10-19-2016

WCET Analysis for Concurrent Execution of Multiple Applications on Safety Critical Embedded Multicores

Kartik Lakshminarasimhan
University of Connecticut, kartik.lakshminarasimhan@uconn.edu

WCET Analysis for Concurrent Execution of Multiple Applications on Safety Critical Embedded Multicores

Kartik Lakshminarasimhan

B.E., Anna University, 2014

A Thesis

Submitted in Partial Fulfillment of

Master of Science

At the

University of Connecticut

2016

APPROVAL PAGE

Master of Science Thesis

WCET Analysis for Concurrent Execution of Multiple Applications on Safety Critical Embedded Multicores

Presented by

Kartik Lakshminarasimhan, B.E, Electronics and Communication

Major Advisor _____
Dr.Omer Khan

Associate Advisor _____
Dr.Marten Van Dijk

Associate Advisor _____
Dr.John Chandy

University of Connecticut

2016

ACKNOWLEDGMENTS

I am thankful to my thesis advisor Dr.Omer Khan for giving me an opportunity to work with him and guiding me for my thesis research. I would also thank my labmates Qingchuan, Masab, Farrukh, Kamran, Halit and Hamza for healthy discussions related to Computer Architecture. Finally, I would thank my Guru and undergraduate advisor Prof.Venteswaran for motivating me to pursue Graduate Studies in Computer Engineering. Last but not the least, I thank my family for their continual support.

DISSERTATION DRAFT FOR THE MASTER'S DEGREE

WCET (Worst Case Execution Time) Analysis for Concurrent Execution of Multiple Applications on Safety Critical Embedded Multicores

Kartik Lakshminarasimhan

Abstract: The computing hardware layer for modern embedded systems (such as internet-of-things) must be able to execute many concurrent applications under tight deterministic execution guarantees to meet the real-time requirements. Those time-critical cyber-physical systems are designed with respect to their Worst Case Execution Time (WCET). Single-chip multicores are attractive for such embedded systems due to their lightweight form factor. However, multicores aggressively share hardware resources, leading to interference that in turn makes hard to determine the WCET for multiple concurrent applications.We propose an approach to use a set of methods to spatio-temporally partition shared multicore resources, analyse the WCET and quantify its overheads.

5

Contents

List of Figures

List of Tables

1 Introduction

The distributed and embedded software in modern cyber-physical systems (such as the internet-of-things and automobiles) is increasingly becoming a first order design constraint. For example, the internet-of-things (IoT) paradigm integrates many smart devices/objects that are managed using a diverse set of software applications. Due to the interlinking of the physical and the cyber worlds, the IoT environment poses tight and continuous execution constraints. This envisioned paradigm must integrate several unrelated or loosely related software applications formerly distributed across multiple systems, and lessen the system weight and energy consumption. Traditionally, such applications are executed on discrete processors. The key question we seek to answer in this research is whether multicore technology can be deployed in such time-critical cyber-physical systems.

In case of a multicore processor this implies a single 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 nondeterministic timing and power behaviors across applications, leading to loss of performance guarantees. On the other hand, multicore's efficient form factor offers desirable capabilities for such embedded systems.

Traditionally, worst case execution time (WCET) [8] analysis is used to ensure determinism by not violating it during the application scheduling process. WCET is defined as the execution time when minimum shared resources are deployed. For example, in a multicore the shared LLC causes interference that leads to unpredictable application behavior. Thus, it becomes difficult to measure WCET for a multicore setup. As a consequence, the lack of on-chip data locality leads to undesirable performance penalty. This can potentially

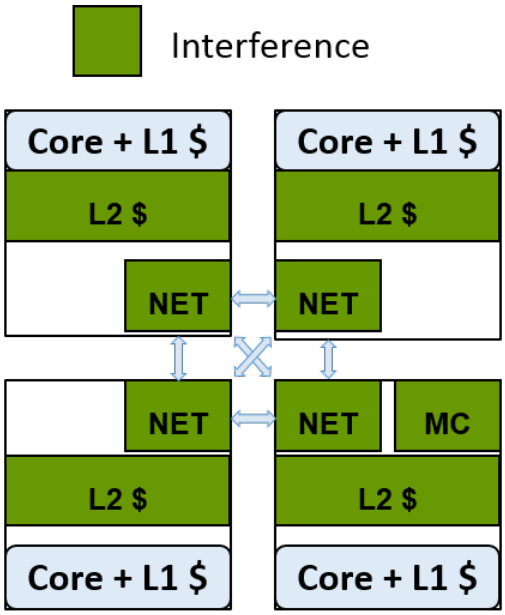

Figure 1: Baseline system with Shared LLC and Memory Controller

limit the utility of deploying multicore technology in time-critical cyber-physical systems.

This thesis report eschews the use of traditional course grain WCET schemes and proposes a holistic approach to remove all on-chip interference channels via a set of lightweight methods that spatio-temporally partition shared multicore resources. The objective is to quantify the WCET bounds and enable the deployment of concurrent applications. Our approach is a step towards satisfying deterministic performance guarantees for an embedded system that deploys multicore processor technology.

2 Partitioning of Shared Resources

2.1 Overview

Figure 1 shows the baseline multicore system that we consider. The system has four cores, where the compute pipeline and the level-1 (L1) caches are private for each core. Hence, an application mapped to a core has private access to these resources without any interference. The Network-on-Chip (NoC) is a 2D Mesh network where the data packets sent by each core have shared routers between the source and destination. The Last Level Cache (L2$) and Memory Controller (MC) are logically shared across applications as well. These interference channels 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.

2.2 Spatial Partitioning of Last Level Cache

There is no state-of-the-art partitioning scheme that performs temporal partitioning of the LLC and guarantee zero interference. It is because of the latency overhead in flushing the LLC data to off-chip memory and bringing it back during each partition's execution is prohibitive. 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 [14]. This distributed nature of the LLC places the onus on partitioning and placement of the data in the LLC, also know as the non-uniform cache access (NUCA) [16]. We exploit this NUCA capability to spatially partition the LLC slices across the concurrent applications. Although further fine-grain partitioning can be done at the cache way and/or set granularities, we leave such exploration for future work.

Each LLC slice consists of multiple physical cache banks. Before the 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.

2.3 Temporal Partitioning of Memory Controllers and Network on Chip (NoC)

The memory controller and NoC are temporally partitioned among the applications as temporal time slots. A slot is defined as the certain number of time cycles during which a specific application can only occupy the shared resource. A round robin temporal scheduler queue is shown in the Figure 2 where the different application slots haven *taken* and *available* slots. The Q_{Delay} is the amount of time spent by the respective application's data in the queue.

$$Queue_d = T_{sch} - T_{arr} \tag{1}$$

where,

$Queue_d$ = Queue Delay, the time spent by the applications' request in the queue

T_{arr} = Arrival time, the timestamp that request from an application arrives and enters the queue

T_{sch} = Request Schedule time, the timestamp that request from an application is scheduled from the queue.

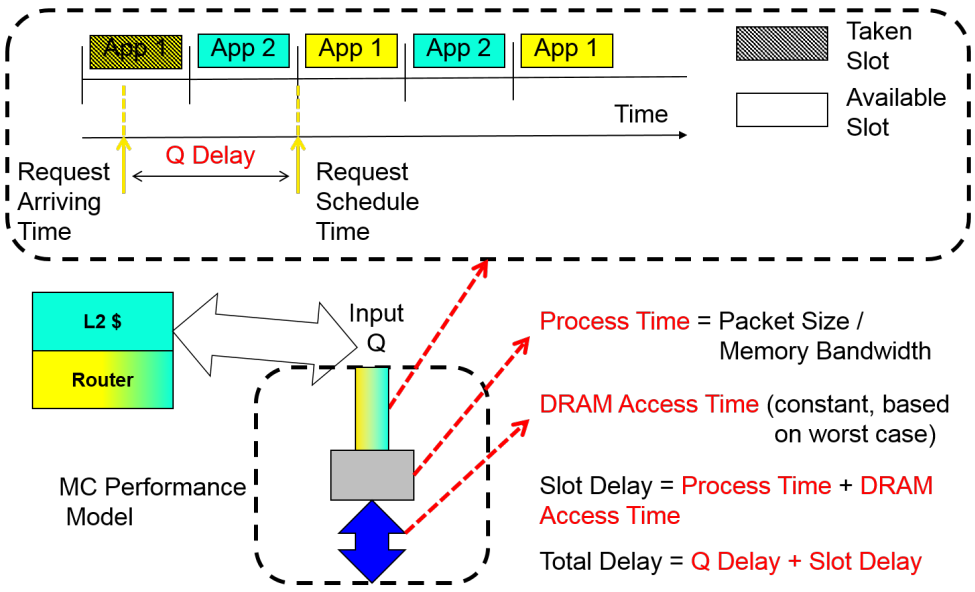

Figure 2: Memory Controller accessing the DRAM with Temporally partitioned input queue

2.3.1 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. Before the start of execution, memory controller bandwidth is divided among the concurrent applications. Each core then gets its allocated memory bandwidth, temporally partitioning the memory controllers among the applications. The shared memory controller is temporally partitioned such that each application gets its time slot similar to the scheme by Wang et al. [17]. Figure 2 shows a memory controller accessing the DRAM with temporally partitioned input queue. The following equations 2 3 4 explain the delay incurred by the memory controller performance models.

$$Time_{proc} = Size_p / M_{bw} \tag{2}$$

$$Slot_d = Time_{pro} + Time_{acc} \tag{3}$$

$$Total_d = Queue_d + Slot_d \tag{4}$$

where,

$Time_{proc}$ = Process Time, the ratio of $Size_p$ packet size and M_{bw} Memory Bandwidth

$Slot_d$ = Slot delay, the sum of $Time_{pro}$ Process Time and $Time_{acc}$ DRAM Access Time(constant)

$Total_d$ = Total delay, sum of Queue Delay and Slot Delay

2.3.2 Network on Chip(NoC) Temporal Partitioning

The routers in the NoC are temporally partitioned for a certain interval which allows certain application to pass through the router in a certain time-slot. The router delay depends the queue delay, constant delay and link delay. Figure 3 shows the detailed architecture a router that can send/receive in all four directions (North,South,East and West) and from the tile as well. The router receives input from the core/L1, shared last level cache and memory controller through the injection queue. The router injection queue is temporally partitioned with time slots assigned for each application. Its helps to regulate the multi-application traffic across all 4 directions. The equation below gives the delay depending upon which the temporal slot is allocated.

$$Total_d = Router_d + Queue_d + Serialization_d + Link_d \tag{5}$$

where, $Total_d$ = Total Delay ,the delay from the NoC temporal Partitioning

$Router_d$ = Router Delay, the delay of the router(constant)

$Queue_d$ = Queue Delay is the delay from Equation 1

$Serialization_d$ = Serialization Delay is the processing delay similar to Equation 2

$Link_d$ = Link delay is the delay of the link(constant)

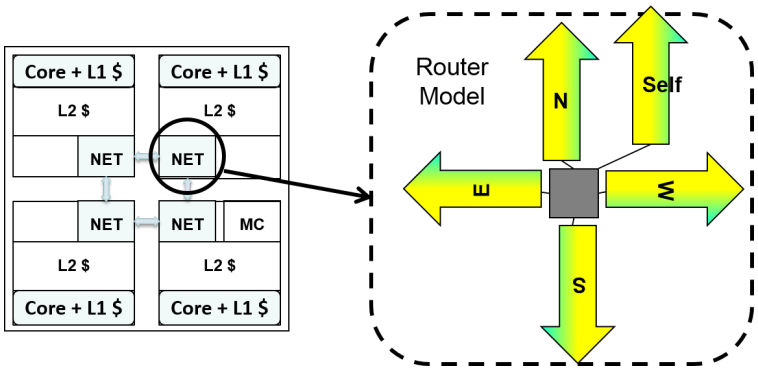

Figure 3: Baseline system with Shared LLC, Network on Chip and Memory Controller

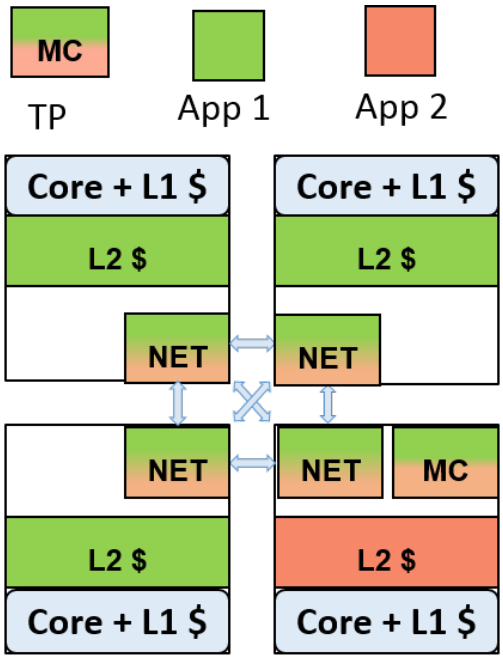

Figure 4: An Example of partitioning schemes: Spatial partitioning of LLC , temporal partitioning of NoC and memory Controller

Figure 4 shows an example of spatial partitioning for the baseline system shown in figure 1. $LLC(3,1)$ implies application 1 is allocated three LLC slices, whereas application 2 is allocated one LLC slice. The system has a single memory controller, it is temporally partitioned (MC_TP) for deterministic time intervals.

2.4 Realistic Worst Case Execution Time(WCET) model

There has been a lot of literature about WCET model for single core systems but not for multicore systems. Its hard to find the WCET in a multicore scenario because of the shared interference channel in a multicore. A realistic WCET model is configured such that all the LLC slices are spatially partitioned with application getting equa number of slices, the Network on Chip and the shared Memory Controller is temporally partitioned. This gives a realistic WCET model because all the interference channels are partitioned such that minimizing the role of interference in the WCET analysis.

3 Experimental Methology

3.1 Performance Modeling

All experiments are performed using our modified version of the Graphite multicore simulator [3]. We modified Graphite 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. 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. A memory controller can also be placed on any of the tiles. Cache coherence is implemented using the directory based MSI protocol. The on-chip NoC is a point-to-point network with a 2-cycle per hop (router + link) delay. We also account for the contention delays and appropriate pipeline latencies associated with loading and unloading a packet onto the network.

3.1.1 Baseline

The baseline 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. The baseline performance is the maximum completion time of the two applications when executed on two physically separate multicores. All results of concurrent application executions are normalized to this baseline.

3.1.2 Uncontrolled Sharing (U_Sharing)

The uncontrolled sharing (U_Share) 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. Its interesting when a critical and a non-critical application are mixed together . There is no guarantee in this setup that the critical application would finish within the specific deadline.

3.1.3 Worst Case Execution Time (WCET)

The notion of WCET is that in order to eliminate interference, no shared on-chip channels can be used. Two applications are isolated on-chip by only allowing accesses to the private components. Thus it is configured in a way that all applications receive equal L2 slices, the shared NoC and memory controller being temporally partitioned.

3.1.4 LLC and Memory Controller Partitioning Schemes

Our proposed spatio-temporal partitioning scheme is implemented for the LLC and memory controller interference channels and NoC routers. For each application combination, there are 3 ways to partition LLC slices (**(LLC(1,3), LLC(2,2), LLC(3,1)**). Because of 1 memory controller, the only way to partition it is temporal partitioning. The NoC is also temporally partitioned. In **MC_100**, the memory controller is temporally partitioned such that each application gets a slot of 100 cycles. In **NoC_10** each the router ports are partitioned temporally with each application getting an interval of 10 clock cycles. In **MC_none** and **NoC_none** configurations, the memory controller and NoC are not partitioned.

3.2 Benchmarks

The benchmarks and their input sizes are shown in Table 2. We use three SPLASH-2 benchmarks (FFT, RADIX, and CHOLESKY) [23], matrix multiplication (MM), MultiLayer Perceptron(MLP), SSSP(Single Source Shortest Path), Pagerank benchmarks to represent applications for safety-critical systems. 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

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	8 KB, 4-way Assoc., 1 cycle
L1-D Cache per core	8 KB, 4-way Assoc., 1 cycle
L2 Cache per core	32 KB, 8-way Assoc., 8 cycle
	Inclusive, NUCA
Cache Line Size	64 bytes
Num. of Memory Controllers	1
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	1 flit
(Src, Dest, Addr, MsgType)	
Cache Line Length	8 flits (512 bits)

Table 1: Architectural parameters for evaluation.

Application	**Problem Size**
RADIX	524288 integers
FFT	2^{18} complex doubles
MM	256×256 matrix, 16×16 blocks
CHOLESKY	tk23.0
SSSP	512 node graph with 16 neighbours for each node
MLP	784 inputs, 256 hidden layers, 10 output layers
PAGERANK	131072 node graph with 16 neighbours for each node

Table 2: Benchmarks and their input sizes.

where model reduction is done during the state-space analysis. 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 such data according to some threshold, such as magnitude. MultiLayer Perceptron is a neural network Machine Learning algorithm which is to used in modern day Advanced Driver Assistance System(ADAS). The MLP algorithm used in this thesis predictions the handwritten digits with an accuracy of 99 percent. SSSP(Single Source Shortest Path algorithm) is an implementation of Dijkstra's algorithm pathfinding algorithm which is used in aircrafts,ships and future autonomous vehicles. Pagerank is a Graph-based prediction algorithm used in recommendor systems in various platforms.

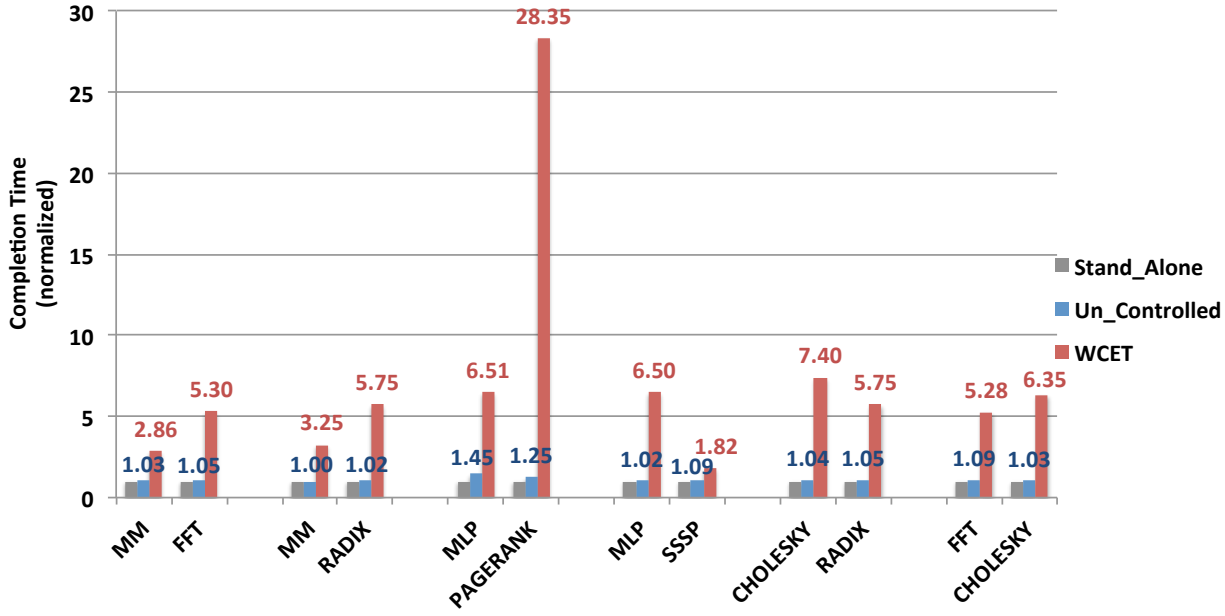

Figure 5: Completion time of uncontrolled sharing and WCET of benchmark combinations.

4 Results

Figure 5 shows the completion time of uncontrolled sharing and WCET for different benchmarks combinations. The results are normalized to the stand alone. In most of the cases, the uncontroller sharing is worse than stand along. This is due to on-chip interference. Because of benchmarks' inherent characteristics, WCET slow downs very from ∼3X to ∼28X. The main reason of this is because benchmarks have diverse demands of on-chip resources. WCET spatial-temporal partitions the interference channels, the resources allocated to a benchmark may not fullfill its demand. Furthermore the temporal parition schemes have significant performance overheads, due to reduced NoC and memory controller bandwidth. Benchmarks with heavy no-chip traffic and more off-chip accesses would be affect significantly. PAGERANK performs much worse for WCET, because it has large amount of cache misses in current system setup, which generates requestes in NoC and can result in accessing the DRAM through memory controller.

Figure 6 shows the results for the MM-FFT combination with different partition schemes. When only LLC is spatial, shown as L2_XX_MC_none_NoC_none, the results of

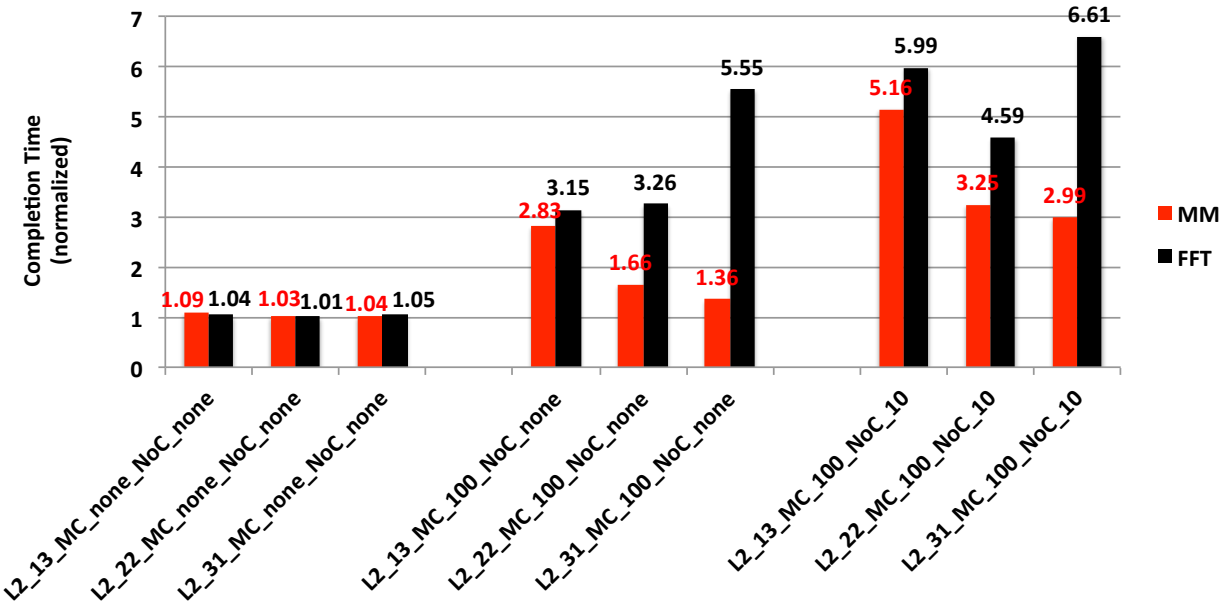

Figure 6: Partitioning combinations for MM-FFT

different number LLC slices for each benchmark do not vary much. This is due to the relatively high memory controller bandwidth. LLC misses caused by limited LLC space can access the DRAM without substantial overhead. When memory controller is also partitioned (temporally), shown as L2_XX_MC_100_NoC_none. The performance overhead is more evident due to limited memory controller bandwidth. This also makes different LLC spatial partitions distinct, because contentions at memory controller makes the LLC misses much more costly. Similar trend is followed when NoC is also temporally partitioned, shown as L2_XX_MC_100_NoC_10.

Figure 7 shows the results for the MLP-PAGERANK and MLP-SSSP combinations. For the same MLP benchmark, when it is running with different benchmarks (PAGERANK and SSSP) without any partitioning schemes, shown as Un_Controlled, its completion time is inconsistent. This justifies the nondeterministic due to interference. It also shows that when only LLC is spatially partitioned, shown as L2_XX_MC_none_NoC_none, interference still exist. When all the interference channels are parttioned, shown as L2_XX_MC_100_NoC_10, MLP completion time becomes stable without affected by the other application. This justifies that spatio-temporally partition all shared resources is the practical scheme to eliminate on-

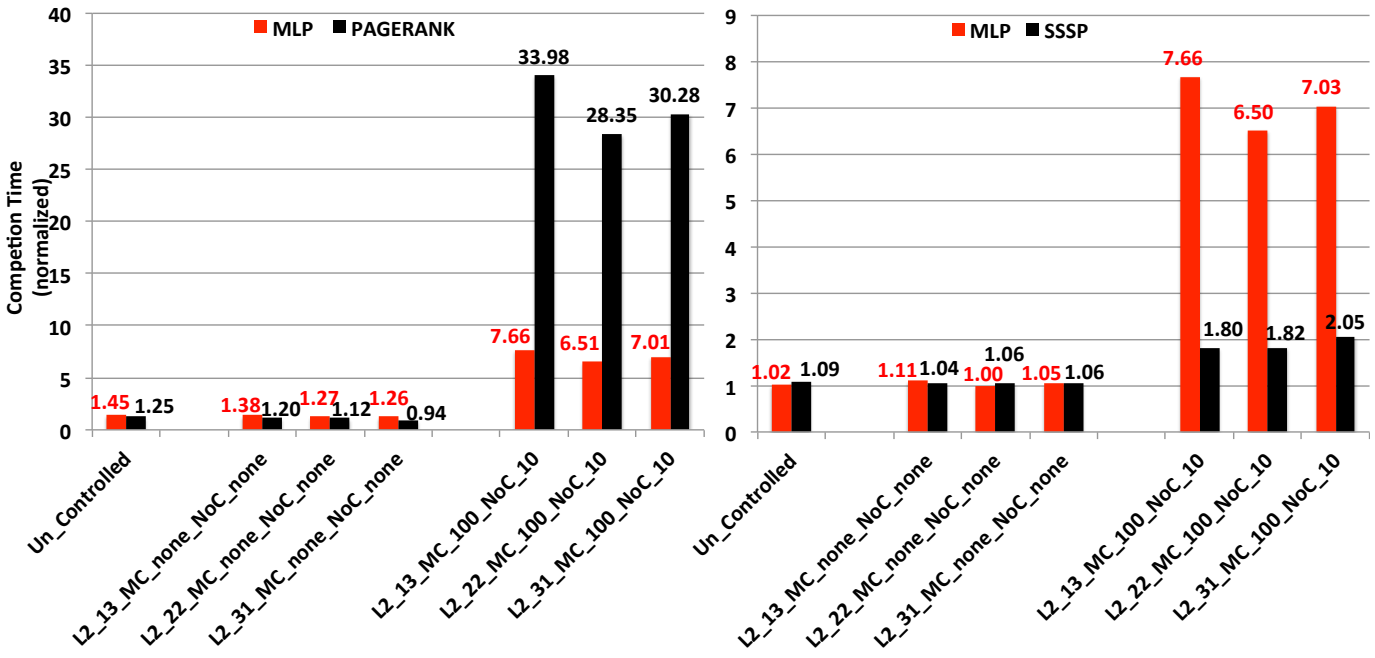

Figure 7: Application isolation through partition schemes example: MLP

chip interference.

5 Conclusion

Thus, this thesis proposes an approach to use a set of methods to spatio-temporally partition shared multicore resources, analyse the WCET and quantify its overheads. This thesis is a step towards deployment of multicores in safety-critical system, such as avionics and automobiles.

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] Cyber Physical Systems Design Challenges Report No. UCB/EECS-2008-8 http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

[7] NASA Avionics Architectures for Exploration (AAE) and Fault Tolerant Computing, `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140008709.pdf`

[8] 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.

[9] Samuel, Andrew Waterman, and David Patterson. "Roofline: an insightful visual performance model for multicore architectures." Communications of the ACM 52.4 (2009): 65-76.

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

[11] 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.

[12] 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.

[13] Yan, Kaige, and Xin Fu. "Energy-efficient cache design in emerging mobile platforms: the implications and optimizations." Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition. EDA Consortium, 2015.

[14] Tilera *TileGx*9, http://www.tilera.com/files/drim_TILE-Gx8009_PB036-02_WEB_7663.pdf

[15] OCTEON II CN68XX Multi-Core MIPS64 Processors , http://www.cavium.com/OCTEON-II_CN68XX.html

[16] 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.

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

[18] Daniel Sanchez and Christos Kozyrakis.Vantage: Scalable and Efficient Fine-Grain Cache Partitioning, *Proc. of the 38th annual International Symposium in Computer Architecture* (ISCA-38),2011

[19] 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

[20] Beckmann, Nathan and Sanchez, Daniel Jigsaw: Scalable Software-defined Cache *Proc. of International Conference on Parallel Architectures and Compilation Techniques* (PACT-22), 2013

[21] 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

[22] 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*

[23] 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.*