Modelling Bus Contention during System Early Design Stages

David Trilla[†,‡], Carles Hernandez[†], Jaume Abella[†], Francisco J. Cazorla[⋆,†]
[†] Barcelona Supercomputing Center (BSC). Barcelona, Spain
[‡] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
[⋆] Spanish National Research Council (IIIA-CSIC). Barcelona, Spain.

***Abstract*—Reliably upperbounding contention in multicore shared resources is of prominent importance in the early design phases of critical real-time systems to properly allocate time budgets to applications. However, during early stages applications are not yet consolidated and IP constraints may prevent sharing them across providers, challenging the estimation of contention bounds. In this paper, we propose a model to estimate the increase in applications' execution time due to on-chip bus sharing when they simultaneously execute in a multicore. The model works with information derived from the execution of each application in isolation, hence, without the need to actually run applications simultaneously. The model improves inaccuracy with respect to the existing model, and tends to over-estimate. The latter, is very important to prevent that, during late design stages, applications miss their deadline when consolidated into the same multicore, causing costly system redesign.**

I. Introduction

AUTOSAR [2] in the automotive domain and Integrated Modular Avionics [1] in the avionics allow system integrators subcontracting the development of some functionality to different software providers (SWPs) and providing them with a time budget in which applications must fit. During Early Design Phases (EDP) each SWP develops its applications incrementally. On every release the SWP validates application's execution time against its assigned time budget. The complexity of timing validation significantly increases with multicores, since the timing behaviour of one application depends not only on its own characteristics, but also on other applications (likely developed by other SWPs) running in parallel in the multicore. In particular, the accesses that an application makes to hardware shared resources impact other application's execution time. Hence, modelling multicore contention, whose impact has been shown to be very high [17][18], during EDP it is fundamental to gain confidence on whether each application will fit its budget. Otherwise, the integration process will be exposed to the risk of costly changes during Late Design Phases (LDP) due to task overruns. To make things worse, intellectual property restrictions may prevent applications to be exchanged among SWPs.

It is also worth noting that the real-time embedded industry (e.g. avionics, rail, space and automotive) is very cautious adopting new hardware/software. This results in the adoption of multicore processors that are simpler than those used in the mainstream market on top of which software can be verified at a reasonable cost.

To reduce hardware procurement costs, in many projects SWPs are provided with a virtualized environment, which allows them to develop and test their applications without the need to acquire a board. This Virtual-Machine (VM) based approach – in use by several system integrators including the European Space Agency (ESA) for several projects [14], [23] – enables performing functional verification of applications. However, due to the lack of timing information, the timing verification process cannot be carried out. Recently, this problem has been alleviated by light-weight a *contention model* that derives estimates to applications' execution time for multicore systems [7]. That model takes as input an execution profile (EP) of the application that comprises information extracted from its execution in isolation (e.g. cache miss rates, and instruction mix). This allows EPs to be exchanged among SWPs without violating IP restrictions. The integrator, who knows the application schedule, determines which applications run simultaneously. The contention model is executed on the EP of co-running application deriving the slowdown they are going to suffer. This helps determining whether they fit their budget, taking proper actions early in the design phase if this is not the case.

Contribution. We propose a bus contention model for round-robin (*roro*) and first-in first-out (*fifo*) policies. These policies, together with *tdma* – less adopted by industry – are the most common time-predictable arbitration policies. For instance, *roro* is implemented in the ARM Cortex-A9 [24] and the NXP MSC8122 [21] processors. In particular the main contributions of this paper are:

1) We make an in-depth analysis of the bus model proposed in [7] for *roro*. We show the main assumptions made by that bus model and analyze the inaccuracies they cause. We show reasons behind the trend towards under-estimation of that model (i.e. the model's predicted value is smaller than the observed (real) value). This can result in having applications at LDP that do not fit their budget, causing significant costs: either the system integrator has to change the schedule of applications granting more budget to those missing their deadlines, or the SWPs are required to change their applications to fit their assigned budget. As part of our analysis we provide insights on why the model in [7] also works for *fifo*. In particular we show how expected contention is higher with *fifo* than with *roro*.

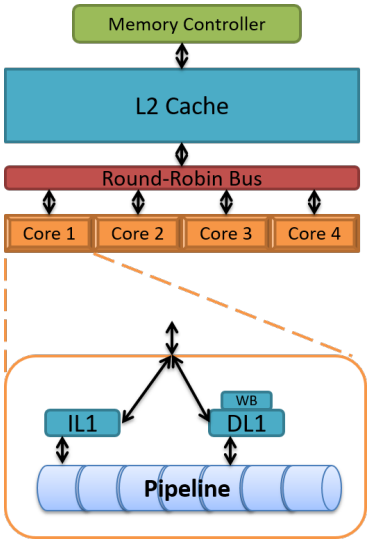

Fig. 1: Block Diagram of a generic 4-core architecture.

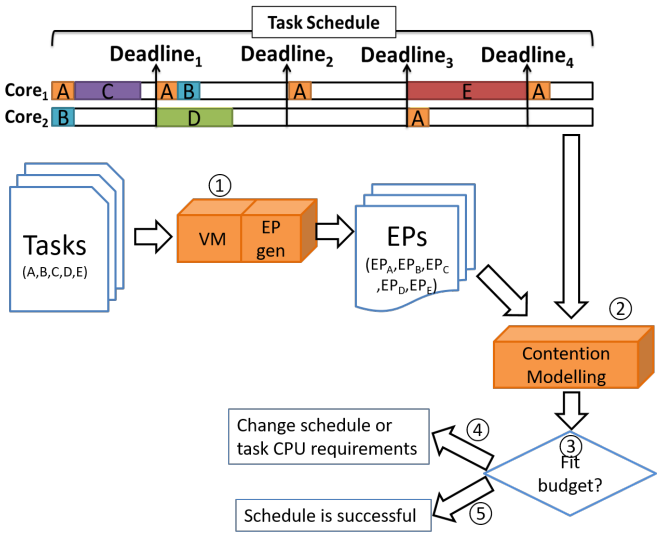

Fig. 2: Application steps for a 2-core setup

2) We propose a new bus model that captures the main inaccuracies of that in [7]. First, unlike [7], our model captures the fact that buses support different request types and each can take a different latency. Our model factors in this information when modelling bus contention. And second, while [7] accounts for the (self) load introduced on the bus by the task under analysis, we factor it out and focus only on the load that contender tasks put on the task under analysis. As a result the proposed model reduces the inaccuracy of the previous model and more importantly tends to over-estimate.

We evaluate the accuracy of our model in a cycle-accurate simulator: as representative real-time multicore we model the Cobham Gaisler's four-core NGMP processor [4][3] – a strong candidate for ESA's future missions. For the NGMP (comprising per-core data and instruction caches; and a shared bus, L2 and memory controller) the contention in the on-chip bus has been identified to have major impact on the timing estimates [7]. Our evaluation with EEMBC Automotive benchmarks [19] and two Space applications shows that our model achieves higher accuracy when modelling bus contention than the previous approach and inaccuracies are typically from above (over-estimation). Further, it does not require changing the EP derived for each application, keeping the cost low in the applicability of the overall approach.

The rest of this paper is structured as follows. Section II introduces contention modelling in early design stages. Section III presents our analysis of the bus model in [7]. Section IV presents our new bus model that is evaluated in Section V. Section VI is devote to the related work. Section VII shows the main conclusions of this study.

II. Contention Modeling in EDP

We focus on bus-connected multicore processor as the 4-core processor architecture sketched in Figure 1. We assume that each core has a private instruction cache (iL1) and data cache (dL1), and a global (shared) unified second level cache (uL2). Cores and caches are connected with a *roro* or *fifo* AMBA bus. A memory controller acts as interface between the processor cores and memory. This multicore processor architecture is similar to that of the ARM Cortex A9 and Cobham Gaisler's NGMP.

Bus accesses are not split so the bus is locked by a master until the request is processed. Therefore, the bus contention model also captures contention tasks suffer in memory. Requests to the bus may have different durations, each generating different inter-task contention. For instance for the case of the NGMP bus reads that either hit (*l2h*) or miss (*l2m*) on the L2 cache and bus writes that either hit (*s2h*) or miss (*s2m*) on the L2 cache. In our setup the maximum latency of each request type measured in processor cycles is as follows: $lat_{l2h} = 10$, $lat_{s2h} = 3$, $lat_{l2m} = 24$ and $lat_{s2m} = 10$.

During EDP, the system integrator defines a set of tasks [1] to perform the required functionality along with a schedule that assigns time budgets for each task. Budgets are implemented by partitioning time into windows – usually with a cyclic executive scheduler. Time windows are then assigned to the different tasks. In single-core architectures, SWP perform the time analysis of their tasks from EDP in isolation since tasks only suffer some overheads. Interestingly, the interaction in input/output resources among tasks is handled via forcing that the input/output operations of a task occur during its assigned window or during a specific period designated for that purpose.

In multicore, however, the timing behaviour of a task depends on the other tasks executing at the same time [17][18]. This complicates determining whether a given application from a SWP fits its assigned budget since until other applications are not integrated in the multicore its contention

[1]We focus on single-task applications so we use the terms task and application interchangeably.

impact cannot be factored in. If no other means are provided, integration occurs in LDP, in which any change is costly.

The execution time of a task τ_j in a multicore (et_j^{muc}) can be broken down into its execution time in isolation (et_j^{solo}) and the overhead created by multicore contention (Δt_j^{cont}):

$$et_j^{muc} = et_j^{solo} + \Delta t_j^{cont} \quad (1)$$

Deriving Δt_j^{cont} for a task τ_j requires understanding how its contender tasks $c(\tau_j) = \{\tau_i, \tau_k, \tau_l\}$ use shared resources. In [7] authors proposed an approach based on the idea of Execution Profiles (EP) that comprise information extracted from the execution of the application in isolation. For each application that can be executed together – information that the system integrator derives from the task schedule – an EP is generated. Then, two models, the bus contention model (BCM) and the cache contention model (CCM), combine the EP of those applications to derive Δt_j^{cont} for each of the applications. In the next subsections we explain the main steps in the contention modelling process, which are depicted in Figure 2 for a dual-core scenario. For the tasks developed by the different SWPs ($\{A, B, C, D, E\}$) an EP is derived (1). From the task schedule (see top left part of Figure 2) each SWP knows the other tasks against which its task will run. Then, the contention models are used to determine how much each task is slowed down due to contention (2). This allows each SWP to assess whether its applications fit their budget (3), so the schedule is deemed as successful (5). Otherwise, the SWP needs to either change applications to reduce their computation requirements or ask the system integrator to grant longer time to applications (4).

The desired properties for the contention model are:

(**p1**) It has to work with information derived from executions in isolation. Multicore-related information, e.g. how the requests of different tasks overlap, can only be derived when applications are actually integrated, which occurs in LDP.

(**p2**) The model has to be fast so that different schedules can be analyzed and let, early in the design, the integrator and SWPs gain confidence on the feasibility of the timing.

(**p3**) While relatively good timing estimates are desirable, no particular figure has been reported for the required accuracy in timing predictions during the EDP. For the NGMP, contention has shown to be as high as 5.5x for EEMBC automotive[12] (i.e. contention cause the benchmark to take 5.5 times longer than if run in isolation). We take as reference the accuracy results reported provided by the model in [7] that range from 1.4 to 0.6 the actual values.

(**p4**) Violations during LDP of applications' assigned timing budgets may require costly application re-coding or even changing the scheduling plan. This, of course, may significantly increase the overall product (system) cost and time-to-market. Therefore, obtaining early estimates that tend to over-estimate is desirable.

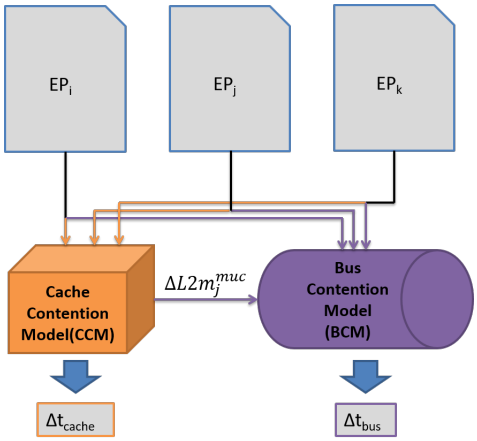

Fig. 3: Execution Profile Derivation Process.

A. Execution Profile Generation

Multicore contention overhead Δt_j^{cont}, which has been shown to be very high [17][18], can significantly impact the scheduling plan defined by the system integrator. To derive Δt_j^{cont} we use a CCM and BCM [7]. The overall modelling process starts by extracting information about request duration; and from the execution of each task τ_i in isolation. For the latter, the execution is performed on the virtualized environment, which allows extracting detailed execution information, known as the EP. In our case, the EP for a task contains: per-type instruction count and cache information (hits, misses, time between accesses).

B. Contention Modelling

Once EPs have been obtained, the model executes the CCM that predicts both, the increment in number of misses that τ_j suffers due to the contention created by its co-runners ($\Delta L2m_j^{muc}$) and the execution time increment caused by those extra misses (Δt_j^{cache}), see Figure 3. Then the BCM – the focus of this paper – is executed deriving the increment in time τ_j suffers due to bus contention (Δt_j^{bus}). This allows deriving et_j^{muc} as shown in Equation 2.

$$et_j^{muc} = et_j^{solo} + \Delta t_j^{cont} = et_j^{solo} + \left(\Delta t_j^{cache} + \Delta t_j^{bus}\right) \quad (2)$$

Δt_j^{bus} is obtained by adding the time the bus spends serving bus accesses, i.e. uL2 cache hits, uL2 misses in isolation and uL2 contention misses ($\Delta L2m_j^{muc}$) – the latter is obtained from the CCM.

III. Deconstructing the existing BCM

Equation 3 presents the bus contention model (BCM) introduced in [7] in which τ_j is the task under analysis, $c(j)$ are its $Nc-1$ contender tasks, and Nc is the number of cores. While the model was proposed for *roro*, in Section IV-D we show how this model and our model also cover *fifo* arbitration policies and the expected accuracy.

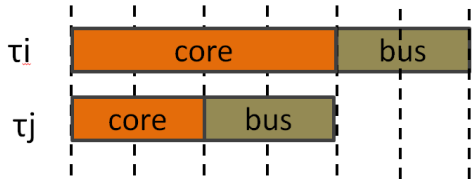

Fig. 4: Non overlap scenario among τ_i's and τ_j's bus usage times.

$$\Delta t_j^{bus} = etb_{isol}^j \times \left(\frac{1}{1 - \frac{ubus_{c(j)}}{1+ubus_{c(j)}}} \right) \quad (3)$$

In this equation, τ_j's bus utilization in isolation (etb_{isol}^j) is increased by a factor (the second multiplicand) that captures the contention created by contender tasks. The denominator of the second multiplicand accounts for the bus utilization of contender tasks – that already factors in the contention of L2 misses. It builds on $ubus_{c(j)}$ that is defined as:

$$ubus_{c(j)} = \sum_{i=0, i \neq j}^{N_c - 1} \frac{etb_{isol}^i}{et_{isol}^i} \quad (4)$$

etb_{isol}^i is the time the contender uses the bus in isolation and et_{isol}^i its execution time in isolation. As a result, $1 - ubus_{c(j)}/(1 + ubus_{c(j)})$ provides a measure of how often the bus is available for the task under analysis.

$$avail_j = 1 - \frac{ubus_{c(j)}}{1 + ubus_{c(j)}} \quad (5)$$

Hence Equation 3 can be defined as:

$$\Delta t_j^{bus} = etb_{isol}^j \times \left(\frac{1}{avail_j} \right) \quad (6)$$

The inverse of the availability, i.e. $1/avail_j$, represents how many arbitration rounds the task under analysis τ_j is expected to wait until being granted access to the bus. This factor is used to increase its in-isolation bus utilization time.

This BCM makes several assumptions that impact its accuracy as we analyze next.

A. Assumption A1: worst-case overlapping

BCM assumes worst-case overlapping in the bus among requests of accessing tasks. For instance, let us assume two tasks, τ_j and τ_i. The BCM assumes that *all* bus accesses of τ_j conflict with those of τ_j. In reality, however, depending on when τ_i and τ_j bus accesses occur, they may not conflict, as show in Figure 4.

At request level, this translates into making the pessimistic assumption that each contender request delays τ_j. That is, every time τ_j is ready to perform an access, there is an access of each contender task ready to use the bus, and the latter are given priority by the arbiter.

Let md_{bus} be the maximum delay (measured in number of cycles) that a request can wait to get access in a *roro* arbitrated bus. This is given by $md_{bus} = (Nc - 1) \times l_{bus}$, where Nc is the number of cores and l_{bus} the latency of the bus. In a 4-core multicore this results in $md_{bus} = 3 \times l_{bus}$. For instance, when contenders have a combined utilization of 3, this results in an availability (see Equation 5) of 0.25, i.e. $1 - 3/4$. This in turn results in the model assuming that on every access τ_j waits 3 rounds before being granted access to the bus.

B. Assumption A2: similar request duration

The BCM assumes that τ_j's requests and its contenders requests use the bus for the same duration once they are granted access. This can be better illustrated with an example. Let us assume 4 tasks, each having back-to-back accesses to the bus, i.e. utilization of 100%. In this scenario Equation 4 results in $ubus_{c(j)} = 3$ and Equation 5 in $avail_j = 1 - 3/(1+3) = 0.25$. Hence, with 4 contending tasks and *roro* arbitration each core receives one slot every 4 arbitrations. However, with this factor the model assumes that the number of cycles each task is using the bus is the same, while in reality this depends on the actual duration of each request. For instance if τ_j generates 4-cycle requests to the bus and its contenders requests of one cycle. Then, τ_j with its 25% of the slots uses the bus for 4 cycles whereas its contenders use it only 1 cycle each, see Figure 5. Hence, the fraction of bus bandwidth received by c_j is 57.1% on average (4/7), while the rest of the cores receive 14.3% each (1/7).

C. Assumption A3: self bus contention

The BCM computes the increase in bus cycles due to contention. However, when multiplying the inverse of the availability by the etb_{isol}^j (see Equation 6), the model accounts as part of the contention factor the arbitration round in which τ_j is granted access to the bus. This should not be effectively accounted for as an increase in bus cycles of τ_j requests. This is an unnecessary source of pessimism introduced by this BCM that our model corrects.

IV. New Bus Model

The proposed contention model takes into account the analysis performed on the BCM [7] as follows.

A. Handling A1: Worst-case overlapping

According to the required property *p1* defined in Section II, during EDP applications are, in general, not yet integrated, but SWPs proceed with their application development in isolation. Hence, applications from different SWPs cannot be run simultaneously on the multicore and information from that execution cannot be derived. This is why the EPs include only information of the run of each task in isolation. In this regard, contention models for EDP cannot determine how task are going to interleave in reality during LDP. To keep estimates on the over-estimation side (required property *p2*), in our model we keep the choices towards worst-case

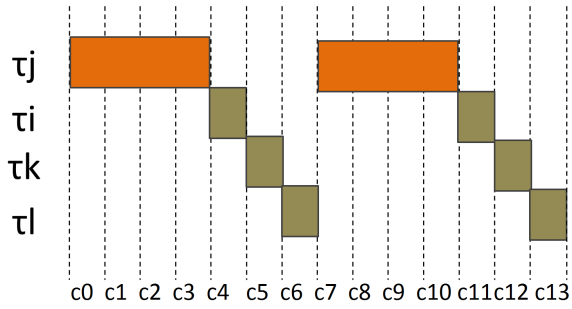

Fig. 5: Unfair cycle (fair slot) assignment with round robin

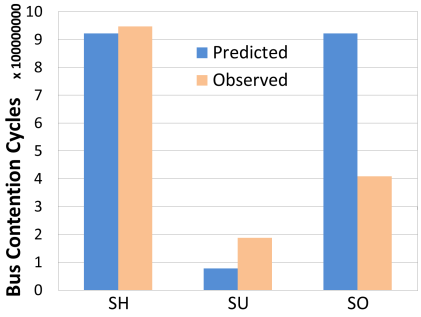

Fig. 6: Predicted and Observed cycles in controlled scenarios

estimation of [7]. As in [7], we do not use address traces and stick to execution profiles extracted from each application, which would significantly increase modelling time, which may preclude achieving principle *p2*. It is worth noting that the same assumption is done for other models used in late-design stages [16][8] since they also acknowledge the impossibility to get tight estimates to runtime interleave of requests.

B. Handling A2: similar request duration

Based on the discussion in Section III-B, the following situations can arise under a maximum load scenario. When all cores have requests of the same length, the model would be accurate. Instead, if τ_j requests tend to be shorter (longer) than the requests of its contenders, the model would tend to under-estimate (over-estimate). In order to validate this result we perform the following experiments.

Benchmarks. We use two resource stressing kernels [12] that create high contention in shared resources: `l2full` and `l2quarter`. These benchmarks traverse an array occupying the whole uL2 and a quarter of it, respectively. Both benchmarks have high bus utilization.

Experiments. We create three scenarios in which, τ_j requests take the same duration, longer and shorter than its contenders' requests, respectively. In all cases uL2 is partitioned across cores.

High-accuracy scenario (SH). We run four copies of the `l2full`. The lack of uL2 cache space causes all accesses to miss, hence creating bus requests with the same duration.

Underestimation scenario (SU). As τ_j we use `l2quarter` and as contenders we use 3 copies of `l2full`. `l2quarter` (τ_j) fits in uL2 causing bus hit requests ($l2h$). Its contenders exceed uL2 capacity resulting in misses, i.e. `l2m` bus requests.

Overestimation scenario (SO). We use `l2full` as τ_j and three copies of $l2quarter$ as contenders. As a result, contender tasks experience uL2 hits. Meanwhile, τ_j experiences misses since it exceeds cache capacity.

Results. Figure 6 shows the increase in execution time due to contention of τ_j (observed), in each of the three scenarios above. For each scenario we also report the results for BCM [7] (predicted). These results confirm our hypotheses. Furthermore, the amount of under-/over-estimation equals the ratio among the different duration of the requests of τ_j and its contenders, as shown in Equation 7.

$$\begin{aligned} SH: &\ \frac{cycles_{pred}}{cycles_{obs}} \simeq \frac{lat_{l2m}}{lat_{l2m}} \longrightarrow \frac{9.21E{+}08}{9.47E{+}08} \simeq \frac{24}{24} \\ SU: &\ \frac{cycles_{pred}}{cycles_{obs}} \simeq \frac{lat_{l2h}}{lat_{l2m}} \longrightarrow \frac{7.72E{+}07}{1.88E{+}08} \simeq \frac{10}{24} \\ SO: &\ \frac{cycles_{pred}}{cycles_{obs}} \simeq \frac{lat_{l2m}}{lat_{l2h}} \longrightarrow \frac{9.21E{+}08}{4.09E{+}08} \simeq \frac{24}{10} \end{aligned} \tag{7}$$

Proposed improvement: Equation 6 can be improved by adding a factor to account for the effect of requests duration. To that end, we add a request duration correction factor (*rdcf*) that, for a given task τ_j, is derived as the ratio of the average request duration of its contenders and the average duration of its own requests, see Equation 8.

$$rdcf_j = \left(\frac{1}{N_c - 1} \times \sum_{i=0, i \neq j}^{Nc-1} rd_i \right) / rd_j \tag{8}$$

rd_j is computed as shown in Equation 9, where Y is the set of instruction types; lat_y the latency of that request type; N_y^j the number of requests of that type that τ_j executes; and N_{total}^j the total number of requests executed by τ_j.

$$rd_j = \frac{1}{N_{total}^j} \times \sum_{y=1}^{y=Y} lat_y \times N_y^j \tag{9}$$

Overall Equation 6 is updated as follows:

$$\Delta t_j^{bus} = etb_{isol}^j \times \left(\frac{1}{avail_j} \right) \times rdcf_j \tag{10}$$

C. Handling A3: self bus contention

BCM [7] factors in the number of arbitration rounds τ_j waits until it gets the bus including the round when τ_j gets the bus. However, this last round is not actual contention, but τ_j's intrinsic latency. Therefore, we discard the arbitration round from the core under analysis by decreasing the number of arbitration rounds by 1. By doing so, we obtain the actual number of arbitration rounds due to contention. With this, the second multiplicand in Equation 6 is reformulated as follows.

$$\frac{1}{avail_j} - 1 \tag{11}$$

By combining Equation 10 and the expression above, our proposed BCM is as shown in Equation 12:

$$\Delta t_j^{bus} = etb_{isol}^j \times \left(\frac{1}{avail_j} - 1\right) \times rdcf_j \tag{12}$$

D. Generalization for fifo arbitration

Let $md_{bus^{roro}}$ and $md_{bus^{fifo}}$ be the maximum delay (measured in number of cycles) that a request can wait due to contention under *roro* and *fifo* arbitration. Interestingly, $md_{bus^{roro}} = md_{bus^{fifo}} = (Nc-1) \times l_{bus}$, where Nc is the number of cores and l_{bus} the latency of the bus.

Under high contention, with *roro* the grant is given to cores in a rotating fashion: $c_i, c_{(i+1)\%N_c}, c_{(i+2)\%N_c}, ...$ When the request of core c_i reaches the bus at an arbitrary time, the bus can be granted to any core. Hence, the request may wait zero cycles if the grant is given to c_i in the same cycle the request arrives; fewer cycles if the grant is given to c_{i-1}; or up to $md_{bus^{roro}}$ cycles if the grant is given to c_{i+1}. Thus, the range of latencies for a request varies from 0 to $md_{bus^{roro}}$.

With *fifo*, c_i requests wait for all pending requests. Under contention, c_i will usually wait for $Nc-1$ requests, though one of those $Nc-1$ requests can be partially processed when c_i request arrives, which makes c_i wait shorter. Hence, in contrast to *roro*, with *fifo* request waiting times are in the range $(Nc-2) \times l_{bus}$ to $md_{bus^{fifo}}$.

Since our model (on purpose) tends to over-estimate, it remains valid for both *roro* and *fifo* arbitration. As shown in next section, the degree of expected over-estimation is lower in the case of *fifo* since our model approximates maximum delay contention, and average contention for *fifo* is closer to the maximum.

V. Experimental Evaluation

In this section we evaluate the accuracy of the original BCM [7] (oBCM) and our proposed BCM (pBCM) in Equation 12 for a representative real-time multicore processor, the NGMP. Note that our model covers both *roro* and *fifo* policies, while the NGMP implements *roro*.

A. Experimental Setup

We used a detailed simulator of the NGMP. In the NGMP each core comprises private 16-KB 4-way set associative first level instruction and data caches (iL1 and dL1). The 4-way 256KB uL2 is partitioned so that each core receives a fair-share of the cache space. A round-robin arbitrated bus serves as bridge among the cores and the uL2. We validated the execution time accuracy of the simulator against a real board for the N2X implementation of the NGMP [5]. For the EEMBC automotive suite and reference space application our simulation platform reaches an accuracy higher than 97%.

Benchmarks. As reference applications (τ_j) we use the well-known EEMBC Automotive benchmarks [19] (aifftr-AF, aiifft-AT, bitmnp-BI, cacheb-CB, canrdr-CN, idctrn-ID, iirflt-II, puwmod-PU, rspeed-RS). We also use two representative benchmarks from the ESA. OBDP-OB contains the algorithms used to process raw frames coming from the state-of-the-art near infrared HAWAII-2RG detector [6]; and DEBIE-DE is the software that controls an instrument, which was carried on PROBA-1 satellite, to observe micro-meteoroids and small space debris by detecting impacts on its sensors, both mechanically and electrically.

As contenders we use the resource stressing kernels presented in Section IV-B: `l2full` (U) and `l2quarter` (H). We further use the following micro-benchmarks: `l2miss` (M) and `l1miss` (L), which continuously miss on uL2 and dL1 respectively; and `mixed-8-12-80` (E), which executes a specific mixture of instructions (8% stores, 12% loads and 80% adds).

We build 4-task workloads: for each reference application we generated eight workloads comprising three randomly selected resource stressing kernels each.

Metrics. To evaluate the accuracy of oBCM and pBCM we first measure accuracy in terms of bus cycles (i.e. Δt_j^{bus}). Then we measure models accuracy when predicting contention time (et_j^{muc}) when each BCM is integrated with the CCM [7]. In both cases we measure inaccuracy by comparing observed and predicted values as presented in Equation 13.

$$inacc = \frac{max(obs, pred)}{min(obs, pred)} \tag{13}$$

The rationale behind this metric is penalizing similarly over-estimations and under-estimations. Usual techniques, like Average Absolute Deviation or Root-Mean-Square Deviation, are meant to be used with experiments sharing a baseline. Error computations of absolute values offer limited meaning when assessing global accuracy across benchmarks. Furthermore, our metric grows linearly for both, over- and under-estimations. This allows us to fairly compare under/over-estimations. If, instead, we used $\frac{pred}{obs}$ as usual techniques do, then under-estimations would map in the range (0,1] whereas over-estimation would do it in the (1,∞) range. Such an approach produces asymmetric impact between over- and under-estimations, which typically leads to biased evaluation.

Our inaccuracy metric shows how distant the predicted value is w.r.t. the observed one. Interestingly, inaccuracy is always above one, with 1 representing no inaccuracy. This allows taking averages across all experiments preventing results from cancelling out each other when the predicted value is smaller than the observed one (underestimation) or vice versa (over-estimation), which would lead to misleading accuracy results.

B. Accuracy Results

Bus Accuracy Results(Δt_j^{bus}). The left part of Table I shows that pBCM drastically increases the percentage of workloads with over-estimated predictions (P.Over) w.r.t oBCM

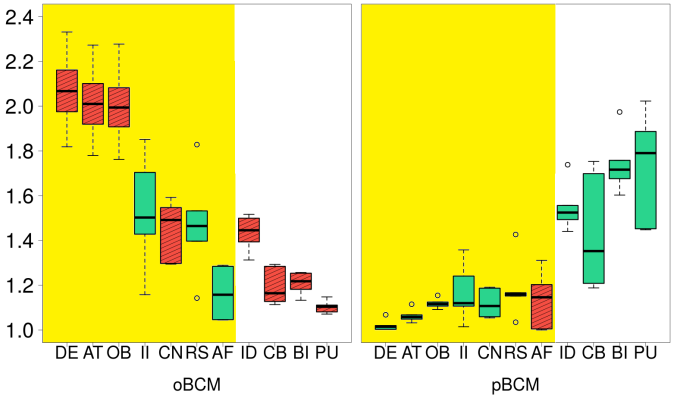

Fig. 7: Bus time prediction accuracy of pBCM and oBCM. Benchmarks sorted from best to worst improvement of pBCM w.r.t oBCM.

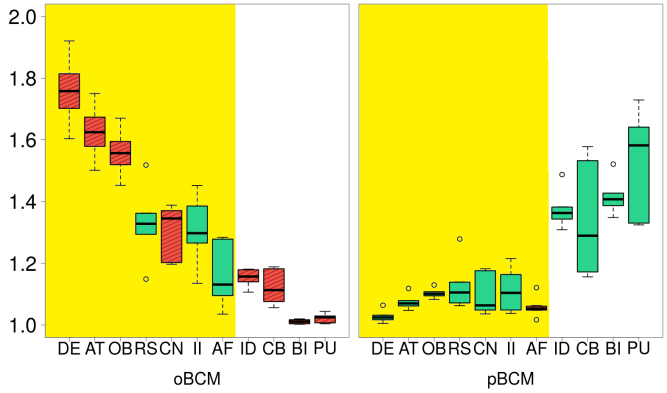

Fig. 8: Overall contention time accuracy of pBCM and oBCM. Benchmarks sorted from best to worst improvement of pBCM w.r.t. oBCM

(from 25% to 92%). We also see that the inaccuracy for those workloads of which pBCM over-estimates (I.Over) is smaller than that obtained by oBCM (from 1.39 to 1.28). Likewise, pBCM inaccuracy for under-estimated workloads (I.Under) also reduces (from 1.51 to 1.07).

Per-benchmark results are shown in Figure 7 with boxplots[2]. Red (green) boxes – used for those benchmarks for which on average the model under-estimates (over-estimates) – show that oBCM under-estimates for most benchmarks while pBCM over-estimates for all but one benchmark, achieving *p4*. In terms of accuracy pBCM leads to reduced inaccuracy in 7 out of the 11 benchmarks, improving the achievement of *p3*. Those benchmarks, highlighted with a yellow background, include the ESA benchmarks (OB and DE).

TABLE I: Percentage of workloads over-estimated and inaccuracy in over-estimated and under estimated workloads

	Bus (Δt_j^{bus})			Overall(et_j^{muc})		
	P.Over	I.Over	I.Under	P.Over	I.Over	I.Under
oBCM	25.0%	1.39	1.51	30.1%	1.20	1.28
pBCM	92.0%	1.28	1.07	95.5%	1.20	1.01

Multicore Execution Time Accuracy Results (et_j^{muc}). In the right columns of Table I and in Figure 8 we show the counterpart results in terms of overall contention time accuracy. We observe that the pBCM increased accuracy, translates into increased multicore time accuracy. pBCM results in more benchmarks over-estimated and an overall reduction in inaccuracy. Overall this shows that pBCM achieves *p3* and *p4* properties, described in Section II. Furthermore, pBCM maintains oBCM low-execution time overheads (*p2*) taking less than 0.2 seconds to execute. pBCM also keeps EP simplicity, not requiring any information coming from applications simultaneous (multicore) execution, hence achieving *p1*.

Fifo. In order to assess the accuracy of our model with *fifo*, we implemented it in our simulation infrastructure. We have observed that the difference among *roro* and *fifo* is small with an average execution time variation of 0,5% and *fifo* leading to longer execution times. This makes that the results in terms of accuracy are roughly the same for pBCM. As *fifo* tend to provide higher execution times (see Section IV-D) this results in our model reducing the over-estimation: the number of workloads with over-estimation remains roughly the same (96%) with the over-estimation for those benchmarks reducing in 1 percentage point.

pBCM internal evaluation. pBCM proposes two improvements over oBCM whose contribution to pBCM's accuracy we break down next. Our results show that failing to deal with the bus contention generated by the task under analysis (τ_j) explained in Section IV-C, i.e. implementing only the solution on Section IV-B, makes all bus time predictions to overestimate 73% on average, with some benchmarks reaching inaccuracies as high as 2.6x. This is caused due to the fact that oBCM factors in the cycles used by τ_j to access the bus in an isolation scenario, making predictions more pessimistic. By implementing the proposal in Section IV-C, we correct this problem resulting in more accurately predicted contention in the bus.

Model execution time overhead. The proposed changes introduce no extra overheads in the overall approach. Executing pBCM takes less than 0.2 seconds for each workload.

VI. Related Work

Several works target different timing analysis related aspects during EDP. Next we analyze several representative works. Some approaches focus on integrating timing in high level modelling environments such as Matlab/Simulink [9], before

[2]In the boxplot the bolded middle line represents the median of the *inacc* of τ_j across all 8 workloads. Two boxes expand comprising together 50% of the samples discarding outliers. The top and bottom line of the box mark the third (upper) and first (lower) quantiles. Following the main box are the whiskers with another 25% of samples each, ending at the minimum and maximum value that is not an outlier. Finally the little blank filled dots outside the whiskers are outliers and represent values that are greater or lesser than $3/2$ times the upper or lower quantile respectively.

the binary is generated. These approaches provide the developer timing information "as the code is written" [10], so the soonest possible. All these techniques focus on single-core architectures, while our focus is on multicore contention. Moreover, they require access to the code, which is often unavailable to end users.

Other proposals provide fast estimates by simplifying the timing model. To that end they use a mixture of measurements and modelling [15][13]. In [15] measurements are used to calibrate its models, and similarly [13] uses measurements and combines them with regression models to obtain their estimates. Other approaches use simple single-core processor designs that are more *WCET friendly* [10]. As before, these techniques focus on single-core architectures.

For multi-core processors, some methods can be used to obtain high contention measurements by using interference benchmarks at the bus level [12][20]. By replacing core contenders by dummy benchmarks that clog up the interconnection hardware, contention is maximized. Despite simple, these methods trade-off time-composability with tightness, so those measurements are usually far higher than the actual impact of contenders. As a result, they are not representative of real applications.

Some techniques help the designer deciding the processor best fitting system requirements [11][22]: the program is run on a parameterizable processor simulator (model) from which timing information is gathered. In our case the target processor is fixed, so such an approach is not necessary.

The work most related to ours is [7], which provides the first approach in multicore contention analysis during EDP, including cache and bus contention estimations. In our paper, we reuse the former and enhance the latter. We made a qualitative comparison of oBCM [7] and our proposal (pBCM) in Sections III and IV. We also performed a quantitative comparison in Section V showing that our pBCM outperforms oBCM in the relevant metrics for EDP.

VII. Conclusions

We made an in-depth analysis of the existing time-predictable bus model [7] for EDP and provided a new model that improves its accuracy. The proposed changes introduce negligible impact in the overhead of the simulation, making the overall approach remain lightweight – as required for EDP. Furthermore, our model does not need additional information to be stored in the EPs leaving the EP generation phase of the overall approach untouched.

In terms of accuracy, the proposed model reduces the number of workloads resulting in underestimation, from 75% to 8%. This reduces the risk of costly LDP changes. Our model also reduces overall inaccuracy figures for the bus from 1.48 to 1.25. As a result, overall multicore contention time is better predicted. Our model works for both round-robin and first-in first-out arbitration policies, providing high coverage of existing time-predictable policies.

Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation grant TIN2015-65316-P. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness and FEDER funds through grant TIN2014-60404-JIN.

References

[1] Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *ARP4761*, 2001.

[2] AUTOSAR. *Technical Overview V2.0.1*, 2006.

[3] Cobham Gaisler. *NGMP Preliminary Datasheet Version 2.1, May 2013*.

[4] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet + Users Manual*, 2011.

[5] Cobham Gaisler. *LEON4-N2X Data Sheet*, 2013.

[6] A. Jung et al. The H2RG infrared detector: introduction and results of data processing on different platforms. Technical report, European Space Agency, 2012.

[7] D. Trilla et al. Improving early design stage timing modeling in multicore based real-time systems. In *RTAS*, 2016.

[8] J. Jalle et al. Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP). In *8th European Embedded Real Time Software congress (ERTS)*, 2015.

[9] R. Kirner et al. Fully automatic worst-case execution time analysis for matlab/simulink models. In *ECRTS*, 2002.

[10] Trevor Harmon et al. Fast, interactive worst-case execution time analysis with back-annotation. *IEEE Trans. Industrial Informatics*, 8(2), 2012.

[11] C. Ferdinand et al. Integration of code-level and system-level timing analysis for early architecture exploration and reliable timing verification. In *ERTS2*, 2010.

[12] M. Fernández et al. Assessing the suitability of the ngmp multi-core processor in the space domain. In *EMSOFT*, 2012.

[13] J. Gustafsson et al. Approximate worst-case execution time analysis for early stage embedded systems development. In *SEUS*, 2009.

[14] M. M. Irvine and A. Dartnell. The use of emulator-based simulators for on-board software maintenance. In *Data Systems in Aerospace (DASIA)*, 2002.

[15] R. Kirner and P. Puschner. A simple and efficient fully automatic worst-case execution time analysis for model-based application development. In *Workshop on Intelligent Solutions in Embedded Systems*, 2003.

[16] J. Nowotsch et al. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *ECRTS*, 2014.

[17] M. Paolieri et al. *An Analyzable Memory Controller for Hard Real-Time CMPs* . Embedded System Letters, 2009.

[18] Zheng Pei Wu et al. Worst case analysis of DRAM latency in multi-requestor systems. In *RTSS*, 2013.

[19] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

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

[21] http://cache.freescale.com/files/dsp/doc/app_note/AN3735.pdf. *MSC8122: Avoiding Arbitration Deadlock During Instruction Fetch*, 2008.

[22] http://www.absint.com/timingprofiler. *Timing Profiler*. AbsInt.

[23] http://www.gmv.com/en/aeronautics/products/air/. *Robust Partition Safety-Critical Real-Time Operating System*. GMV.

[24] www.st.com/resource/en/datasheet/spear1340.pdf. *ST Dual-core Cortex A9 HMI Datasheet*, 2012.