High-performance real-time systems design from cloud to embedded edge

Matteo Maria Andreozzi
Arm
Cambridge, United Kingdom
matteo.andreozzi@arm.com

Girish Shirasat
Arm
Cambridge, United Kingdom
girish.shirasat@arm.com

***Abstract*—Real-time computer systems are rapidly evolving into high-performance heterogeneous systems where co-location of multiple workloads can improve utilisation and re-use of system resources. This, however, comes at the cost of performance degradation due to interference on shared resources, and increased uncertainty. Resource sharing critically increases the need for predictively and deterministically managing the systems' resources. This will become a crucial property of future computing systems following the cloud-native design paradigm, in order to predict worst-case execution times (WCET) for their dynamic real-time workloads before their deployment to the embedded edge. In this work, we'll cover the impact of interference on shared resources in heterogeneous compute platforms, and we'll define the Arm terminology and the principles for high performance real-time. We will also cover system software architectures that are being envisioned in initiatives such as SOAFEE (Scalable Open Architecture for Embedded Edge [2]) to address the need to enable mixed critical workloads and the orchestration of it from cloud to embedded edge.**

***Index Terms*—real-time, Arm, high-performance, mixed-criticality, QoS, SOAFEE**

I. Introduction

In computer-science, a radical shift towards heterogeneous compute platforms is happening now, accelerated by the rise of Machine Learning and thus dedicated accelerators, and the plateauing of the Moore's law applied to CPU compute power. In the real-time computing landscape, this shift has given rise to high-performance real-time applications. On high-performing, heterogeneous systems co-location of multiple mixed criticality workloads on the same SoC can dramatically improve the utilisation of system resources, enabling resource sharing (e.g., IO devices, hardware accelerators, etc.) and improving the efficiency of data sharing across workloads.

However, co-location also comes at the cost of potential performance degradation, both average and worst-case, due to interference on shared resources, and increased uncertainty in terms of workload execution time. Both the academia and industry have been investigating the impact of shared resource contention on real-time and mixed critical software, on hardware requestors (e.g., CPU, GPU, other hardware accelerators) and on memory bandwidth availability, resources access latency, and jitter [12], [13], [14]. The advent of larger integrated platforms which will run real-time workloads alongside GPOS workloads now calls for those systems to being able to provision their resources in a quantifiable and predictable way. This becomes crucial to determine acceptable worst-case execution times (WCET) for real-time workloads and to ensure smooth and responsive operation of the GPOS workloads running alongside them.

To aid compartmentalise traffic streams on shared resources, silicon hardware designers and manufacturers have introduced, primarily in the infrastructure market, technology that allows memory transactions to be labelled and then subsequently confined to partitions of shared resources: Arm, MPAM [3], and Intel, CAT [7] . In this paper, we introduce our key design principles, methodology and metrics for designing high-performance real-time systems. We will also look at the role software plays in achieving such systems.

II. ARM HIGH-PERFORMANCE REAL-TIME DESIGN PRINCIPLES

This section describes the foundation concepts and theory behind designing Arm-based high-performance real-time systems.

A. *Real-Time Performance Metrics*

Power consumption, performance (typically average or peak performance), and chip area are widely utilized design metrics considered when designing a computing system. Such metrics are typically obtained through measurement under a set of conditions representative of the intended system production deployment operations (platform target workloads). When designing real-time systems, additional performance metrics should be considered, such as quantifying how much the system allows confident computation of worst-case execution times (WCET) for each of the real-time workloads it is being designed to execute [1]. Typically, the degree of uncertainty on computing the WCET that characterizes current high-performance real-time compute platforms makes classical methods of computing the WCET unfeasible (such as analytical) [6]. We therefore advocate the adoption of the following empirical performance metrics: i) Worst-case measured performance and ii) Time-predictability, defined as the quotient between the best-case measured performance and the worst-case measured performance.

B. *Sources of uncertainty*

The reason for high uncertainty in determining the WCET is typically down to specific sources of uncertainty. The sources of uncertainty we consider in the following affect the ability to predict or even precisely measure the timing characteristics of real-time systems:

- Workload input data or events: they cause uncertainty when influencing the software control flow or the amount of computation performed by it. In this case we say that the workload is data-variant. For example, conditional branches based on values provided by or calculated from input data can lead to different paths of execution. Also the depth of loops or recursions may depend on the content or size of the input data.
- Hardware state: state of the hardware resources at beginning of execution. Examples are initial cache contents or memory controller row buffer content.
- Interference: deviation in performance caused by workloads that contend for the same shared resources, alter the initial hardware state for other workloads or both.

C. Shared resources and interference channels

As interference arises from contention between workloads, on accessing or using shared resources, co-location of workloads on high-performance system is prone to be affected by such contention, which calls for its accurate quantification. Each hardware shared resource can exhibit one or more interference channel, each one corresponding to a place in the resource where a specific type of contention can happen. The following are examples of potential resource interference channels:

- Internal hardware buffers between pipeline stages: a congested buffer may result in a general resource stall, delaying the service provided by the resource.
- Arbitration policies: they govern which workload has access to the resource at any given time. Biased policies (e.g., strict priority ones) or generally non-work-conserving ones can cause starvation of workload request flows

III. QUALITY OF SERVICE (QOS)

Solutions that address the need for predictively and deterministically managing shared resources are collectively named Quality of Service (QoS). We define here the **QoS principles** for architecting and designing a QoS-enabled computing system capable of delivering differential performance treatment to its users (workloads).

Principle 0 - There is no controllability of a system without observability.

A system where QoS controls are successfully deployed should provide a consistent monitoring infrastructure which can sample the system and provide feedback on the functioning of such controls. It also allows and enable software to discover which shared resources are utilised by a workload.

Principle 1 – QoS is a system-level feature:

A QoS-enabled system should orchestrate its resources in a consistent way, so that the system's users (workloads) are provisioned with certain Levels of Service (memory access bandwidth and latency, compute time, peripheral access, etc.) consistent throughout.

Principle 2 - QoS is quantifiable and predictable:

The set of guarantees a QoS-enabled system can provide should be clear and the level of service that the system will guarantee to its user should be predictable based on system configuration and other conditions.

These principles should be considered both for individual hardware components design and when approaching the whole system design and integration, including software stack and functions. [4]

IV. APPLICATION MODEL

In high-performance real-time systems, hardware should be configurable and configured to provide service guarantees to a certain mix of software, and software should be enabled to manage and monitor the resources of known hardware. We here adopt a workload analysis process, the data-flow model [8], aimed at identifying the QoS requirements of the applications to be deployed on a target system. The data-flow model allows to identify the resources (processing nodes and data paths) which are involved in the execution of such workloads. Those resources will be the ones requiring their service to be characterised, and resource management when contended upon.

We start with capturing use cases requirements, as this is fundamental to enable correct resource provisioning. Workloads should have specific goals. A well-characterised workload is one for which we can specify its QoS requirements and identify a range of QoS values over which it can operate and meet said goals. A generic QoS Framework can manage different types of guarantees, all contributing to various workloads' goals, for instance performance, power, or precision. In the following, we look at leveraging QoS controls to enforce real-time guarantees, i.e., those referring to the timing properties of a real-time workload.

Realtime workloads are typically composed of a collection of entities (program code, devices, data streams) that co-operate in a non-trivial way. Workloads might have elements and functions which are dependent on external events or input, computations which might be triggered dynamically, and which might be unpredictable both in terms of activation time and duration. [5] For such systems, it is generically unfeasible to statically compute any concept such as ”anticipated peak or average load” by means of classic real-time compute approaches such as static computation graph analysis. Therefore, to define key timing parameters and constraints, we break workloads down by means of the Data-Flow model.

The Data-Flow model we adopt consists of ***Processing Nodes***, compute elements which react to events, that can produce and/or receive data, and can be either software (e.g., threads) or hardware entities (peripherals), and ***Data Paths***, which can either be physical links such as network interconnections or software communication structures, and enable connecting Processing Nodes to provide their service to the workload, but are not directly providing it to the workload itself.

According to Bikash S., et. Al [8] we can define QoS as resource management of the end-to-end allocation and scheduling of resources to workloads, based on their QoS requirements, such as:

- Each processing node **N** satisfies its local constraints
- Each data path **P** satisfies the timing constraints of all nodes N it connects, when activated

The mapping of use-cases onto the system's shared resources **N – processing nodes** and **P – data paths**, leads to the identification of interference channels where monitoring and control is needed to preserve data-flow isolation for such use-cases. This is easily identified as the set of N, P which appear in more than one dataflow.

A system capable of delivering high-performance real-time, in which QoS-based resource management is implemented to mitigate interference on its shared resources, should define its following characteristics: Granularity, Resource Monitoring and Resource Control.

A. Granularity

A system providing real-time can do so at different levels of coarseness with respect to how it identifies the users of its resources, i.e., the system **granularity**. A system's **granularity** is defined as the finest resolution at which the system – as whole – can identify users of its shared resources (the set of N and P), both for monitoring and control purposes.

B. Resource Monitoring

Monitoring of data paths and processing nodes provides insight into which shared resources are utilised by workloads, by what extent, and what causes interference on them. It also supports the implementation of control loops in the system by providing feedback to software to adapt its schedule or operate resource controls. A monitoring infrastructure should enable observation of all data paths and processing nodes involved in the computation of the system workloads. A real-time system can provide, for each identified path P, monitoring capabilities to observe performance characteristics that are specific to each shared resource, for instance:

- **Path traversal latency**: the end-to-end latency of the path from input to output. Could be punctual, or aggregated over a time window, with standard deviation and average jitter over the same time window.
- **Path utilization**: the amount of path capacity in use at any given time.
- **Path bandwidth**: i.e., amount of information transferred over the path in a time window.

For each identified processing node N, the monitoring infrastructure can also provide:

- **Node utilization**: information processed/time unit as a proportion of the node maximum capacity
- **Node access latency**: the wait time a user experiences before obtaining compute service from the node, either punctual or aggregated as average over a time window, standard deviation, and average jitter over the same time window

C. Monitor Characterization

Monitors can be characterized in terms of invasiveness, precision, frequency, measurement lag:

- **Invasiveness**: expressed as absolute delta or tolerance variance on the observed values due to perturbation caused by the monitor on the observed metric
- **Precision (resolution)**: expressed as minimum detectable unit of measurement
- **Frequency**: maximum measurement collection frequency (max between sampling frequency and interrogation frequency) at which fresh measurement (not replicas of past values) can be obtained from the monitor
- **Measurement Lag**: maximum measurement collection delay (max between sampling collection and availability to interrogation)

D. Resource Control

A Resource Control infrastructure should enable consistent regulation of all data paths P and processing nodes N identified in the data-flow analysis of the system workloads. Examples of resource controls are:

- **Path traversal latency control**: maximum input (access) to output (exit) latency for the path
- **Path bandwidth control**: minimum amount of data guaranteed transferrable by a resource user over a defined time window
- **Path utilization control**: minimum amount of resource guaranteed to a resource user, expressed as a proportion of the total available resource.

E. Resource Control Characterisation

Similarly, controls can also be characterised in terms of transitory, precision and frequency:

- **Transitory**: maximum time over which the control converges to a new steady state (regulates as intended) after receiving input. Measured as time difference between the time a new input is submitted to the control to the time the control produces a stable and updated output
- **Precision (resolution)**: minimum configurable unit of control
- **Frequency**: maximum number of (re)configurations per unit of time

F. Shared Resources Characterisation

Real-time computing systems are designed to execute workloads where data processed by computing nodes (N) and data flowing between them through paths (P) requires service guarantees. Some form of arbitration - implicit or explicit – will regulate how users are granted access to those shared resources. The execution time of a workload depends upon the forward progress of the shared resourced used by its data flow. The forward progress of a shared resource depends on the arbitration points they contain. Arbitration points can be managed by Resource Controls. Characterisation of the shared resources consists in specifying what level of service those resources provide to their users based on their arbitration policies and configuration.

For example, the above could be about

- Resource-specific attributes such as local scheduling policies or resource access rules, including configuration options and effects on the policies.
- Resource concurrent access and interference properties

Resource type and performance characteristics, scheduling policies, cost/performance functions will all contribute to resource-specific characterisation. A resource can be characterised when its service curve is known, e.g., when, given a user resource access pattern, its level of service can be predictably determined [1]. This means given a specific input into the resource, and a set configuration for its operable controls, it is possible to know a-priori what will be the measurable outcome of key resource metrics.

V. Standard Hardware Resource Monitors and Controls

The Armv8.4-A Memory System Resource Partitioning and Monitoring (MPAM) extension of the Arm Architecture define mechanisms to provide traffic flows identification throughout the system, and monitoring and control interfaces for MPAM-enabled system resources. MPAM enhances the system memory request and responses with identifiers (PARTID and PMG)

- Partition Identifiers (PARTID) that identify the flow that generated a particular request for the purpose of monitoring and control
- Performance Monitoring Group (PMG) property of PARTIDs, which can be used for the purpose of finer grain monitoring

MPAM enables operating systems or other software entities to assign a PARTID to parts of a workload, and to monitor and control their usage of the MPAM-enabled system shared resources.

A. *Monitoring Interfaces*

MPAM provides two standard monitoring interfaces, both of which are optional:

- Cache-storage usage monitors that report the cache utilisation for a given PARTID and PMG
- Memory-bandwidth usage monitors that report the number of bytes transferred for a given PARTID and PMG

Monitors can be configured to filter requests by type, for example read or write, and by a choice of PARTID and PMG or PARTID only.

B. *Control Interfaces*

MPAM provides 6 types of standard control interfaces, all of which are optional:

- Cache-portion partitioning
- Cache maximum-capacity partitioning
- Memory-bandwidth portion partitioning
- Memory-bandwidth minimum and maximum partitioning
- Memory-bandwidth proportional-stride partitioning
- Priority partitioning

Cache-portion partitioning subdivides a cache resource into several portions of equal and fixed size. Cache maximum-capacity partitioning limits the ability of a flow to occupy more than a configurable fraction of the cache capacity. Cache maximum-capacity partitioning can be combined with cache-portion partitioning, for example to restrict the ability of a single flow to occupy all the capacity of cache portions that have been made available to multiple flows.

Memory-bandwidth portion partitioning subdivides memory bandwidth into several portions (quanta). Memory-bandwidth minimum and maximum partitioning allow setting of a minimum guaranteed and maximum permitted memory bandwidth that is applied to a flow in the presence of contention. Memory-bandwidth proportional-stride partitioning is based on a configurable stride for each flow, permitting a flow to consume bandwidth in proportion to its own stride relative to the strides of other flows that are competing for bandwidth.

Priority partitioning provides a way for resources to expose partition-based configuration of internal arbitration policies. These can be used by system software for fine-grained control over scheduling and arbitration policies in the memory system.

VI. CLOUD NATIVE SOFTWARE ARCHITECTURES FOR MIXED CRITICAL WORKLOADS - SOAFEE

In the software defined embedded systems of the future, cloud-native design patterns are being considered to enable an agile DevOps environment for accelerated software feature deployments and increased developer efficiency [11]. As complex system designs get enabled with the resource controls to provide a defined QoS required by an application workload, the cloud native system software architectures and corresponding infrastructure needs to provide the ability to express the workload's spatial and temporal requirements , configure the system to achieve these requirements which includes configuring the the resource controls described in this paper and orchestrate them across the distributed embedded-compute system best suited to achieve those requirements. This is a complex problem to solve because the existing cloud-native infrastructure does not cater to mixed critical workload development and there are no standards-based configuration model or solution for mixed critical workload orchestration.

To enable complex features like mixed critical workload orchestration, the QoS features described in this paper need to be advertised in a standardized way from hardware to firmware through constructs like ACPI/DT and then feed into the operating system before exposing the platform capabilities to an orchestrator like Kubernetes. Once the scheduler in the orchestrator is aware of the system capabilities that includes the general compute attributes like CPU core count, frequency, amount of RAM, etc., along with real-time hardware capabilities like MPAM, it should then configure the system attributes using standardized interfaces and data models to satisfy the application required service level agreements before deploying the workload into the most appropriate compute element in the system.

There are several technology issues that need to be solved to address the above orchestrator usage scenario. We are listing a few below, including:

- Standardized firmware interfaces for real-time system features from hardware to firmware to operating system.
- Standardized software interfaces from OS to automotive middleware and orchestrators
- Virtual development environment in cloud to enable mixed critical workload development and deployment in production system [10]
- Rich ecosystem of commercially supported and where appropriate functionally safe / certified software components.

The SOAFEE SIG was launched to bring together major ecosystem players, including OEMs, Tier 1s, CSPs, OSVs and ISVs, SIPs and other technology providers to address some of these complex infrastructure issues. SOAFEE SIG will deliver a cloud native architecture that is enhanced for mixed-critical automotive applications and an open-source reference implementation that enables commercial and non-commercial offerings. [2].

VII. CONCLUSIONS

The paper presented an overview of our approach to designing Arm-based systems for high-performance real-time, with attention to workload decomposition and system-level requirements. We have also briefly summarized the Arm architectural support and the complementary software initiative SOAFEE. In follow-up contributions we will expand on our real-time verification methodology and on details pertaining hardware and software support for mixed criticality real-time workloads.

VIII. ACKNOWLEDGMENTS

We would like to acknowledge here the contributions of: Frances Conboy, Sam Danyo, Adrian Herrera, Jan-Peter Larsson, and Andriani Mappoura. We also thank the University of Pisa research group led by Prof. Giovanni Stea for their continued and valued research collaboration relationship with our team.

REFERENCES

[1] Alan Burns and Robert I. Davis. 2017. A Survey of Research into Mixed Criticality Systems. ACM Comput. Surv. 50, 6, Article 82 (November 2018), 37 pages. DOI:https://doi.org/10.1145/3131347

[2] SOAFEE Initiative, http://soafee.io

[3] Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A, available online at https://developer.arm.com/docs/ddi0598/latest

[4] F. Rehm et al., ”The Road towards Predictable Automotive High - Performance Platforms,” 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1915-1924, doi: 10.23919/DATE51398.2021.9473996.

[5] M. Andreozzi et al., ”Heterogeneous Systems Modelling with Adaptive Traffic Profiles and Its Application to Worst-Case Analysis of a DRAM Controller,” 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), 2020, pp. 79-86, doi: 10.1109/COMPSAC48688.2020.00020.

[6] Marco Paolieri et al., 2009. Hardware support for WCET analysis of hard real-time multicore systems. In Proceedings of the 36th annual international symposium on Computer architecture (ISCA ’09). Association for Computing Machinery, New York, NY, USA, 57–68. DOI:https://doi.org/10.1145/1555754.1555764

[7] Intel Cache Allocation Technology, https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[8] B. Sabata et al, ”Taxomomy of QoS Specifications,” in Object-Oriented Real-Time Dependable Systems, IEEE International Workshop on, Newport Beach, CA, 1997 pp. 100. doi: 10.1109/WORDS.1997.609931 url: https://doi.ieeecomputersociety.org/10.1109/WORDS.1997.609931

[9] Matteo Andreozzi et al., A MILP approach to DRAM access worst-case analysis, Computers & Operations Research, Volume 143, 2022, 105774, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2022.105774.

[10] Girish Shirasat et al., “Accelerating Software-Defined Vehicles through Cloud-To-Vehicle Edge Environmental Parity” , https://soafee.io/blog/2022/sdv_with_cloud/

[11] Girish Shirasat “Cloud Native Approach to the Software Defined Car”, https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/cloud-native-approach-to-the-software-defined-car

[12] Mohamed Hassan and Rodolfo Pellizzoni. ”Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11):2323–2336, 2018.

[13] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo. ”Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs.” 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[14] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. ”A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling”. In Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020), 2020.