Timing Anomalies in Multi-core Architectures due to the Interference on the Shared Resources

Hardik Shah, Kai Huang and Alois Knoll
Department of Informatics VI, Technical University Munich,
85748 Garching, Germany.
{shah,huangk,knoll}@in.tum.de

Abstract— Timing anomalies in single-core processors have been theoretically explained and well understood phenomenon. This paper presents new timing anomalies which occur in multi-core architectures due to the interference on the shared resources. We derive formulation to capture these anomalies and provide practical evidences using real applications from the Mälardalen WCET benchmark suit executing on NIOS II multi-core architecture on an Altera FPGA.

I. Introduction

Timing anomaly is a *counter intuitive* timing behavior. The term was coined by Lundqvist & Stenström [1]. They observed that in a dynamically scheduled processor, a cache hit at certain execution point could lead to longer execution time than a cache miss at the same point. Thus, the timing anomaly for processor architectures is defined as, *A processor architecture is said to be timing anomalous when a locally favorable event (e.g. cache hit) could result in a globally unfavorable event (e.g. longer execution time) and vice versa.* The formal definition of timing anomaly is provided by Reineke et al [2].

Simple processors could also exhibit timing anomalous behavior [3]. The occurrence of timing anomalies can be analyzed using static worst case execution time (WCET) analysis techniques [4]. However, these events are rarely (or never) observed in real life. This argument is often used against the static WCET analysis techniques. Hence, it is important to present real-life evidences of the presence of timing anomalies.

In multi-core architectures, the shared resource interference caused by accesses from the co-existing applications increases shared resource access latencies for the application-under-test. This leads to longer execution time of the application. It is intuitive that if the co-existing applications are very aggressive in accessing the shared resource then the application-under-test experiences higher latencies to the shared resource and the execution time increases accordingly. The other intuition is, the higher the number of aggressive co-existing applications, the higher the latencies to the shared resource. In this paper, we show that certain applications behave counter intuitively to these intuitions.

The major contributions of the paper are as follows. i) We prove that in the presence of aggressive accesses from the co-existing applications, some applications-under-test benefit and experience less than the average case latencies to the shared resource. ii) We prove that in the presence of the aggressive accesses, for some applications-under-test, shared resource latencies under less number of interfering applications could be more than the shared resource latencies under more number of interfering applications. iii) We also provide practical evidence of these timing anomalies using real applications from the Mälardalen WCET benchmark suit executing on the NIOS II based multi-core architecture on Altera Cyclone III FPGA. The paper focuses on the round robin arbiter which is one of the most popular *starvation free* arbiters and a default component of many off-the-shelf interconnect architectures [5, 6]. However, other *starvation free* arbiters are discussed in Sec. VI.

The paper is organized as follows. Sec. II provides existing work related to this paper. Sec. III provides necessary background information. Sec. IV provides formalism from which the timing anomalous behavior is inferred. Sec. V provides practical evidences of timing anomalous behavior. Sec. VI discusses the timing anomalies in other *starvation free* arbiters and Sec. VII concludes the paper.

II. Related Work

This paper has two dimensions, i) Interference analysis in multi-core architectures and its effect on the WCET. ii) Timing anomalies. We address the related work on both the topics in this section.

Lv et al [7] propose to build Timed Automata (TA) models of concurrently executing applications using abstract interpretation. These TA models are then combined with the shared resource arbiter's TA model. The combination is analyzed using a model checker to find the WCET of each application considering the maximum interference among them. Pellizzoni et al [8] use cache activity trace of the application-under-test and upper bound on I/O traffic. These inputs are analyzed using real-time calculus to calculate WCET of the application-under-test considering maximum interference from the I/O traffic. Both of these approaches take the knowledge of the co-existing applications into the account.

On the contrary, our previous work [9, 10, 11] and Paolieri et al [12] analyze applications in isolation for shared SDRAM interference. Here, the worst possible behavior from co-existing applications is assumed, including the faulty behavior. All the above mentioned works assume timing-anomaly-free architecture.

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

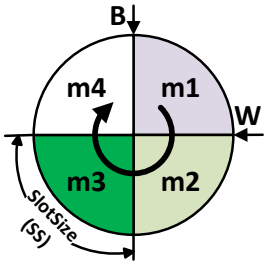

Fig. 1.: Graphical View of the Round Robin Arbiter

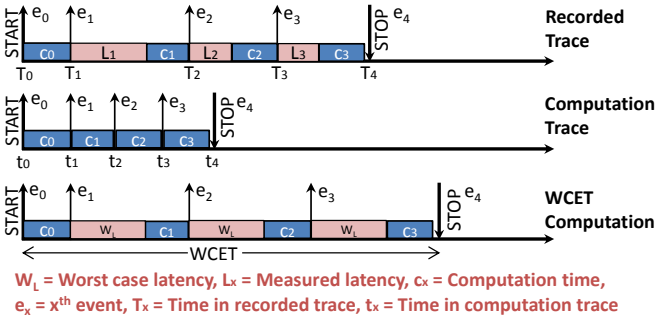

Fig. 2.: WCET Computation using the Computation Trace

Li et al [13] model timing anomalous processor for the WCET analysis. They analyze the interaction between Basic Blocks (BB) and its effect on the instruction cache state. Kirner et al [14] identify a *new* timing anomaly that arises due to the parallel decomposition. The parallel decomposition is used for hardware state space reduction. These approaches target single-core architectures.

Recently, there has been some work on the WCET analysis in multi-cores in the presence of timing anomalies. Chattopadhyay et al [15] provides the first unified analysis that takes various micro-architecture components into account. Interaction among these components is analyzed to estimate the WCET of applications on a timing anomalous multi-core architecture. Kelter et al [16] investigate when a BB will start executing with respect to the statically assigned slot to access the shared bus. The shared bus access latency is estimated by analyzing whether the access from the BB lays in the current slot or in next slots. Both, [15] and [16] use TDMA as a shared bus arbiter.

All the above mentioned work related to timing anomalies explain it hypothetically without real evidences. In this paper, we derive formulation which is used to infer timing anomalies due to the shared resource interference in multi-core architectures. Moreover, we provide real-life evidences of their presence.

III. Background

In this section, we provide some background information in order to facilitate discussion in the later sections.

A. Work Conserving Round Robin Arbitration

The Fig. 1 depicts the Round Robin (RR) arbitration graphically. Under the RR scheme, the shared resource contenders are assigned fixed number of slots in a virtual ring depending on their bandwidth requirements. Although, our analysis is valid for any slot allotment, for simplicity, we consider one slot per contender without loss of generality. The figure shows four contenders (master1, master2, master3 and master4) in the ring. Here, we assume that these contenders are processor cores executing independent applications and the shared resource is shared main memory. These cores access the shared main memory when a cache miss occurs. *Throughout the paper, we use master and core terms interchangeably. Similarly, we use memory and shared memory interchangeably.*

The arbiter continuously searches for a master which wants to access the memory in a clock-wise direction. We call this master an active master. As soon as an active master is encountered, it is granted the memory for a predefined maximum number of clock cycles (`SlotSize - SS`). The `SS` is big enough to accommodate the burst issued for one cache-line fill. After the granted master finishes its burst access, the search process resumes from the next slot in the ring. Thus, the memory is always occupied as long as there is at least one active master (hence the name, "*work conserving*" or "*greedy TDMA*").

Now, let us assume that the application-under-test is executing on `m1` and the `SS` is same for all masters. For this architecture, an access request from `m1` experiences the worst case completion latency ($W_L = 4 \times SS$) if it is issued when the arbiter pointer is at **W** in Fig. 1 *AND* all other masters utilize their slots. Similarly, an access request experiences the best case completion latency ($B_L = 1 \times SS$) if it is issued when the arbiter pointer is at **B** in the figure. If the exact location of the arbiter pointer *OR* activity of other masters are unknown, the completion latency could be any number in the range $[B_L, W_L]$.

Intuitively, the average case latency, $A_L = (B_L + W_L)/2$. *In this paper, by latency of an access, we mean completion latency of an access (scheduling latency + time required to complete the access). Moreover, we also assume that the application-under-test executes on `m1`.*

B. Computation Trace

The computation trace is an execution trace of an application path where cache misses are denoted by timeless events. The motivation behind the timeless events is the following. In the shared memory architecture, the shared main memory is accessed when a cache miss occurs. The contention on the shared memory delays service to this memory access. Typically, the collision of cache misses on the shared memory is extremely difficult to predict. Moreover, the delay in service also delays the subsequent cache misses (memory accesses) of the application-under-test by the same amount. This causes difficulties in estimating the worst case interference and its impact on the WCET. To avoid these difficulties, at first, we remove all the latencies related to the memory accesses. This means, there is no interference at all and the shared memory takes zero cycles to respond. Later, theoretically calculated worst possible latency is added for each cache miss.

Computation trace, depicted in the Fig. 2, can be easily obtained through simulation. At first, occurrence time $(T_0, T_1, ...)$ of each cache miss event $(e_0, e_1, ...)$, and its

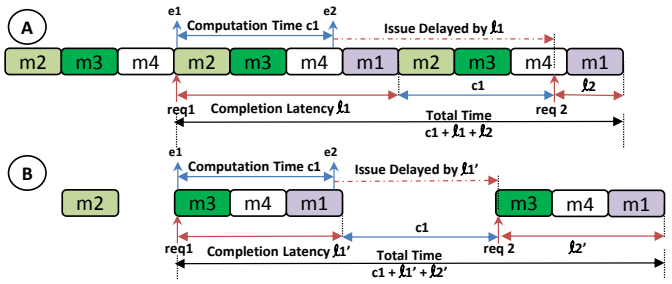

Fig. 3.: In **case A**, over all execution time is lower than in **case B** although the accesses generated by the co-existing application is more aggressive in **case A** than in **case B**

experienced latency $(L_0, L_1, ...)$ are recorded in a trace by executing the application on cycle accurate simulation models of processor and memory. Later, these latencies are removed and each event is shifted towards left in time. The resulting trace is the computation trace. Now, to compute the WCET considering the worst case interference, each cache miss event in the computation trace is annotated by the worst possible latency (W_L) and all the subsequent accesses are shifted to the right. Now, the computed WCET contains the effect of the worst possible interference the application may experience.

It must be noted that the latencies in the recorded trace depend on shared memory interference at the time of measurement. However, the computation trace remains unchanged[1] when the same path is executed multiple times, provided that each time we start the application from the same cache state[2] and use the same data as an input.

Note that the WCET computed using this method is actually WCET of the *path* being executed on the application-under-test. Moreover, the method considers only shared resource interference as an execution time modifying source. Practically, applications have multiple paths through execution and caches, pipelines, branch predictors etc. contribute heavily to the execution time deviation. The core contribution of the paper is to identify the timing anomalies originating from the shared resource interference and to provide practical evidences of its presence. Therefore, we do not present analysis of caches, pipelines, branch predictors, execution path etc in this paper and fully concentrate on interference analysis.

C. Latencies under round Robin Arbitration

In this subsection, we analyze different latency scenarios for the computation trace. As explained before, the computation trace remains constant for multiple runs of the application, however, the interference scenarios can be different resulting in different execution times. Fig. 3 depicts two different interference scenarios. In scenario A, the co-existing applications generate aggressive accesses (uninterrupted accesses/interference) to the shared memory while in scenario B, the co-existing applications generate sparse accesses. The computation trace in the figure contains two cache miss events `e1` and `e2`. The computation time between these events is `c1`.

In scenario A, the request 1 (corresponding to event `e1`) is issued just after the arbiter has scheduled `m2`. Since cores `m2`, `m3` and `m4` are generating uninterrupted accesses, the request 1 from `m1` can be scheduled only after $3 \times SS$. After request 1 is scheduled, it needs another SS to complete. Hence, request 1 has latency of $l1 = 4 \times SS$, which is the worst case latency (W_L). After requests 1 is served, the `m1` does computation for `c1` amount of clock cycles. During this time, the `m1` executes from caches and on-chip registers and does not send any request to the shared memory. However, the co-existing cores send uninterrupted accesses to the shared memory and keep on rotating the arbiter pointer. Thus, when request 2 (corresponding to `e2`) is issued, the arbiter pointer is close to the next scheduling opportunity of `m1`. Hence, the latency of request 2, $l2 << W_L$.

In scenario B, although the co-existing applications generate sparse accesses, the total execution time is longer than that of scenario A. Similarly, another scenario can be presented where both the accesses experience the worst case latencies which results into the real WCET[3].

IV. Latency Analysis under α Interference

This section focuses on the uninterrupted interference and derives equations for experienced latencies under it. The first subsection defines the α interference and the second subsection provides analysis of experienced latencies.

A. α Interference

Definition: *The α interference is defined as the uninterrupted interference produced by α number of co-existing masters.* Under α interference, the rotation of the arbiter pointer becomes deterministic since accesses from the co-existing masters are deterministic (uninterrupted or no access at all). Here, except `m1`, on which the application is being executed, other masters either continuously utilize their slots or they do not utilize their slots at all.

The α interference occurs in real life in the following scenarios, i) Immediately after reset or after a new task is scheduled on co-existing applications, there are high number of cache misses. At this point, the co-existing applications send many accesses to the shared resource in a relatively short period of time. ii) When some of the co-existing applications generate aggressive traffic e.g. DMA and the remaining are idle for a short period of time. iii) When some of the co-existing masters have a stuck-at-fault on the request line and other masters are idle for a short time. It is clear that the α interference could occur randomly for a short period of time in real life. However, applications with relatively short life times (typical

[1] In this paper, we assume absence of jitter in occurrence of cache misses due to operating system, floating point unit, pipeline etc.

[2] In order to fully concentrate on the *new* timing anomalies originating from interference, we assume classical timing-anomaly-free architecture in this paper. Certainly, these *new* timing anomalies are valid in the presence of the classical ones as well.

[3] In this paper, we focus only on the effect of the interference on WCET and consider the effects of other components such as caches, pipe-lines, branch predictors etc as constant.

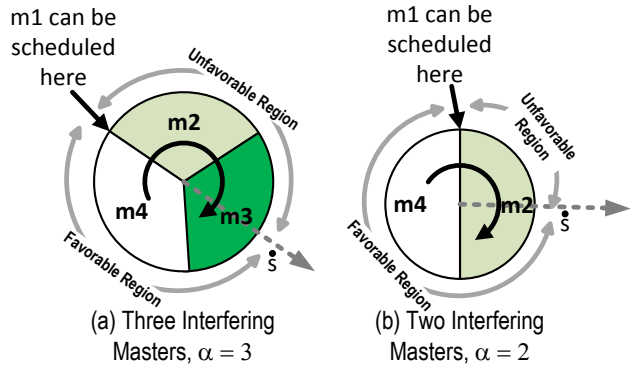

Fig. 4.: Deterministic Rotation of the Arbiter Pointer (`SlotSize - SS` not to the scale)

hard real-time control applications) could experience it during their entire execution. In the remaining sections, we will show that, counter intuitively, these uninterrupted accesses from co-existing masters could also be beneficial to some applications.

B. Analysis

Fig. 4 depicts two scenarios, $\alpha = 3$ and $\alpha = 2$. In $\alpha = 3$ scenario (same as Fig. 3 **scenario A**), all other masters `m2`, `m3` and `m4` do uninterrupted accesses to the shared resource. In $\alpha = 2$ scenario, only masters `m4` and `m2` do uninterrupted accesses; `m3` is idle. Hence, there are only two slots in Fig. 4(b). Note that there are total four masters in the system and the theoretical values of B_L, W_L and A_L are derived considering all masters in the system, *irrespective* of number of active masters.

We denote the latency of i^{th} access under α interference as deterministic latency (DL^i_α) since it is derived assuming the deterministic rotation of the arbiter pointer. Its value can be obtained using the following equation.

$$DL^i_\alpha = (\alpha + 1) \times SS - \{c_{(i-1)} \bmod (\alpha \times SS)\} \quad (1)$$

Here, $c_{(i-1)}$ is the computation time between i^{th} and $(i-1)^{th}$ cache miss events in the computation trace (Fig. 2). Let $\Theta^{(i-1)}_\alpha = \{c_{(i-1)} \bmod (\alpha \times SS)\}$,

$$DL^i_\alpha = (\alpha + 1) \times SS - \Theta^{(i-1)}_\alpha \quad (2)$$

Consider average of all DL^i_α values is $\overline{DL_\alpha}$ and the average of all Θ^i_α values is $\overline{\Theta_\alpha}$ (note that $\overline{DL_\alpha}$ and $\overline{\Theta_\alpha}$ represent average values over entire execution path). Hence, equation (2) can be re-written as,

$$\overline{DL_\alpha} = (\alpha + 1) \times SS - \overline{\Theta_\alpha} \quad (3)$$

$\overline{DL_\alpha}$ is the average experienced latency when the application is executed in the presence of α interference.

Recall that $A_L = (B_L + W_L)/2$, $B_L = 1 \times SS$ and $W_L = N \times SS$, N is total number of master in the system. From this information, the average-case latency of the i^{th} access, A^i_L, is given by the following equation,

$$A^i_L = \frac{N+1}{2} \times SS \quad (4)$$

Since value of A_L in (4) depends only on constant numbers, the average of all A^i_L values is,

$$\overline{A_L} = \frac{N+1}{2} \times SS \quad (5)$$

Equations (1) to (5) are used to infer counter intuitive timing behavior. First, let us prove couple of lemmas.

Lemma 1 $\exists c_{(i-1)}, \alpha : A^i_L > DL^i_\alpha$. *In other words, for a combination of α, and $c_{(i-1)}$, it is possible that the observed latency of i^{th} access (DL^i_α) is less than the theoretical average-case latency of the i^{th} access (A^i_L).*

Proof: Since $\Theta^{(i-1)}_\alpha = \{c_{(i-1)} \bmod (\alpha \times SS)\}$, $\Theta^{(i-1)}_\alpha \in [0, (\alpha \times SS - 1)]$. Thus, if $c_{(i-1)} = n(\alpha \times SS) - 1$, $\Theta^{(i-1)}_\alpha = \alpha \times SS - 1$, $n \in \mathbb{N}^+$.

Putting $\Theta^{(i-1)}_\alpha = \alpha \times SS - 1$ in (2), $DL^i_\alpha = SS + 1$. Here, $DL^i_\alpha < A^i_L$, $\forall N > 1, SS > 2$. Hence, the lemma holds. □

The lemma 1 leads to a counter intuitive observation. It proves that an access from an application could experience less than the average-case latency under α interference. Moreover, the latency does not depend on the absolute value of $c_{(i-1)}$, rather on the $\Theta^{(i-1)}_\alpha$. This phenomenon is explained graphically in Fig. 4. Here, favorable and unfavorable regions are depicted. If the application has all c_i such that $\overline{\Theta_\alpha}$ lies in the favorable region, the average experienced latency ($\overline{DL_\alpha}$) is less than the average-case latency ($\overline{A_L}$). The sweet point ($\dot{s}$) between the boundaries of the regions is derived by the following equation.

$$\dot{s} = W_L - A_L \quad (6)$$

The proof of lemma 1 can also be used to derive condition for experiencing the worst case latency for all accesses is, $\forall i, c_i = n(\alpha \times SS), n \in \mathbb{N}^0$ AND $\alpha = (N-1)$. The application fulfilling this condition always requests exactly when the next master to it in the ring is scheduled (at point **W** in Fig. 1) and all other masters in the system utilize their allocated slots.

In real-life, it is difficult to find an application which fulfills the above mentioned conditions for the worst case latencies. However, applications that experience less than the average-case latencies under α interference are not rare (see Sec. V).

Lemma 2 $\exists c_{(i-1)} : DL^i_{\check{\alpha}} > DL^i_{\hat{\alpha}}$, $\check{\alpha} < \hat{\alpha}$. *In other words, under α interference, for a particular value of $c_{(i-1)}$, less number of interfering masters could result in longer latency than more number of interfering masters.*

Proof: Let, $c_{(i-1)} = (\check{\alpha} \times SS)$. From equation (1),

$$DL^i_{\check{\alpha}} = (\check{\alpha} + 1) \times SS = \check{\alpha} \times SS + SS \quad (7)$$

Again using equation (1) and since $\check{\alpha} < \hat{\alpha}$, $c_{(i-1)} \bmod (\hat{\alpha} \times SS) = \check{\alpha} \times SS$. Hence, the value of $DL^i_{\hat{\alpha}}$ can be given by the following equation,

$$DL^i_{\hat{\alpha}} = (\hat{\alpha} + 1) \times SS - (\check{\alpha} \times SS) \quad (8)$$

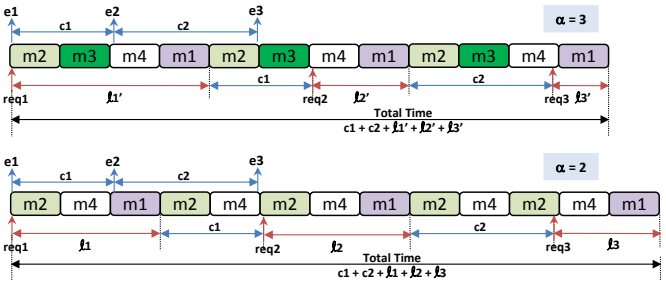

Fig. 5.: Experienced Latencies under $\alpha = 3$ and $\alpha = 2$ Interference

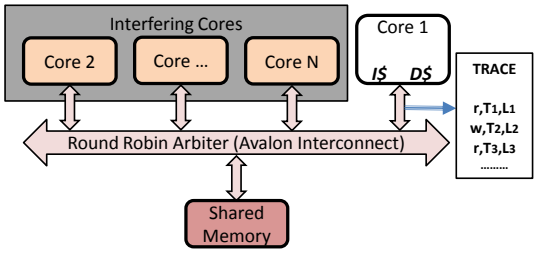

Fig. 6.: Test Set-up

$$DL^i_\alpha = (\hat{\alpha} - \check{\alpha}) \times SS + SS \tag{9}$$

From equation (7) and (9), $DL^i_{\check{\alpha}} > DL^i_{\hat{\alpha}}, \forall \hat{\alpha} : \check{\alpha} < \hat{\alpha} \leq 2\check{\alpha}$. Hence, the lemma holds. □

The Fig. 5 provides supporting example for the lemma 2. Here, for the given application, $\alpha = 2$ interference produces higher latencies than $\alpha = 3$ interference.

V. Test Cases

The goal of this section is to provide real life evidences of timing anomalies inferred by the lemmas of previous section. We did intensive testing by executing applications from the Mälardalen WCET benchmark suit[4] [17] on Altera NIOS II multi-core architecture as depicted in Fig. 6. We experimented with different cache sizes and different number of cores. Note that with the variation in cache size, the computation trace also varies (c_i and total number of cache miss). Hence, for each new cache configuration, new set of traces for each application was created using cycle accurate simulation models.

We did experiments on Altera Cyclone III FPGA Development board. Altera provides cycle accurate simulation models of processor and memory. These models were used to capture the recorded trace of each application under different cache configurations. For recording trace, connection point between core 1 and the shared memory was probed as depicted in the figure. Here, core 1 executes one of the applications and all other cores execute a dummy application (similar to [18, 19, 20]) that uninterruptedly accesses the shared memory. We started with total 4 cores in the system and step by step increased the total number of cores to 8. Note that, unlike variation in cache size, variation in number of cores does not change the computation trace since computation trace of an application is independent of experienced latency.

[4]We chose only single path applications from the suit to avoid path analysis since the path analysis is not our contribution.

Benchmark	OET	ACET	WCET	$\overline{DL_3}$
cover	12207	11586	14274	24.95
crc	104331	101196	108396	26.57
duff	6777	5936	7364	28.63
edn	**360574**	**361342**	**391834**	**19.91**
expint	16573	16469	16781	23
fac	1129	1107	1227	24
fdct	**22079**	**24173**	**32453**	**18**
fibcall	1110	1098	1182	24
jane	832	813	921	25
jfdcint	28209	28107	33891	21.97
minver	158910	142289	189893	25.95
prime	196676	186533	228197	25
quart	224508	204366	271530	24.99
ud	38248	34815	46443	24.93

TABLE I
: Execution times in clock cycles under $\alpha = 3$ Interference

	α	$\alpha = 3$	$\boldsymbol{\alpha = 4}$	$\boldsymbol{\alpha = 5}$	$\alpha = 6$	$\alpha = 7$
cover	OET	10717	**11686**	**11046**	11942	13662
	$\overline{DL}$	23	**30**	**27**	32	42

TABLE II
: Execution times in Clock Cycles under varying α interference

A. Test 1

In this experiment, we used instruction and data caches of 512 Bytes each. Total of $N = 4$ cores were used, hence, $\alpha = N-1 = 3$. As depicted in Fig. 2, we inserted the worst case (W_L) and the average-case (A_L) latencies for each cache miss to obtain the WCET and the ACET (Average Case Execution Time) of the applications, respectively.

Table I depicts the results. The first column depicts Observed Execution Time (OET) of the applications. It is clear from the table that most of the applications experienced more than the average-case latencies. However, the `edn` and the `fdct` applications experienced less than the average-case latencies under this hardware configuration. These applications did majority of accesses in the favorable region of Fig. 4 (note the lower values of $\overline{DL_3}$ for these applications). Thus, the results provide evidence for the lemma 1 of the previous section.

B. Test 2

The evidence of the lemma 2 was captured when we increased the cache size to 1 KB. We started from total 4 cores ($\alpha = 3$) and step-by-step increased the total number of cores to 8 ($\alpha = 7$). After adding each core, the OET was measured. For the `cover` application, execution times are listed in the Table II. This application exhibits higher execution time under $\alpha = 4$ interference than under $\alpha = 5$ interference. Due to its shared memory access pattern (cache miss pattern), it experiences longer latencies in the presence of less number of aggressive masters than presence of more number of aggressive masters, which is counter intuitive.

Note that the cache configuration is different in this experiment and the previous experiment. Hence, the `cover` application has different OET in the tables for $\alpha = 3$. Except the `cover` application, other applications followed intuition and experienced more latencies as α was increased.

VI. Discussion

These anomalies are not limited to the round robin arbiter. In the budget based arbiters, such as Credit Controlled Static Priority (Ccsp), Priority-based Budget Scheduler (Pbs) and Dynamic Priority Queue (Dpq) this phenomenon can also be observed. Under these arbiters, all masters are assigned a unique budget to access the shared resource per unit time. If a master consumes its budget, it is termed ineligible and cannot access the shared resource until the unit time expires [9, 11]. Thus, due to the aggressive accesses, if co-existing applications become ineligible, shared resource accesses from the application-under-test experience low latencies. Again, it depends on access pattern of the application-under-test. If the application-under-test itself is aggressive then it quickly becomes ineligible and following accesses experience high latencies. Thus, similar to classical timing anomalies, these anomalies are application dependent. Only under the Tdma and the Priority Division [21] arbiters, these timing anomalies are absent.

The timing anomalies presented in this paper depend on the shared resource access pattern (cache miss pattern) of the application. Modification in cache line size, associativity, cache size etc modifies the shared resource access pattern. The modification can either remove these timing anomalies or introduce them. This makes the measured execution time in the presence of uninterrupted interference a highly unreliable Wcet candidate.

VII. Conclusion

This paper has identified two new timing anomalies which occur due to the interference on shared resources in multi-core architectures. The anomalies are as follows: i) Some applications could benefit from aggressive co-existing applications and experience less than the average-case latencies while accessing the shared resource. ii) Some applications experience more latencies in the presence of less number of aggressive co-existing applications than in the presence of more number of aggressive co-existing applications while accessing the shared resource. The anomalies are inferred from formulation and the real-life evidences of their presence are provided using applications from the Mälardalen Wcet benchmark suit. The experiments are conducted on the Altera Nios II multi-core with shared memory architecture implemented on Altera Cyclone III Fpga development board.

Collisions of cache misses of concurrently executing applications on a shared main memory is extremely difficult to predict. Moreover, such prediction is limited to the particular set of application execution paths and precise phase of their starting time. Thus, this prediction is not useful for Wcet analysis, practically. To estimate Wcet in the presence of unpredictable interference, it is intuitive to let the co-existing applications generate uninterrupted accesses to the shared memory and assume that this is the highest possible interference. However, this paper concludes that the measured execution time of application-under-test in the presence of uninterrupted shared resource accesses from co-existing applications could be highly optimistic and well below the Wcet considering the theoretical worst case interference.

Acknowledgment

This work was funded by German BMBF projects ECU (13N11936) and Car2X (13N11933).

References

[1] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *Proc. RTSS*, 1999.

[2] J. Reineke, *et al.* A definition and classification of timing anomalies, wcet 2006.

[3] Gernot Gebhard. Timing Anomalies Reloaded. In *WCET 2010*, Dagstuhl, Germany.

[4] Reinhard Wilhelm *et al.* The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), 2008.

[5] Advance microcontroller bus architecture (amba).

[6] Avalon interface specifications.

[7] Mingsong Lv *et al.* Combining abstract interpretation with model checking for timing analysis of multicore software. In *Proc. RTSS*, 2010.

[8] R. Pellizzoni *et al.* Impact of peripheral-processor interference on wcet analysis of real-time embedded systems. *IEEE Transactions on Computers*, 2010.

[9] H. Shah *et al.* Bounding WCET of Applications Using SDRAM with Priority Based Budget Scheduling in MPSoCs. In *Proc. DATE*, 2012.

[10] Dynamic priority queue: An SDRAM arbiter with bounded access latencies for tight WCET calculation. Technical report, 2012.

[11] Hardik Shah, Alois Knoll, and Benny Akesson. Bounding sdram interference: detailed analysis vs. latency-rate analysis. In *Date '13, Grenoble, France.*

[12] Marco Paolieri *et al.* An Analyzable Memory Controller for Hard Real-Time CMPs. *Embedded Systems Letters, IEEE*, 2009.

[13] Li. Xianfeng *et al.* Modeling out-of-order processors for software timing analysis. In *RTSS, 2004.*

[14] R. Kirner *et al.* Precise worst-case execution time analysis for processors with timing anomalies. In *ECRTS '09.*

[15] S. Chattopadhyay *et al.* A unified wcet analysis framework for multi-core platforms. In *RTAS 2012.*

[16] T. Kelter *et al.* Bus-aware multicore wcet analysis through tdma offset bounds. In *ECRTS*, 2011.

[17] J. Gustafsson *et al.* The Mälardalen WCET benchmarks – past, present and future.

[18] M. Fernández *et al.* Assessing the suitability of the ngmp multi-core processor in the space domain. In *EMSOFT '12.*

[19] P. Radojković *et al.* On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *TACO*, 2012.

[20] J. Nowotsch *et al.* Leveraging multi-core computing architectures in avionics. In *EDCC '12.*

[21] H. Shah, A. Raabe, and A. Knoll. Priority division: A high-speed shared-memory bus arbitration with bounded latency. In *Proc. DATE*, 2011.