Resource Isolation in Multicore Safety-Critical Systems

Multicore processors have been the subject of much interest in safety-critical domains. The increased performance and parallel execution brought by the multiple cores open new horizons in terms of size, weight and power efficiency. However, this parallel execution results in contention on shared SoC (System on Chip) resources. This phenomenon, called interference, prevents the precise execution time analysis of applications. In this paper, we present solutions to control, reduce and bound interferences using M-RTOS, an ARINC-653 multicore-capable RTOS.

1 Introduction

The transition from single core to multicore SoC presents multiple advantages in the embedded world. Their increased power efficiency and the size and performance ratio make them perfect candidates for more computationally intensive applications. Such processors embed multiple cores capable of executing different threads at the same time. Resources present on the SoC such as the caches, memory, bus or interconnect are shared between the cores.

Figure 1 presents a generic multicore architecture. Each core has its own private cache and shares a global shared cache. The cores access the same memory bus and their requests are scheduled by a bus arbiter. Finally, the memory and peripherals can be accessed by all cores.

All shared components manage concurrent accesses made by cores and ensure the transactions integrity. This allows easy communications between the different applications running in the system.

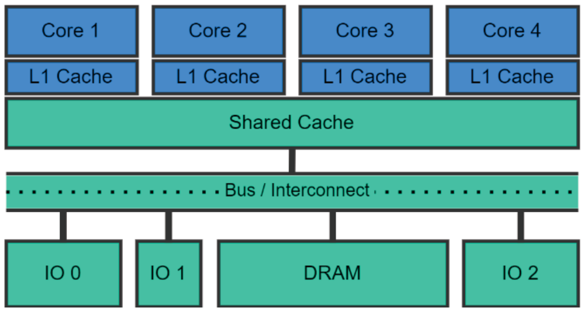

Figure 1: Multicore architecture with shared caches.

2 Resource Contention and Interferences

Shared caches contain recently accessed data (and instructions) to provide faster servicing times. However, due to their limited capacity, loading data in caches may entail the eviction of previously stored data. This means that a core, by loading new data in the cache, can evict data previously loaded by other cores. Later during the execution, if the evicted data are requested by a core, they need to be reloaded, which introduce execution time delays, referenced as interference.

The DRAM (Dynamic Random Access Memory) exhibits a similar mechanism. To increase performance, it is divided into banks. Each bank, composed of rows, has a private cache called *row buffer*. When a core accesses a row, it is loaded in the row buffer, which allows a core accessing contiguous data to be serviced faster from the row buffer. But if another core accesses a different row in the same bank, the row buffer is populated with the newly accessed

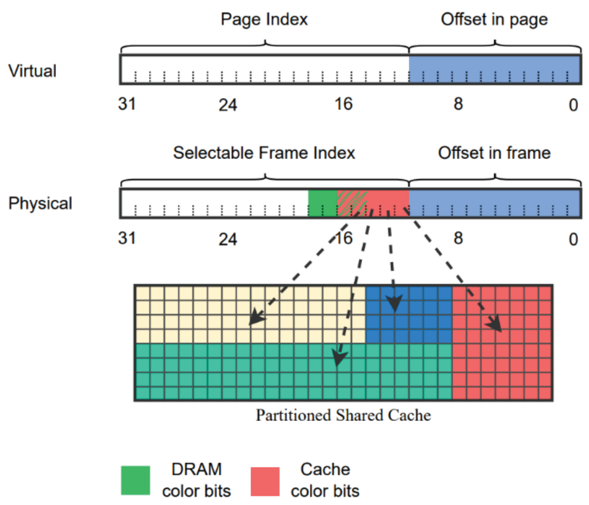

Figure 2: Memory coloring scheme with 32 cache colors (5 bits) and 15 banks (4 bits) overlapping.

row. When cores access the same bank but different rows, the buffer is swapped multiple times. This phenomenon increases the execution time compared to a case where only one core accesses the same row.

Most buses and interconnects can only service one transaction at a time. When multiple transactions are initiated from different cores, they are serialized to ensure the exclusive use of the bus. This serialization can be managed by multiple policies (priority based, First-In First-Out, etc.). Each core freezes until its transaction is serviced. The time during which a core is frozen depends on the contention on the bus and introduces interference.

Interrupts are used to react to system events such as timers, peripherals update, etc. When triggered, an interrupt is handled by the RTOS (Real-Time Operating System) code through an interrupt service routine. This execution path is independent of the applications' flow and can generate interferences. An excessive or unbounded number of interrupts impacts the execution time and predictability of applications. Interrupt interferences can be found in single-core systems but are increased due to the contention of multicore processors.

To acknowledge interferences and their impact in safety-critical systems, the CAST-32A (Certification Authorities Software Team, 2016) positional paper from the Certification Authorities Software Team and the more recent AMC-20-193 (European Aviation Safety Agency, 2022) document from the European Union Aviation Safety Agency were published. They provide guidelines and reminders on the management of interference. Although no mitigation means or implementation detail is provided, these documents pave the way for the transition to multicore architectures.

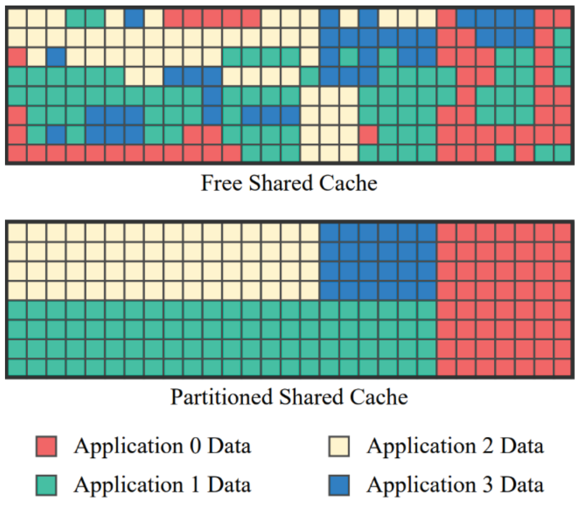

Figure 3: Effect of cache partitioning on a four-core processor.

The technical solutions presented in this paper are part of a global solution to help the user meet the CAST-32A / AMC-20-193 objectives once the final product is integrated. The approaches proposed in this paper guide the user towards meeting such objectives.

3 Memory Isolation

When isolating the memory components (shared caches and DRAM), memory coloring is a method of choice. This approach allows allocating specific regions of the cache and designated DRAM banks to each core. The CAST-20 (Certification Authorities Software Team, 2003) document preconizes to flush and invalidate the cache during each partition switch, meaning that all applications mapped to the same core can use the same shared cache regions without interfering between each other. Hence, cache partitions and DRAM banks are allocated on a per-core basis.

Memory coloring is achieved by carefully selecting addresses during the virtual-to-physical translation. Specific bits in the physical address designate in which region of the cache and which DRAM bank the data or

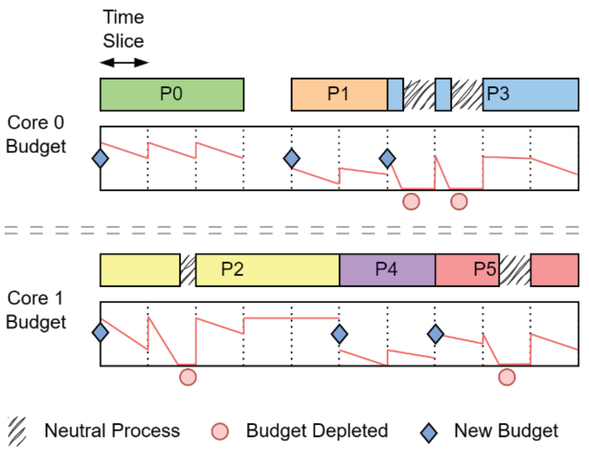

Figure 4: Bus budgeting mechanism throttling the CPU usage of partitions 2, 3 and 5.

instruction is located. Using this concept, caches can be divided in different regions, called colors. Figure 2 gives an example of memory coloring. The page index allocation provides a contiguous virtual memory space to applications. However, the frame indices associated to the page indices during translation can be scattered through the memory. This allows selecting which index the memory manager assigns to the physical addresses allocated to a given core. By carefully choosing bits 12 to 16, the manager chooses which cache regions will be accessed. The same applies when choosing bits 15 to 18 for the DRAM banks.

Each color is allocated to a core, which means that applications executing on a given core can only be provided physical addresses located in specific cache colors and DRAM banks. This ensures that applications on a specific core cannot evict cache data loaded by applications on a different core. In the same manner, DRAM bank row buffers can only be updated by applications on the same core. Figure 3 depicts the effect of cache partitioning.

Data shared between applications on different cores can be assigned to any color shared among the cores. A particular attention shall be given to balancing the shared data colors to avoid color starvation (overusing one color for shared data while other colors are only used by private data).

The advantage memory coloring has compared to hardware cache partitioning is that it has a higher portability (it only requires the architecture to have

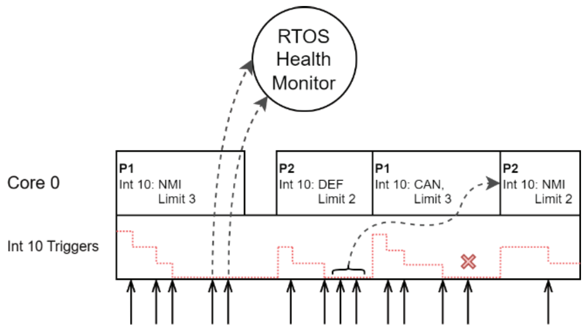

Figure 5: Interrupt management system. Interrupt 10 is deferred twice during P2 window (P2 will be notified the next time it is scheduled) and canceled once during the second P1 window. The Health Monitor is notified that Interrupt 10 sets as NMI exceeded its limit during the first time window.

memory translation mechanisms) and it allows combined DRAM bank coloring.

4 Bus Bandwidth Control

To ensure bus contention is bounded, it is important to limit the access rate made by cores to it. The bus has a maximal number of transactions for a given time slice. This number of accesses can be divided between the different cores to give an upper limit to the contention occurring during this time slice and ensure QoS (Quality of Service) among the cores. This number of accesses per time slice is called access budget and is replenished at the beginning of each time slice.

In a recent research article (Awan, Souto, Bletsas, Akesson, & Tovar, 2019), the authors show that allocating bus budget on a per-time-window basis instead of per-core basis provides better performance and allow integrating more partitions is the system. Using the memory manager proposed in (Torres Aurora Dugo, Lefoul, Harnois, Gohring de Magalhaes, & Nicolescu, 2022), M-RTOS is capable of controlling the bus access and throttle the transactions made by a partition when its budget is exceeded. Figure 4 shows a simple system with six partitions. When partition 2 exceeds its budget, it is unscheduled until its budget is replenished. The same happens for partitions 3 and 5.

Upon budget depletion, the partition is unscheduled and a neutral process is scheduled. The neutral process does not generate any access to the bus. Using different memory configuration methods (e.g., cache partitioning, scratchpads, etc.) the neutral process can

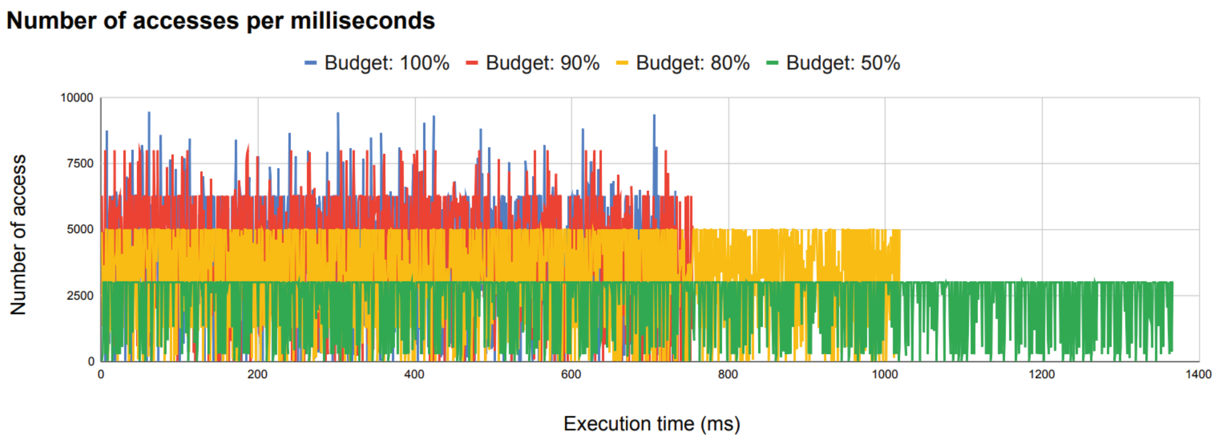

Figure 6: Effect of bus throttling on an application access pattern.

perform control tasks without accessing the bus. This improves the CPU utilization by performing background work without having to rely on an idle process that would simply halt the CPU.

Unscheduling the partitions ensures that other applications executing on different cores will be able to access the bus given their allocated budget. Figure 6 shows the effect of bus budgeting on the application execution time an access pattern.

5 Managing Multicore Interrupts

Interrupt-generated interference analysis can become an impossible task when the number of interrupts in the system is not bounded. Not only the fact that the regular interrupt

management execution path will introduce contention on the shared components, but also interrupt storms that can make the system completely unusable. Interrupt storms are situations where one or multiple interrupts are triggered in an uncontrolled manner.

This phenomenon can occur because of an incorrectly configured or defective component, but also a miss-behaving partition that generates system calls (usually triggered through interrupts) without limitation. By continuously servicing interrupts, not only the servicing core will not be able to execute applications, but it will also impact applications on other cores by contenting on shared resources.

In M-RTOS, two means are deployed to reduce and bound interrupt generated interferences. First, interrupts handlers' instructions and data can be placed in private caches or scratchpads to reduce the contention generated by the interrupt handling mechanism. Using private caches removes contention on the lower memory levels and the bus while using scratchpads reduces the servicing time, hence reducing the time during which contention is possible on the shared components.

Secondly, an interference-aware interrupt manager was developed to control the interrupt servicing rate and prevent interrupt storms.

This novel interrupt manager enforces a limit to the number of times a specific interrupt can be serviced. Three classes of interrupts are configurable.

- Cancelable Interrupts (CAN): When reaching their servicing limit, cancelable interrupts will be discarded before being serviced. Depending on the hardware capabilities, cancelable interrupts will be disabled until their servicing limit is replenished.
- Deferrable Interrupts (DEF): Deferrable interrupts are not serviced after their servicing limit is reached. However, they are not disabled and each time a deferrable interrupt is triggered, M-RTOS registers it. When the deferrable interrupts see their budget replenished, the executing application is notified with the number of

times the interrupt was deferred and can execute a configurable action.

- Non-Maskable Interrupts (NMI): This class of interrupts gathers critical interrupts. They can be given a servicing limit but will always be serviced. If the interrupt exceeds its servicing limit, M-RTOS will be notified based on the platform configuration.

The servicing limit is configured by the system integrator and is applied at the time-window scope. Meaning that each new time window, the interrupts servicing limits will be replenished with a new budget. Each interrupt can be configured to have a specific class and servicing limit per time window. Figure 5 illustrate the mechanism present in M-RTOS to control interrupt storms.

6 Using M-RTOS for Multicore Interference Mitigation

6.1 Safety Net

M-RTOS is a multicore-capable ARINC-653 compliant RTOS. It provides interference mitigation mechanisms to ensure isolation between the SoC components. M-RTOS also embeds *Safety Net* monitors to ensure the system's robustness and notify the user in case of unmitigated or unexpected interference. The *Safety Net* concept was introduced in the CAST-32A (Certification Authorities Software Team, 2016) document to mitigate unexpected breaches of isolation. M-RTOS integrates different monitors, first defined as formal constraints represented by mathematical equations. The constraints are implemented as monitors at key points of the RTOS such as the context switch, the page fault handler, etc. In the following equation, the number of interrupts handling for a given time window and a given interrupt index must be less than or equal to its defined budget. This constraint allows preventing interrupt storms.

$$\forall w, \forall i \in I_A(w), \sum_{t=Q_{wf}}^{Q_{wl}} I_C(w,i)$$

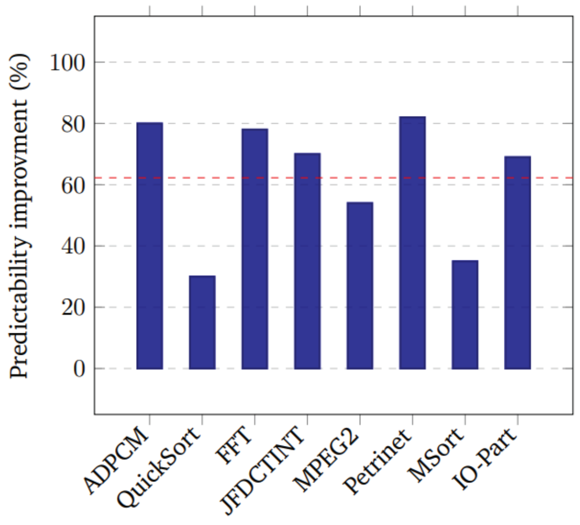

Figure 7: Predictability increase over eight applications using the M-RTOS memory partitioning feature.

6.2 Memory Partitioning

Using memory partitioning, M-RTOS is capable of bounding shared cache and DRAM interferences. By correctly isolating the shared memory components, the predictability of partitions running on four different cores is increased by 62.25% on average. Figure 7 presents the predictability increase for eight different applications running across the four cores. The predictability quantified by the standard deviation of the applications' execution time is depicted with respect to the applications' predictability when no memory isolation is used.

Applications exhibit different improvements depending on their memory and cache usage. When an application is prone to suffer from interferences, the gain obtained from partitioning the memory is greater than when an application is prone to generating interferences. Applications that profit from the cache will not see their data evicted from applications running on other cores.

Memory partitioning is completely transparent to applications in M-RTOS. The mechanism can be enabled and disabled and the allocation of the different cache and DRAM partitions to the cores is achieved in the system configuration. The number of available cache partitions and DRAM banks depends on the hardware specifications.

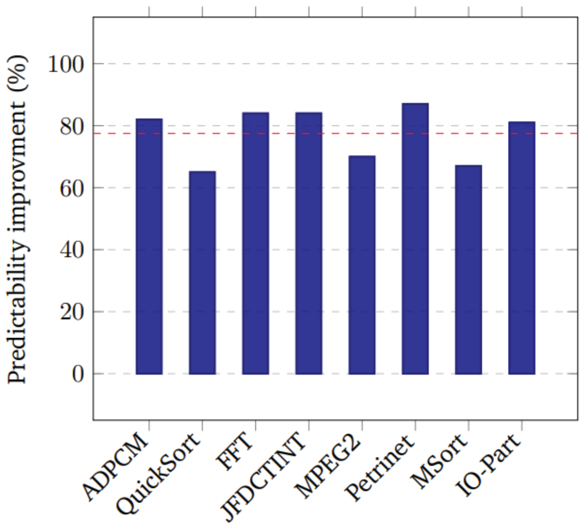

Figure 8: Predictability increase over eight applications using the M-RTOS memory partitioning and bus bandwidth limitation features.

6.3 Bus Bandwidth Limitation

Throttling memory bandwidth allows reducing and bounding contention on the bus. M-RTOS uses per-partition bus budgeting to ensure QoS for all applications. Through the configuration, the user can set the maximal number of accesses (maximal budget) that the bus can handle during a given period of time called *regulation period*. Based on this upper bound, each application can be allocated a portion of bus access budget. While validating the configuration offline ensures that the sum of the budget of the co-scheduled applications does not exceed the maximal budget, the *Safety Net* enforces that this constraint is respected at runtime.

Using bus budgeting alongside memory partitioning, the predictability of the system presented in Figure 8 was increased by 77.5% on average. To provide better best-effort performance and based on the work presented in (Yun, Yao, Pellizzoni, Caccamo, & Sha, 2016), M-RTOS proposes to add a free shared budget pool that can be used by applications to reclaim budget when they exceeded theirs. Four different execution modes allow providing more configuration flexibility. The budgets and execution modes can be set independently for each partition running on M-RTOS. Bus bandwidth limitation can be enabled or disabled through the configuration of the RTOS and the provided budget can be calculated using methods such as presented in (Torres Aurora Dugo, et al., 2022).

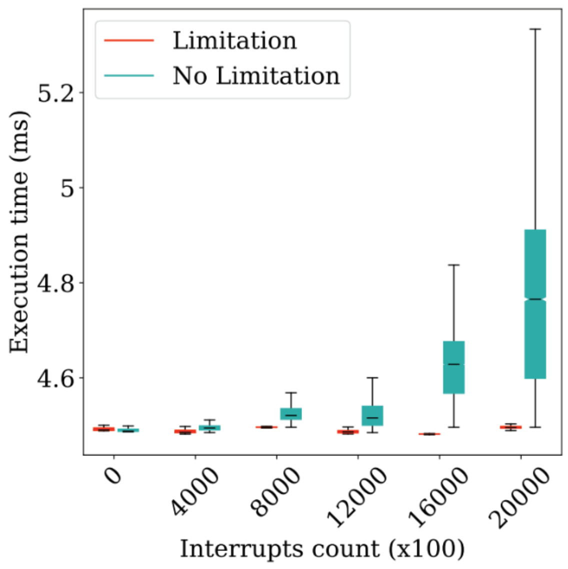

Figure 9: Execution time distribution with limited interrupts (orange) and without limitation (green).

6.4 Interrupt Management

M-RTOS provides support to bound interrupt servicing. This mechanism sets a limit to the number of times an interrupt can be serviced. M-RTOS can be configured to provide a different limit for each interrupt occurring in the system. Furthermore, it implements the three interrupt classes presented in Section 5. Selecting the interrupt classes, interrupt limitations and the activation of the mechanism is achieved through the RTOS configuration. M-RTOS uses fast interrupt filters that do not generate interference. Thus, canceled, or deferred interrupts happening on one core cannot impact the execution time of applications executing on different cores.

Figure 9 shows the distribution of the execution times of an application running on M-RTOS. The number of interrupts being triggered on other cores is depicted on the horizontal axis while the execution time of the application is shown on the vertical axis. When interrupts are all serviced, interferences generated by the other cores handling the interrupts increase the application's execution time and reduce its predictability. However, when interrupts are set as deferrable (they are registered by M-RTOS but not serviced after they depleted their servicing limit), interference are removed, and the application is not impacted anymore.

7 Conclusion

Multicore architectures are becoming more and more present in embedded systems. However, their use still presents challenges when it comes to safety-critical applications. The contention on the resources shared among the core introduces timing delays called interference. Those delays prevent the correct analysis of the applications' execution time and present a roadblock in the path to certification.

M-RTOS is an ARINC-653 compliant Real Time Operating System that proposes interference mitigation approaches to correctly isolate the shared memory hierarchy and ensure both Quality of Service and an improved timing predictability for applications. M-RTOS embeds run-time monitors that control the behavior of the complete system and prevent unexpected interference from impacting the applications' execution.

Through easy and intuitive configuration tools, M-RTOS allows the mitigation of interferences while being totally transparent to the applications. This permits the seamless integration of the system.

8 Bibliography

Awan, M. A., Souto, P. F., Bletsas, K., Akesson, B., & Tovar, E. (2019). Memory access regulation,Multiframe task model. *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2019.*

Certification Authorities Software Team. (2003). *CAST-20, Addressing Cache in Airborne Systems and Equipment.* USA.

Certification Authorities Software Team. (2016). *CAST-32A Multicore Processors.* USA.

European Aviation Safety Agency. (2022). *General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances AMC 20-193.* Europe.

Torres Aurora Dugo, A., Lefoul, J.-B., Ben-Salem, A., Harnois, S., Gohring de Magalhaes, F., & Nicolescu, G. (2022). Efficient Scheduling, Mapping and Memory Bandwidth Allocation for Safety-Critical Systems. *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS).* Quebec: IEEE.

Torres Aurora Dugo, A., Lefoul, J.-B., Harnois, S., Gohring de Magalhaes, F., & Nicolescu, G. (2022). Certifiable Memory Management System for Safety Critical Partitioned System. *ERTS 2022 - 11th European Congress on Embedded Real Time Software and Systems.* Toulouse.

Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., & Sha, L. (2016). Memory bandwidth management for efficient performance isolation in multi-core platforms. *IEEE Transactions on Computers*.

Mannarino Systems & Software Inc.
100 Boulevard Alexis-Nihon, Suite 800
St-Laurent, (Quebec) H4M 2P4, Canada
+1 (514) 381-1360
biz@mss.ca

RTOS