablets, is expected to embed a growing number of critical functionalities related to monitoring and communication with other devices in the health, car, smart cities, Internet-of-Things domains. The critical real-time market, which includes avionics, automotive, robotic and healthcare applications, has also been shown to demand higher guaranteed performance to meet the needs of an increasing number of complex functions.

High-performance processors in the consumer electronics market are well known to pose difficulties to derive execution time bounds needed for critical real-time applications, while processors in critical real-time embedded systems offer (typically low) guaranteed performance. Therefore, while time-predictable high-performance processors have the potential to satisfy the needs of both markets, reconciling highperformance and predictability is a major challenge. Some issues have already been identified [4], [7], however, such convergence has not been explicitly studied for many popular processor architectures in the consumer electronics market. As an example, the ARM big.LITTLE architecture used in many tablets and mobile phones, as well as in safety support systems in commercial automotive solutions (e.g. Renesas R-Car H3 [12]), has not been fully analysed against the requirements of critical real-time applications.

This paper makes an step in that direction by assessing whether multicore contention and execution time interference can be tightly predicted on the SnapDragon 810 processor, implementing the ARM big.LITTLE architecture. In particular, (1) We review the processor specifications and identify some key features for multicore contention analysis. Our analysis identifies how some features need to be configured, reveals missing detail information in the specification and provides hints on what specific elements need to be assessed quantitatively. (2) We make an attempt to tailor the methodology based on *signatures and templates* [5] to the SnapDragon 810 processor: this methodology, shown suitable for multicore contention estimation in the LEON4 architecture for the space domain, builds on tracing key processor events (e.g. cache misses). (3) Finally, we perform a quantitative assessment with appropriate microbenchmarks with known expected behaviour [6]. Our results show that the behaviour of several Performance Monitoring Counters (PMCs) is nonobvious and hard to correlate with experiments, thus defeating their use to model multicore contention tightly. Moreover, our empirical analysis reveals that documentation is erroneous in some critical elements.

Overall, while the SnapDragon 810 processor has many key ingredients for its use in critical real-time systems, our analysis reveals that some hardware features and documentation practices challenge its direct use for critical real-time applications. Moreover, some of these conclusions can be extrapolated to other processors based on the ARM big.LITTLE architecture, which suffer from the same limitations.

How to cite: Leonardo Marquez, Sofia Rodriguez, and Julian Sanchez (2022). Evaluating Consumer Electronics Processors in Critical Real-Time Systems: Lessons from a Practical Exercise. Journal of Computer Engineering, 11(8), 1–9. DOI: https://doi.org/10.5281/zenodo.19345734

II. Goal and Scenario

The goal of this work is assessing whether the SnapDragon 810 processor can be used in the context of critical real-time applications. We use the term critical real-time to refer to any hardware or software component with any time criticality need: either mission, business of safety related.

A. The Platform

The SnapDragon 810 processor is a Qualcomm implementation of the ARM big.LITTLE architecture used in several recent Sony Xperia mobile phones. It comprises abundant hardware events that can be tracked with PMCs, so conclusions obtained on this specific processor apply to several others in the consumer electronics market, especially those building upon ARM big.LITTLE architecture and those implemented by Qualcomm.

The architecture of the processor, shown in Figure 1, comprises 2 clusters (also referred to as processors according to ARM's nomenclature): an ARM Cortex-A57 cluster with 4 cores and an ARM Cortex-A53 cluster with 4 cores. A57 cores are high-performance cores with out-of-order execution, whereas A53 cores are low-power lower-performance in-order cores. Yet, the A53 cluster is already a relatively highperformance platform w.r.t. current microcontrollers in many critical real-time systems. Therefore, as a first step we analyse this cluster, whose cores are more amenable to current timing

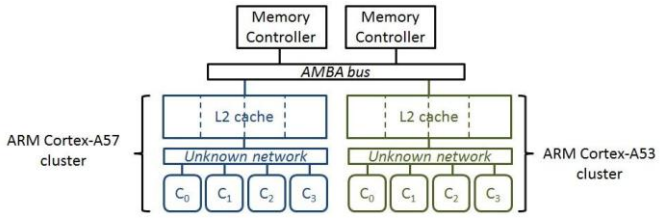

Fig. 1. Schematic view of the elements of the SnapDragon 810 processor analysed in this paper

analysis technology. Each A53 core is equipped with local first level instruction (IL1) and data (DL1) caches. Caches are connected to a shared L2 cache, local to the cluster. An AMBA bus interface connects both clusters to two shared memory controllers to access DRAM. Peripherals and accelerators, also present in this platform but not shown in the figure, are connected to the AMBA bus too. In this work we discount their effect by keeping them either disabled or idle.

Both clusters (A57 and A53) and the AMBA bus have been developed by ARM, the IP provider. Qualcomm, the chip manufacturer, integrates those components along with some others, which may or may not be provided by ARM. Moreover, in the integration process Qualcomm may introduce modifications in some IP components and/or their interfaces.

B. Tracing and Events

We focus on measurement-based timing analysis (MBTA), widely used in most real-time domains. For instance MBTA is used in avionics systems [10], [2], including those with DALA safety requirements [9] (though on top of much simpler single-core processors).

In the context of MBTA, tracing events impacting shared resource contention, e.g. cache misses, has been shown fundamental to derive bounds for a task not factoring in the worst potential contention but a specific contention level [5]. It follows that MBTA techniques demand more and more advanced hardware tracing mechanisms.

For that purpose we build upon the existence of PMCs to derive the type and number of accesses each task does, since this is needed to account for the contention a task can experience from (or produce on) others [5]. We also build upon microbenchmarks, i.e. small user level applications, that are able to create very high contention with each access type to the target shared resource [6].

III. Qualitative Analysis: Specifications

The main source of information for the analysis of the SnapDragon 810 processor is the ARM Cortex-A53 processor technical reference manual [1]. As detailed in the manual, a number of A53 features are regarded as 'implementation dependent', thus meaning that the processor manufacturer has the flexibility to choose among different options available. For instance, this is the case of the DL1, IL1 and L2 cache sizes. From the information available in the A53 manual, we regard as particularly relevant for contention analysis the following:

- The arrangement of the main components in the A53 cluster, including DL1, IL1 and L2 caches, as well as data prefetching features in DL1 and coherence support in L2.
- PMCs for events occurring in the cores (e.g. DL1 and IL1 caches) and in the L2 cache.

However, some parameters are not available in the A53 manual, including the following: (1) Timing characteristics of the interconnect between DL1/IL1 and L2 caches; (2) Specific characteristics of the different cache memories such as, for instance, their sizes; and (3) PMCs for events spanning beyond the A53 cluster such as accesses to the bus connecting A53 and A57 clusters with memory, and PMCs for the memory controllers. From a detailed analysis of each of those missing parameters for

real-time purposes in the A53 manual, we reached the following conclusions:

- The interconnect between DL1/IL1 and L2 caches, as the remaining in-cluster components, should be documented in ARM manuals. The lack of that information in the manuals makes us resort to software testing (e.g. microbenchmarks) to bring some light on the characteristics of this interconnect.
- Some instructions exist to read the particular characteristics of the cache hierarchy so they can be directly retrieved from the platform itself.
- PMCs and events beyond the A53 cluster should be documented in the SnapDragon 810 manual, since the processor manufacturer integrates those components, and so has access to the appropriate information for each component.

By the time we performed this work, ARM manuals were available, so we could retrieve them[1]. However, SnapDragon 810 manuals are neither publicly available in Qualcomm's website, nor included in the documentation coming along with the Intrynsic DragonBoard (whose processor is the SnapDragon 810), nor obtained upon request. In particular, while we requested appropriate manuals through Qualcomm public services as well as through internal contacts, and NDAs are in place, we were unable to get access to them. We are also aware of other companies in the critical real-time domain have experienced similar issues. Therefore, to the best of our knowledge, no information has been obtained on what PMCs/events exist beyond the A53 cluster and how they could be used. We also tried to use information from the ARM Juno development board, which is an ARM big.LITTLE implementation by ARM instead of by Qualcomm that offers further documentation but, as we suspected, ARM and Qualcomm implementations of this architecture differ and so Juno documentation did not help. From our analysis of the available information available, we have reached the following conclusions:

- The interconnect between DL1/IL1 and L2 can only be analysed empirically without specific guidance on its timing behaviour. The confidence on those measurements is limited due to the unknown specification of the interconnect.
- DL1, IL1 and L2 features can be directly obtained from the board via control instructions.
- Specific instructions exist to disable the data prefetcher. This is particularly relevant to discount uncontrolled (prefetcher) effects during operation.
- The L2 is inclusive with DL1 for coherence purposes. Thus, one core can create interferences on the DL1 of other cores by evicting their data from the L2 cache.
- The L2 cache cannot be partitioned across cores. This feature, together with L2 cache inclusivity, leads to potentially abundant inter-core interferences if not controlled by software means.

Listing 1. Structure of a microbenchmark

```
R1 = 0; for ( i =0; i<N; i ++) { r e s
 e t PMCs; for ( j =0; j<M; j ++) {
            . . .
     } read PMCs;
}
```

- PMCs up to the L2 cache exist and are abundant, but no information is had about PMCs beyond the L2 cache.

Overall, several cache features challenge the calculation of inter-core interference execution time bounds, and the lack of documentation for the DL1/IL1-L2 interconnect and PMCs for events beyond L2 challenge the confidence that can be obtained on measurement-based bounds. However, some information about contention can still be retrieved empirically based on information available. In the next section we present the results obtained.

IV. Quantitative Analysis: Experimentation

The number of hardware events that can be monitored in the SnapDragon 810 processor is limited according to ARM's documentation. For instance, while cache and memory accesses can be counted, it cannot derived whether DL1/IL1 and L2 cache accesses turn out to be hits or misses. This complicates the development of our methodology to measure the impact of contention in the access to shared resources.

A. Microbenchmarks

In order to access PMCs we have developed a library with the an interface to read/write PMCs. The main functions of the library include resetting/setting PMCs, activating/stopping PMCs, read/write PMCs and start/stop the Performance Monitoring Unit. To quantify the impact of contention in the access to the different shared resources, we have developed several microbenchmarks that stress each specific resource separately, in line with the method in [6]. This allows estimating the maximum delay that a request to a particular shared resource can suffer. Then this data is used to upper-bound contention impact. As starting point, we have developed microbenchmarks to account for contention in the access to the shared L2 cache and to

[1]

the shared memory controller, see their structure in Listing 1.

Since measurement can be polluted, e.g. by the Linux OS running below, we collect several (*N*) measurements and remove outliers keeping only the mode. The iterator *M* and the number of LOAD operations in the loop are set to values sufficiently high so that the overhead of the loop (i.e. the control instruction) and the overhead to fill the IL1 cache become negligible (e.g. *M* =1000 and 16 LOAD operations). The particular PMCs/events read and reset depend on the contention that is to be measured in a particular experiment. Finally, STRIDE relates to the distance between memory objects accessed so as to make sure that they either hit in L1, miss in L1 and hit in L2, or miss in L1 and L2. Vector size is properly set also with the same goal.

B. Disabling the Data Prefetcher

We disabled the data prefetcher so that read and write operations occurring in the different cache memories are

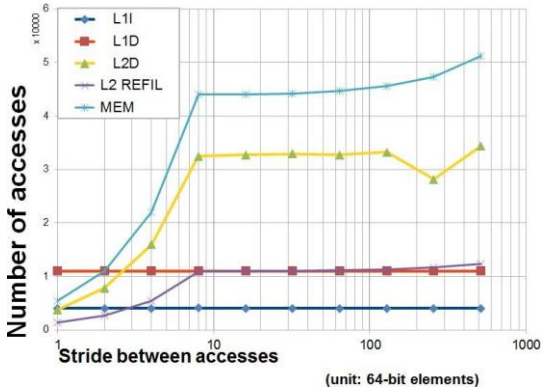

Fig. 2. Avg. number of IL1 (L1I), DL1 (L1D), L2 (L2D) and memory (MEM) accesses, and L2 refills per loop iteration for different data strides.

only triggered explicitly by the instructions executed in the microbenchmarks, rather than being automatically generated by hardware. For that purpose, we have configured the CPUACTLR register as described in the A53 manual [1]. Unfortunately, the execution of these commands leads to a system crash.

In order to verify the source of the problem, we repeated the same experiment on a PINE A64 [11] board. The PINE A64 platform is built with the aim of being a low-cost open source platform. It implements the same Quad-Core A53 Processor as the low-power SnapDragon 810 cluster. Thus, its interface is expected to be the same. In the PINE A64 platform, the commands to disable the prefetcher worked properly and subsequent experiments revealed that the data prefetcher was effectively disabled on that board. However, such board is a low-cost and low-power general-purpose computer, so the board itself is not oriented to the industry in the mobile market. Instead, it is an open platform. Thus, mobile industry will unlikely use it since there is no a large enterprise that provides support in the long term.

Overall, we could not disable the prefetcher in the SnapDragon 810. This problem likely relates to potential modifications introduced by the processor manufacturer, from which we did not succeed in obtaining the information required about the SnapDragon 810.

As a confirmatory experiment, we run a microbenchmark accessing 88KB of data, thus exceeding DL1 capacity (64KB) but fitting L2, with a 8B stride. Hence every 8 accesses we have 1 DL1 miss and 7 DL1 hits due to spatial locality (DL1 line size is 64B). With the prefetcher disabled, we would expect that the number of L2 accesses was 1/8 those in DL1. We observed that the number of DL1 accesses matches quite well our expectations, but the number of L2 accesses is roughly 0, revealing that the prefetcher is active and fetches data into DL1 reducing L2 accesses (the PMC for prefetch requests confirms this hypothesis).

C. Assessing Microbenchmark Results

In order to assess the behaviour of the PMCs in the A53 cluster, we have run our microbenchmark, which performs 11,000 load operations with a specific stride. This code is in a loop iterating 100 times, and we report average results across those 100 iterations to minimise the impact of cold misses in the first iteration and noise in the measurements.

We explore strides ranging between one 64-bit element (8 bytes) and 512 elements. With the smallest stride (8 bytes), we traverse a vector of ≈88*KB* (11,000 elements x 8 bytes), which does not fit in DL1, but it does in L2. Thus, the number of DL1 accesses expected is 11,000 approximately. Each load is expected to miss in DL1 when 64B boundaries (DL1 cache line size) are crossed, and should hit in DL1 otherwise.

Overall, for a 1-element stride we expect 1,375 (11,000/8) L2 accesses per data vector traversal. Then, since ≈88*KB* fit in L2, we expect roughly 0 memory accesses (13.75 in practice on average). When doubling the stride (so with a data vector of 176*KB*), we expect L2 accesses to double until reaching value 11,000 (at stride 8), and then flatten. Memory accesses should remain roughly 0 until L2 cache capacity (512KB) is exceeded, at stride 8 (≈ 704*KB*), when all accesses become also L2 misses so we have 11,000 DL1, L2 and memory accesses.

Figure 2 shows how DL1 accesses effectively match expectations while L2 accesses (L2D in the plot) show much higher values. Interestingly, L2 refills (L2 REFIL), i.e. lines brought explicitly on a DL1 miss, match our expectation for L2 accesses. This reveals that, apart from

the DL1 misses, we have another source of L2 accesses, which seems to be the prefetcher. When looking at the number of memory accesses (MEM), we observe that it matches quite accurately L2 accesses plus L2 refills, thus reflecting a number of accesses largely above expectations. This reveals interferences from the prefetcher since, even when data should fit in L2 (up to stride 8) and so memory accesses should be negligible, we have plenty of them. Overall, this experiment reveals that the prefetcher is active and produces severe interferences that defeat any intent to control contention in shared resources in the A53 cluster.

V. Summary of Lessons Learned

In this paper we analysed the difficulties that entails using a popular microprocessor in consumer electronics, the SnapDragon 810, in the context of critical real-time applications. This microprocessor provides the level of performance needed by many critical real-time applications, but at the same time poses a number of challenges in its utilisation, which we summarise next.

Uncontrolled resource sharing. The use of a fully-shared L2 cache across several cores poses some difficulties to control or tightly upper-bound inter-core interferences. In particular, one task running in one core is allowed to evict any line in the L2 cache, thus affecting the performance of other cores in non-obvious ways. This issue may be exacerbated by the fact that the L2 cache of this processor is inclusive with DL1 caches. Thus, a task may also get its data evicted from DL1 due to the inclusion property with L2.

The most promising approach to overcome this challenge builds upon cache partitioning. For instance, the Freescale P4080 processor, also representative of a high-performance processor of interest for real-time applications, allows configuring its shared L3 cache so that private regions are allocated to specific cores [3]. However, space partitioning may not be enough if buffers and queues are shared, which may still allow high contention across cores, thus leading to low performance guarantees [13]. However, as shown in this paper, some popular processors do not provide such support yet.

Need for documentation. For enabling MBTA based on PMCs, at least some documentation about components interfaces is mandatory. The information on hardware-to-hardware interfaces includes the way in which requests are managed (e.g. whether shared queues are used, what policies are used to serve requests). This allows reasoning about the theoretical worst-case scenarios so that microbenchmarks can be developed to stress them and obtain timing information via measurements.

Regarding software-to-hardware interfaces, which include precise information on how to enable/disable some features (e.g. prefetchers) or how to monitor hardware events through PMCs available, information released is often limited. Again, this prevents appropriate configuration and monitoring of the processor, thus defeating the intent of obtaining tight WCET estimates on top of the SnapDragon 810. The unavailability of this information often relates to IP protection and competition.

Both issues are exacerbated by the fact that many microprocessors incorporate IP from different suppliers, as in the case of the SnapDragon 810 processor, which includes at least IP from ARM and Qualcomm. In our view, detailed information will be made progressively available as market pressure increases and releasing details becomes the only way to make sales grow. Still, this shift towards openness will occur slowly.

Beyond SnapDragon 810. Although the experience with this implementation of the ARM big.LITTLE architecture has been unsuccessful and some design choices are not friendly with critical real-time applications (e.g. shared and inclusive L2 cache), our experiences with development boards implementing this architecture allow us being optimistic on the potential of ARM big.LITTLE architectures for critical applications if used appropriately. Such analysis on development boards is part of our ongoing work.

References

[1] ARM. ARM Cortex-A53 MPCore Processor. Revision: r0p4. Technical Ref. Manual, 2013.

[2] J. Bin et al. Studying co-running avionic real-time applications on multicore COTS architectures. In *ERTS²*, 2014.

[3] E. Bost. Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives), White Paper , 2013.

[4] D. Dasari et al. Identifying the sources of unpredictability in COTSbased multicore systems. In *SIES*, 2013.

[5] G. Fernandez et al. Resource usage templates and signatures for COTS multicore processors. In *DAC*, 2015.

[6] G. Fernandez et al. Computing safe contention bounds for multicore resources with round-robin and FIFO arbitration. *IEEE Transactions on Computers*, 66(4):586–600, 2017.

[7] S. Girbal et al. On the convergence of mainstream and mission-critical markets. In *DAC*, 2013.

[8] High-Performance Embedded Architecture and Compilation. HiPEAC vision, 2011, 2013, 2015 and 2017.

[9] S. Law and I. Bate. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *ECRTS*, 2016.

[10] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*, 2012.

[11] Pine64. Pine64 website, 2016.

[12] Renesas. R-Car H3, 2017. https://www.renesas.com/enus/solutions/automotive/products/rcar-h3.html.

[13] P.K. Valsan et al. Taming non-blocking caches to improve isolation in multicore real-time systems. In *RTAS*, 2016.