A Lightweight Spatio-temporally Partitioned Multicore Architecture for Concurrent Execution of Safety Critical Workloads

Qingchuan Shi†, Kartik Lakshminarasimhan†, Christopher Noll*, Eelco Scholte*, Omer Khan†
*United Technologies Aerospace Systems, USA
†University of Connecticut, USA

***Abstract*—Modern aircraft systems employ numerous processors to achieve system functionality. In particular, engine controls and power distribution subsystems rely heavily on software to provide safety-critical functionality, and are expected to move towards multicore architectures. The computing hardware-layer of avionic systems must be able to execute many concurrent workloads under tight deterministic execution guarantees to meet the safety standards. Single-chip multicores are attractive for safety-critical embedded systems due to their lightweight form factor. However, multicores aggressively share hardware resources, leading to interference that in turn creates non-deterministic execution for multiple concurrent workloads. We propose an approach to remove on-chip interference via a set of methods to spatio-temporally partition shared multicore resources. Our proposed partitioning scheme is bounded within the worst case execution, and ensures efficient performance and deterministic execution.**

I. Introduction

The distributed and embedded software in modern avionic systems is increasingly becoming a first order design constraint. For example, modern aircrafts deploy complex embedded software and hardware to manage various operational and management layers (such as engine controls and power distribution subsystems) [1]. The complexity of the avionics software is rapidly increasing because of the addition of cyber-physical dimension to its system design, making a paradigm shift from a distributed on-board system architecture to an integrated modular avionics (IMA) architecture. IMA integrates several unrelated or loosely related software applications formerly distributed across multiple avionic systems, and lessens the system weight and energy consumption due to fewer compute and wiring components [4].

At the hardware layer, multicore technology has emerged as an ubiquitous computational primitive. These multicores integrate many discrete hardware components on a single chip, such as compute processors, cache memories, interconnection networks, and memory controllers. Through integration, multicores deliver small form factor and reduce the embedded hardware's area, power and energy footprint. The key question we seek to answer in this paper is whether multicore technology can be deployed in safety-critical systems, such as the IMA architecture for avionics.

Traditionally, safety-critical applications are executed on discrete processors. In case of a multicore processor, this implies the application may not fully utilize the hardware capabilities, such as the additional processing cores for computations, or the available memory bandwidth to access data. Therefore, it is beneficial to concurrently execute multiple applications and fully utilize the available hardware capacity. The challenge is that the aggressive integration leads to space-time sharing of hardware resources. This sharing exists even though the concurrently executed applications do not explicitly communicate across their control and dataflow. Sharing causes interference among the hardware resources, such as the on-chip last level cache (LLC), memory controller, and the network-on-chip (NoC). These *interference channels* lead to non-deterministic timing and power behaviors across applications, leading to loss of performance guarantees. For example, the multicore LLC holds the working sets of concurrent applications. Improper allocation of the cache resources to any particular application can lead to unpredictable cache behavior that can be detrimental for performance. In the worst case an application may get locked out from accessing data that can lead to starvation like conditions for the embedded system. This is unacceptable for safety-critical applications, such as the ones executing under the IMA architecture for avionics. On the other hand, multicore's efficient form factor offers desirable capabilities for such embedded systems. Therefore, we propose a holistic approach to remove all on-chip interference channels via a set of lightweight methods to spatio-temporally partition shared multicore resources.

The key contributions of this paper are:

- Propose a holistic spatio-temporal partitioning scheme for the multicore *interference channels* (i.e., LLC, memory controller, and NoC).
- Improve performance of concurrently executing safety-critical applications using the proposed partitioning scheme, which tightens the bound for worst case execution time (WCET). We also show that deterministic performance under the proposed scheme is competitive with respect to a non-deterministic scheme that exploits fine-grain sharing of hardware resources.

Our approach is a step towards satisfying deterministic performance guarantees for an embedded system that deploys multi-core architectures. Relevant initiatives such as CAST-32 [5] are working towards defining common approaches to handling multi-core architectures within the aerospace domain. The proposed scheme fulfills such requirements.

II. On-chip Interference

A. Shared Resources

The baseline is a 4-core shared LLC multicore system with similar configurations as NXP QorIQ T2080 [22], which is widely used in safety critical embedded systems. As shown in

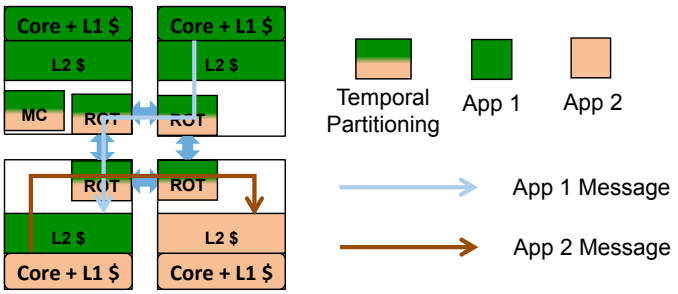

Fig. 1: Interference Channels in a Multicore

Figure 1, each core has private compute pipeline and level-1 (L1) cache. Note that most avionics systems do not use multi-threading, even though the cores have the capability. Hence, an application mapped to a core has private access to these resources. The shared resources (highlighted in Figure 1) are the Last Level Cache (L2$), Memory Controller (MC) and NoC. Shared resources are accessible by each core in the system and traditionally fine-grain hardware schemes, such as cache insertion/replacement policies, dictate how they are accessed. Although such schemes work well in practice, they are highly unpredictable and do not guarantee bounded performance for safety-critical systems.

Worst case execution time (WCET) analysis is always performed such that determinism is ensured by not violating the WCET [6].

The key idea we propose is to tighten the WCET bounds to improve the performance of co-located safety-critical applications on a multicore processor. We achieve this through deterministic spatio-temporal partitioning of the LLC, memory controller, and NoC. The following subsections explain the proposed partitioning schemes where applications are assigned fixed number of LLC cache slices, and a fixed number of memory controllers and/or memory bandwidth. However, we first solidify FAA requirements for such a scheme.

B. FAA CAST-32 Requirements and Evaluation

In this paper, the hardware level requirements related to on-chip resource sharing in FAA CAST-32 [5] are illuminated. The rationale of the concerns for multicore (MCP) are raised by the uncertainty due to on-chip resource sharing, which means same hardware component is accessed by different applications running concurrently. The related aspects of FAA CAST-32 in the scope of this paper are the on-chip interference (including shared LLC, memory controller, and NoC), and on-chip resource usage. Efforts are made to address these in the paper.

1) MCP Interference Channels: Interference arises in the components that are accessed by multiple applications. In a multicore system these revolve around data accesses. As shown in Figure 1, when a compute core needs to fetch some data, if it is not available in the core's private L1 cache, a request is sent to the LLC cache slice location for that data. For the baseline system discussed in this paper, the shared LLC cache places the data based on its address. In case the data is not located in the local LLC slice, the request is sent to the appropritae LLC slice via the NoC. If the data is not in LLC cache neither, the request is forwarded to the memory controller, through which the data is fetched from DRAM. In this process, the LLC, memory controller and NoC are accessed. When there is more than one application running, all of them need to access these components, which are identified as interference channels. The concern is that through these channels, the applications running concurrently could affect each other. This can cause non-deterministic application behaviors.

The relevant task in FAA CAST-32 is to conduct a functional interference analysis to identify all the interference channels between the concurrent applications executing on the multicore, and mitigate the interference channels. Note that there are specific tasks listed for shared caches in the requirement, especially its effects on WCET. In this paper all these three channels are analyzed with respect to WCET.

A qualitative interference analysis is performed for the proposed schemes. On-chip interference is eliminated when all the shared resources are partitioned. As discussed in Section II-A, the interference channels of the proposed system are shared LLC, memory controller, and NoC. Since the proposed scheme spatio-temporally partitions these components, each application gets dedicated resources. We discuss the proposed scheme and its evaluation in the next sections. The sharing effects of the concurrently executing applications in LLC are illustrated, as well as WCET. The proposed scheme spatially paritions the LLC, thus guaranteeing the available LLC capacity for each application.

2) On-chip Resource Usage: The concurrently executing applications change their on-chip resource demands at runtime. The accesses to shared resources can cause contention with performance overheads. Furthermore, if the overall available resources are exceeded by the total demands of the applications, they may behave in a non-deterministic manner.

The relevant task in this aspect is to describe how to allocate, manage and measure the use of resources and avoide contention. It is desirable to certify that total resource demands of all applications do not exceed the total available resources.

In this paper, the contention of different resources are analyzed. The proposed partitioning schemes avoid contention between different applications by allocating dedicated resources. Next, we describe the proposed spatio-temporal partitioning mechanisms for the baseline multicore system introduced in Figure 1.

III. Partitioning Shared Resources

A. Spatial Partitioning of Last Level Cache

It is not desirable to perform temporal partitioning of the LLC to guarantee zero interference. The reason is due to the prohibitive latency overheads in flushing the LLC data to off-chip memory and bringing it back during each partition's execution. It makes spatial partitioning the only viable way to guarantee deterministic resource sharing in LLC, which in turn potentially improves performance. Our baseline multicore has physically distributed and logically shared last level cache (LLC). Each core gets a slice of the LLC. This distributed nature of the LLC places the onus on partitioning and placement of the data in the LLC, also known as the non-uniform cache access (NUCA) [10]. We exploit this NUCA capability

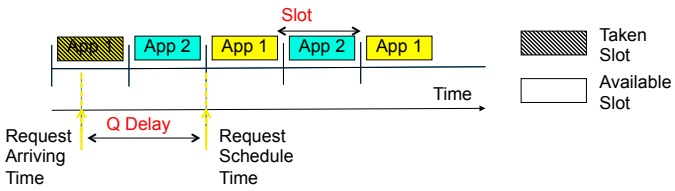

Fig. 2: Timing model of the partitioned queues.

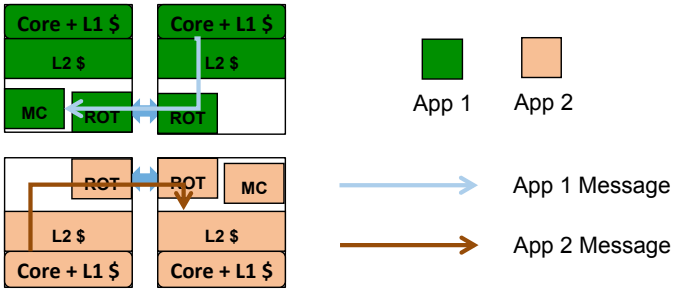

Fig. 3: Isolated applications with 2 memory controllers on-chip.

to spatially partition the LLC slices across the concurrent applications. Although further partitioning can be done at fine granularities (such as way or set partitioning in each cache slice), we leave such exploration for future work.

Each LLC slice consists of multiple physical cache banks. Before start of execution, the number of LLC slices for allocation to each application are determined based on the expected behavior of that application. For example, a memory bound application is expected to work well with smaller number of LLC slices as compared to a compute bound application. Once the number of LLC slices is determined for an application, its address space is *statically* mapped to those LLC slices (through some architecture register setting). A static address interleaving scheme at the cache line granularity is utilized to achieve optimal use of the available LLC capacity allocated for that application [2]. At runtime, when an application accesses a data word from its address space, the static mapping determines the LLC slice for allocating the corresponding cache line from off-chip memory. In this way the LLC interference channel is managed by isolating the application data to be mapped in unique LLC slices.

B. Temporal Partitioning of Memory Controller

An on-chip memory controller consists of multiple FIFO queues for read/write requests to the off-chip memory. Each memory controller queue is associated with the available memory controller bandwidth. Memory bandwidth decides the rate at which off-chip data packets are accessed, and is related to the number of pins that connect the multicore processor to the off-chip memory. In order to avoid interference between applications, memory controller bandwidth is divided among them. Each core then gets its allocated memory bandwidth, effectively temporally partitioning the shared memory controllers among the applications. In this case, each application gets its respected space for a deterministic interval of time, similar to the scheme by Wang et al. [11]. As shown in Figure 2, the queuing delay for each request is calculated as the waiting time between its arrival and schedule. The schedule time only depends on the next available ”slot” of corresponding application. This enables non-blocking access to the memory controller even though the other application might have a significant number of queued up requests.

C. Temporal Partitioning of NoC

In this paper a 2-D mesh NoC is considered, which consists of on-chip routers and links. NoC is accessed by on-chip components on each tile. The router on each tile can be accessed directly by L1/LLC cache and memory controller (if available) on the same tile, and by components from another tile through NoC. Thus NoC utilization is determined by system configurations, such as application mapping, memory controller location, and routing scheme. In the baseline system, there is one on-chip memory controller, as shown in Figure 1, which is accessed by the two concurrent applications. For the application mapping shown in the same figure, application 1 is allocated three LLC slices, whereas application 2 is allocated one LLC slice. When X-Y routing is used, routers on the left side (top and bottom) can be accessed by both applications. However, when the application mapping or routing scheme is changed the router usage could be different. Thus it is not feasible to isolate NoC traffic for each application in certain region without accessing the routers elsewhere on-chip. In order to avoid interference between applications for the system setup in this paper, they need to have dedicated NoC bandwidth. Each application has its own input/output queue to the router, so that requests from one application can not block ones from the other application. Effectively the router is temporally partitioned such that each application has its own time slot to access the router on each tile. The queue model for partitioned router is similar to the one for memory controller.

D. Isolation Scheme with Architectural Support

Due to the fact that there is only one memory controller in the baseline, it has to be temporally partitioned between applications. This also limits the feasible solutions for NoC to avoid interference. As mentioned earlier, temporal partitioning the memory controller and all NoC routers is a feasible solution for the baseline system. However, this introduces system complexity and significant performance overhead due to inefficient utilization of on-chip resources. Thus in this paper we propose a scheme to isolate applications with one additional memory controller, as shown in Figure 3. Each application has its own memory controller, therefore adding a way for the system to spatially isolate the memory traffic for two applications. Note that the off-chip memory bandwidth is distributed among the two memory controllers. Furthermore, with a spatially partitioned LLC and NoC routing scheme, the messages from each application can be restrained in certain regions on-chip, which only contains the tiles hosting that application. All on-chip data of an application are located in the L1/LLC caches of the tiles hosting it. When an application incurs L1 misses, the requests are directed to the LLC slices belonging to that application. When there are LLC misses, the requests are directed to the memory controller located in one of its tiles. This is achieved using a revised physical address

Architectural Parameter	Value
Number of Cores	4 @ 1 GHz
Compute Pipeline per Core	In-Order, Single-Issue
Physical Address Length	48 bits
Memory Subsystem	
L1-I Cache per core	32 KB, 4-way Assoc., 1 cycle
L1-D Cache per core	32 KB, 4-way Assoc., 1 cycle
LLC per core	512 KB, 8-way Assoc., 8 cycle Inclusive, NUCA
Cache Line Size	64 bytes
Num. of Memory Controllers	1 for baseline; 2 for ISOLATION
DRAM Bandwidth	10 GB,
DRAM Latency	75 ns
point-to-point Network	
Hop Latency	2 cycles (1-router, 1-link)
Flit Width	64 bits
Header (Src, Dest, Addr, MsgType)	1 flit
Cache Line Length	8 flits (512 bits)

TABLE I: Architectural parameters for evaluation.

mapping of the applications.

The NoC routing scheme is crucial to isolate the on-chip traffic. Since all resources needed by one application are located on its allocated tiles, the routing scheme needs to guarantee no packet is routed through other tiles. We utilize the inherent determinism of static X-Y routing protocol to ensure isolation of the NoC traffic. Figure 3 shows that X-Y routing can be used for this purpose given the tiles are allocated appropriately to each application.

IV. Experimental Methology

A. Performance Modeling

All experiments are performed using a modified version of the Graphite multicore simulator [3]. Graphite is modified to support multiple multi-threaded applications to execute concurrently. The simulator is configured as a four-core shared memory multicore, as shown in Table 1. The caches and memory controller are configured according to NXP QorIQ T2080 [22]. Each core consists of a compute pipeline, private L1 instruction and data caches, a physically distributed shared LLC with integrated directory, and a network router. NoC model uses a 2-D mesh with X-Y routing scheme. The network delay consists of 4 parts: router latency, link latency, serialization delay, and queuing delay. The contention delay accumulates from the router at each hop. Each router has five output queues (north, east, south, west, and self) with contention delay. It is modeled using a history-tree model for baseline, and the 'slot' model explained earlier for temporal partition. Since modern network-on-chip routers are pipelined, and 2-or even 1-cycle per hop router latency has been demonstrated, we model a 2-cycle per hop delay (1-cycle for router and link delay each), and the serialization delay is 1-cycle for each flit in the n. We assume infinite buffering for router input queue without contention delay. Similar contention delay modeling is in place for the memory controller. Cache coherence is implemented using the directory based MSI protocol.

1) ***Stand_Alone****:* The Stand_Alone executes a single two-threaded application on the 4-core system, with access allowed to all shared resources. Thus 2 compute pipelines, 2 L1 instruction and 2 L1 data caches, 4 LLC slices, and full memory bandwidth is available to the application. Each multithreaded application is run to completion, and we measure the parallel completion time, i.e., the number of cycles to execute the application.

Application	Problem Size
RADIX	1M integers, radix 1024
FFT	1M points
MM	512×512 matrix, 16×16 blocks
CHOLESKY	512×512 matrix, 16×16 blocks
PAGERANK	131072 nodes, 16 edges
SSSP	512 nodes, 16 edges
MLP	Input layer 256 neurons, Output layer 10 neurons

TABLE II: Benchmarks and their input sizes.

2) ***Un_Controlled****:* The uncontrolled sharing configuration executes two applications concurrently without any partitioning scheme. Both two-threaded applications access the 4 LLC slices and full memory bandwidth without any restrictions.

3) ***WCET****:* Spatio-temporal partitioning schemes are implemented for the LLC, memory controller and NoC interference channels. For each application combination, LLC slices are spatially partitioned in three ways ((**LLC(1,3), LLC(2,2), LLC(3,1)**), while memory controller and NoC are temporally partitioned ((**MC_TP, NoC_TP**). The notion of WCET is that in order to eliminate interference, no shared on-chip channels can be used. For the baseline system, without information about each application's behavior, the pratical way is to equally distribute the available resources. Thus WCET is configured in a way that two LLC slices are allocated for each application, while memory controller and NoC routers are temporally partitioned.

4) ***ISOLATION****:* This is the proposed resource partitioning scheme introduced in Section III-D. All interference channels are spatially partitioned including the LLC, NoC and the memory controllers.

B. Benchmarks

We use three SPLASH-2 benchmarks (FFT, RADIX, and CHOLESKY) [16], two graph analytic benchmarks from CRONO suit (PAGERANK, and SSSP) [14], one matrix multiplication (MM) benchmark, and one machine learning benchmark (MLP), to represent various applications for safety-critical systems. The benchmark input sizes used are shown in Table II. We use multi-threaded benchmarks because they are scalable and effectively utilize multicore resources at two threads.

FFT is matrix based, and an important kernel in several signal processing applications. Real-time time applications, such as audio/video/sensor processing use FFT as the basic transform for frequency domain analysis. The inner-product in all matrix computations is used widely in vector based computations, which is an important component for many safety-critical systems. The matrix multiplication (MM) is loop-tiled, and thus optimized for cache accesses. CHOLESKY is also a matrix decomposition algorithm. It is used in systems where model reduction is done during the state-space analysis.

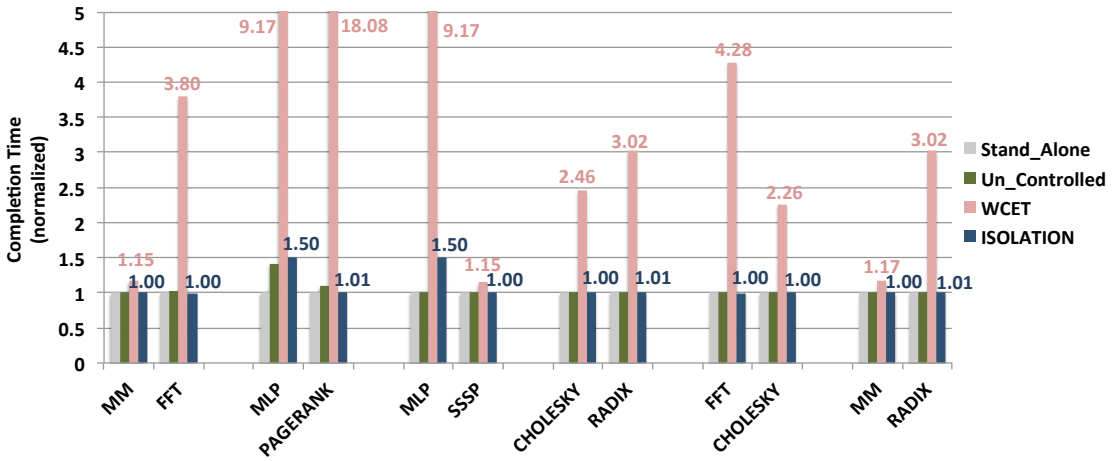

Fig. 4: Uncontroled sharing, WCET, and isolation results for different benchmark combinations. All completion times are normalized to **Stand_Alone**. WCET and ISOLATION results are labeled in the figure.

Consider a helicopter scenario where it needs to optimize its rotor speed according to the environmental conditions. The environment data needs to be reduced for comprehension. RADIX employs a divide-and-conquer methodology for sorting data according to some threshold, such as magnitude. Multi-Layer Perceptron (MLP) is a neural network Machine Learning algorithm which is used in modern day Advanced Driver Assistance System (ADAS). The MLP benchmark in this paper is implemented for predictions for handwritten digits. Single Source Shortest Path algorithm (SSSP) is an implementation of Dijkstras algorithm which is used widely for path finding purposes. PAGERANK is a Graph-based prediction algorithm used in recommendor systems in various platforms.

V. Results and Analysis

Figure 4 shows the completion time of Un_Controlled, WCET, and ISOLATION for different benchmarks combinations. The results are normalized to Stand_Alone. In all cases, the result of Un_Controlled sharing is worse than or equal to Stand_Alone. In some cases, the completion time of the two setups is similar. This is because the system has enough resources, and the benchmarks have relatively small input sizes. When the LLC capacity, NoC bandwidth and memory controller bandwidth well exceed the total demands of both benchmarks, the interference effects may not be distinctive. However this condition varies from case to case, and can not hold even for the same benchmark with different combinations. In Un_Controlled of MLP-SSSP, MLP has similar completion time compared to Stand_Alone. In contrary, for MLP-PAGERANK, MLP has evidently worse completion time in Un_Controlled due to on-chip interference. Note that the Stand_Alone completion time is unique for each benchmark. Thus it is clear that Un_Controlled cannot guarantee deterministic execution.

WCET slowdowns vary from 1.15× to 21×. The main reason is that benchmarks have diverse demands of on-chip resources, and the resources allocated to each benchmark are much limited compared to Stand_Alone. WCET spatially partitions the LLC, and the space allocated to a benchmark may not fulfill its demands. Furthermore, the temporal partitioning schemes have significant performance overheads due to the reduced NoC and memory controller bandwidth. Applications with significant on-chip traffic and off-chip accesses are affected significantly. PAGERANK performs much worse for WCET since it has large amount of cache misses, which generates requests in the NoC and results in accessing the DRAM through memory controller. It is also clear that each benchmark always maintains the same WCET when combined with different benchmarks (and hosted at the same cores). This justifies that WCET eliminates interference, however with large performance overhead. WCET of the same benchmark can change when it is hosted at different cores. This is because the memory controller in WCET is located at the last tile. Benchmarks' relative position to the memory controller affects the path to access it. As shown for FFT, when hosted at the first two cores (mapped as application 1), its WCET is larger than the case when it is hosted at the last two cores (mapped as application 2). This is due to the longer latency and additional NoC traffic when accessing memory controller.

For all combinations, the ISOLATION scheme is able to reduce the performance overhead remarkably. In most cases the completion time is close or equal to Stand_Alone. This is achieved by eliminating on-chip interference without temporally partitioning the NoC and memory controller. Overall, ISOLATION is able to provide deterministic execution at average overhead of ∼1.07X, compared to WCET at ∼5.52X.

Figure 5 shows the results for the MLP-PAGERANK and MLP-SSSP benchmark combinations. For the same MLP benchmarks' execution with different benchmarks (PAGERANK and SSSP), its completion time is inconsistent when no resource partitioning scheme is used (Un_Controlled). This justifies the nondeterminism due to interference. When the LLC is spatially partitioned, and memory controller and NoC are temporally partitioned, shown as L2_XX_MC_100_NoC_10, the performance overhead is evident due to limited memory controller and NoC bandwidth. This also makes different LLC spatial partitions distinct since contentions at memory controller make the LLC misses much more costly. When all the on-chip interference channels are partitioned, MLP completion time becomes stable, and is not affected by the other application. This is a key result in

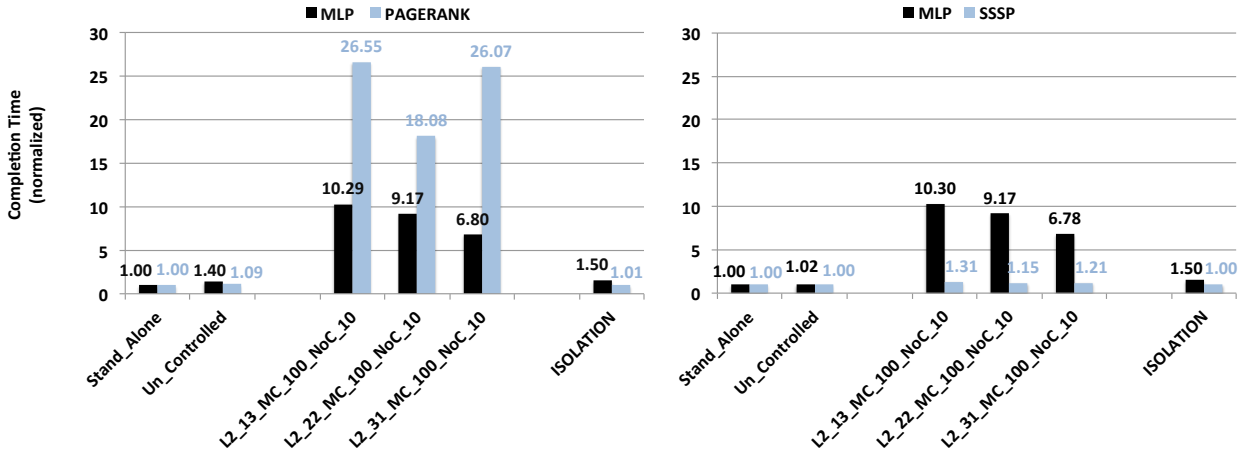

Fig. 5: Benchmarks combinations of MLP. All completion times are normalized to **Stand_Alone**.

this paper that demonstrates that applications can indeed be partitioned for deterministic behavior. **Yet, as evident for ISOLATION scheme in Figure 4, MLP achieves performance that is compared to Un_Controlled. However, the ISOLATION scheme guarantees deterministic execution of concurrent safety critical applications.**

VI. Related work

The related works cover a wide range of research done in safety-critical systems and multicore architectures. In this section, holistic multicore designs for safety critical systems are discussed first, followed by partition schemes for LLC, memory controller and NoC. The partition schemes are not necessarily designed for interference reduction, their practicality for such a purpose is discussed.

Multicore for Safety-critical Systems: Rising efforts have been made to deploy multicore in safety-critical systems. Large projects such as EMC^2 are establishing multicore in such domains [18]. Systems have been designed focusing on mixed-criticality workloads on multicore processors [17], [20]. In these systems, shared resources are assigned in favor of Hard Real-time Tasks (HRT) over Non Hard Real-time Tasks (NHRT). HRT has its won private cache, and high priority when accessing memory controller and bus. When running with NHRT in a multicore, the maximum access time of HRT to shared resources is bounded. Nowotsch et al. [7] evaluated a multicore system in an avionics platform and stressed the interference problem in shared multicore resources. It uses address mapping to spatially partition the cache and memory controller, while suggests using temporal partition for the bus. However, these schemes cannot isolate on-chip traffic and have significant performance overheads. In this work we propose a scheme that eliminates interference in all shared resources of a multicore processor by providing isolation of each application. It avoids temporal partition of the memory controller and NoC, while provides determinism for both applications running concurrently. Moreover, their WCET estimation for multicore system is based on single core analysis, while we evaluate WCET for different application combinations in multicore setup with spatial-temporal partion schemes.

LLC Partitioning: Bui et al. [8] treated cache partitioning as an optimization problem, and used genetic algorithm globally and simulated annealing locally to temporally partition the cache with a cyclic scheduler. However, it only claimed to reduce interference and improve schedulability. Slijepcevic et al. [9] proposes a Time-Randomized (TR) cache approach where a probabilistic timing analysis helped in controlling shared LLC eviction frequency. Time Randomization helps in improving performance of shared cache alone but does not help in co-ordinating the interference with other shared resources. Moinuddin et al. [12] partitioned the shared LLC among multiple applications based on each application's utility. The above scheme known as Utility based Cache Partitioning (UCP) is implemented with the help of Utility MONitor (UMON) hardware counters that sample a small number of cache lines to track their hits and misses. Ubik [13] proposes a dynamic partitioning scheme that predicts and exploits the transient behavior of latency-critical applications, while increasing the LLC space for batch (throughput) applications. These schemes execute a run-time fine-grain partitioning algorithm which modifies the space allocated to these applications in the shared LLC. They all have significant hardware overhead, and can not guarantee determinism. For a small-scale embedded multicore, we propose a simplistic static spatial partitioning scheme for the LLC interference channel. Moreover, we consider all interference channels, and not just the shared LLC.

Memory Controller Partitioning: Wang et al. [11] statically allocated each bank in the memory controller to each application. The static periodic time interval is decided based on the frequency of DRAM row-buffer access and dead-time between two subsequent accesses. The disadvantage of this approach is that each application waits while the channel bandwidth is time-multiplexed, even though the application which has control of the channel doesn't have any request. So, there is under-utilization of the shared channel to the DRAM via memory controller. The worst-case latency is double the optimal case where the idle-time is more for the memory controller. Muralidhara et al.[15] dynamically partitioned the DRAM banks based on the LLC miss characteristics and row-buffer locality of each applications which resolved interference at the DRAM but did not resolve interference at the

shared memory controller in a multicore. The above mentioned works periodically reallocate the shared memory controller bandwidth constantly but do not remove the interference in the interference channels by overlooking shared LLC.

NoC partitioning: Interference-Aware Bus is proposed in previous works [17], [20]. They schedule requests according to their criticality, and allow them to be sent in non-consecutive bus slots. This scheme bounds the delay that a request can suffer due to bus interference. Kelter et al. [21] proposed TDMA accesses on the system bus. Due to lack of scalability for bus-based NoC multicore architectures, A. Burns et al. [19] proposed a revised NoC wormhole switching protocol. It implements a number of prioritized virtual channels for concurrent tasks. However, due to their hardware limitations and interaction of other shared resources, these schemes are not able to isolate the on-chip traffic.

VII. Conclusion

We propose an approach to remove on-chip interference via a set of methods to spatio-temporally partition shared multicore resources. This approach guarantees deterministic execution of safety-critical applications on a multicore, while delivering better than WCET performance. Overall, on average ISOLATION has an average overhead of $1.07\times$ as compared to WCET at $5.52\times$. In fact, the proposed ISOLATION scheme delivers at par performance of the uncontrolled sharing, however, it delivers concurrent execution of safety critical applications deterministically on the multicore. This paper is a step towards ensuring deterministic execution on multicore architectures for safety-critical systems, such as avionics and automobiles.

Acknowledgment

Support for this project was provided by the UTC Institute for Advanced Systems Engineering (UTC-IASE) at the University of Connecticut. Any opinions expressed herein are those of the authors and do not represent those of the sponsor.

References

[1] Nuzzo, Pierluigi, et al. "A contract-based methodology for aircraft electric power system design." Access, IEEE 2 (2014): 1-25.

[2] Qingchuan Shi; Hijaz, F.; Khan, O., "Towards efficient dynamic data placement in NoC-based multicores," in Computer Design (ICCD), IEEE 31st International Conference on , vol., no., pp.369-376, 6-9 Oct. 2013.

[3] Miller, J.E.; Kasture, H.; Kurian, G.; Gruenwald, C.; Beckmann, N.; Celio, C.; Eastep, J.; Agarwal, A., "Graphite: A distributed parallel simulator for multicores," in High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on , vol., no., pp.1-12, 9-14 Jan. 2010.

[4] Krishna and Poovendran. "Aviation cyberphysical systems: foundations for future aircraft and air transport." Proceedings of the IEEE 101.8 (2013): 1834-1855.

[5] Certification Authorities Software Team(CAST), CAST-32 Multicore Processors, May 2014(Rev 0) http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf

[6] Wilhelm, Reinhard, et al. "The worst-case execution-time problemoverview of methods and survey of tools." ACM Transactions on Embedded Computing Systems (TECS) 7.3 (2008): 36.

[7] Nowotsch, Jan, and Michael Paulitsch. "Leveraging multi-core computing architectures in avionics." Dependable Computing Conference (EDCC), 2012 Ninth European. IEEE, 2012.

[8] Bui, Bach Duy, et al. "Impact of cache partitioning on multi-tasking real time embedded systems." Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on. IEEE, 2008.

[9] Slijepcevic, Mladen, et al. "Time-analysable non-partitioned shared caches for real-time multicore systems." Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE. IEEE, 2014.

[10] Hardavellas, Nikos, et al. "Reactive NUCA: near-optimal block placement and replication in distributed caches." ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. ACM, 2009.

[11] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh,"Timing Channel Protection for Memory Controllers." *20th IEEE International Symposium on High Performance Computer Architecture*(HPCA), February 2014.

[12] Moinuddin. K Qureshi, Yale N. Patt. "Utility Based Cache Partioning: A low-overhead, high-performance, runtime mechanism to partition shared caches." In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*(MICRO), 2006

[13] Harshad Kasture and Daniel Sanchez. "Ubik:Efficient Cache-Shariing with strict QoS for Latency-Critical Workloads." *Proceedings of the seventeenth edition of ASPLOS on Architectural support for programming languages and operating systems*,(ASPLOS),2014

[14] Masab Ahmad, Farrukh Hijaz, Qingchuan Shi, and Omer Khan. "CRONO: A Benchmark Suite for Multithreaded Graph Algorithms Executing on Futuristic Multicores." In *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, 2015

[15] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning." *Proceedings of the 44th International Symposium on Microarchitecture (MICRO), Porto Alegre, Brazil, December 2011*

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations." In *Intl Symposium on Computer Architecture, 1995.*

[17] T. Ungerer et al., "Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability." in IEEE Micro, vol. 30, no. 5, pp. 66-75, Sept.-Oct. 2010.

[18] Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments, http://www.artemis-emc2.eu

[19] A. Burns, J. Harbin and L. S. Indrusiak, "A Wormhole NoC Protocol for Mixed Criticality Systems." Real-Time Systems Symposium (RTSS), 2014 IEEE, Rome, 2014, pp. 184-195.

[20] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, Hardware support for WCET analysis of hard real-time multicore systems. 36th Int. Symposium on Computer Architecture, June 2009.

[21] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, Bus-aware multicore WCET analysis through TDMA offset bounds. in 23rd Euromicro Conf. on Real-Time Systems (ECRTS), July 2011, pp. 312.

[22] QorIQ T2080 and T2081 communication processors, http://cache.nxp.com/files/32bit/doc/fact_sheet/T2080FS.pdf